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.

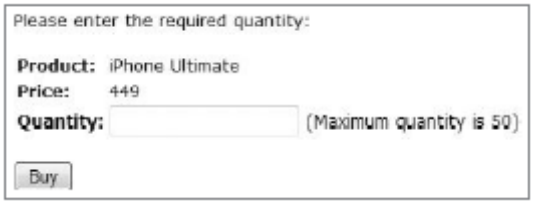
* Jeśli aplikacja jest wdrożona na kilku różnych serwerach, a użytkownicy mogą wchodzić w interakcję z więcej niż jednym serwerem w celu wykonania wieloetapowej akcji, udostępnianie danych po stronie serwera między hostami, które mogą obsługiwać te same żądania użytkownika, może nie być proste. Wykorzystanie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

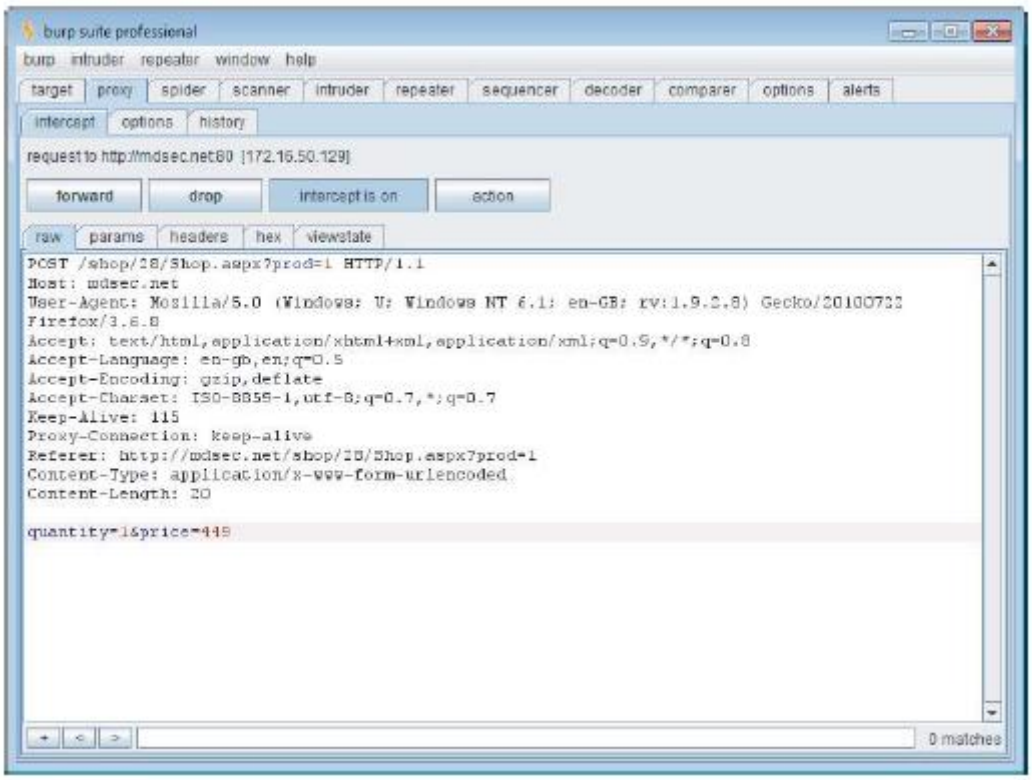
Content-Type: application/x-www-form-urlencoded

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:

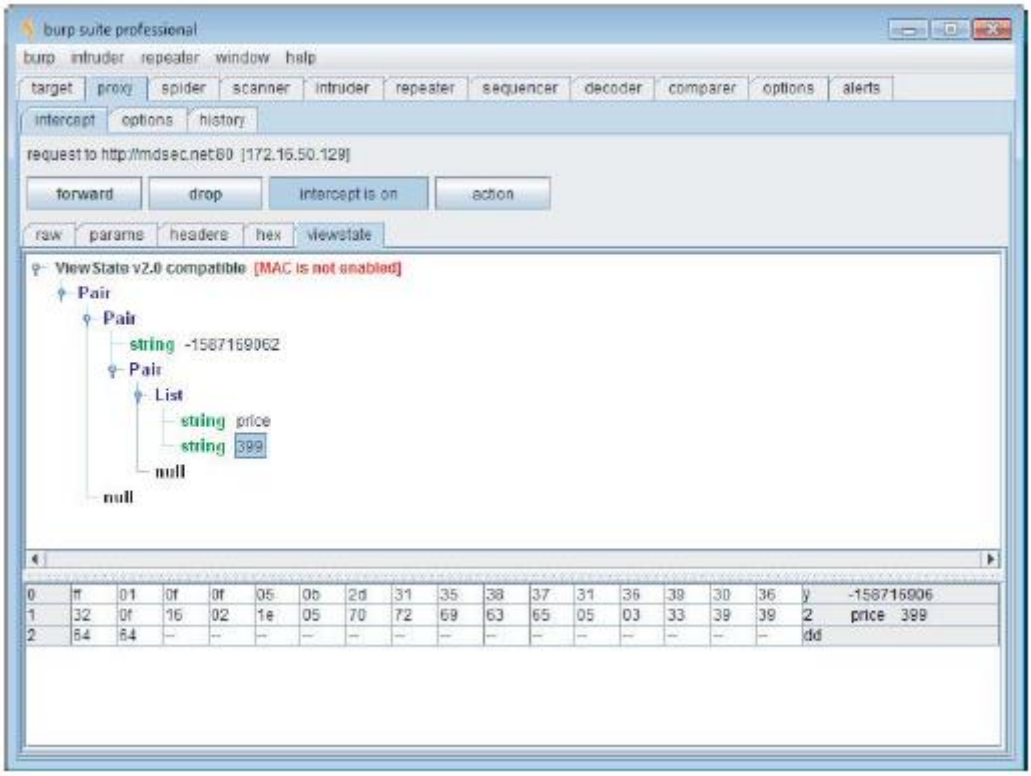
3D FF 01 0F 0F 05 0B 2D 31 35 38 37 31 36 39 30 ; =ÿ.....-15871690

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

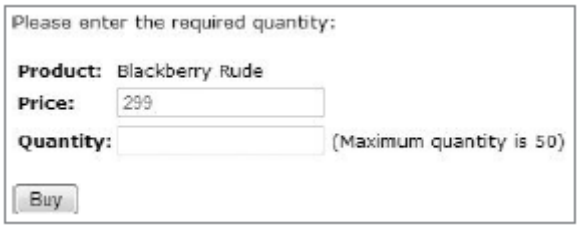
Price: <input type=”text” disabled=”true” name=”price” value=”299”>

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

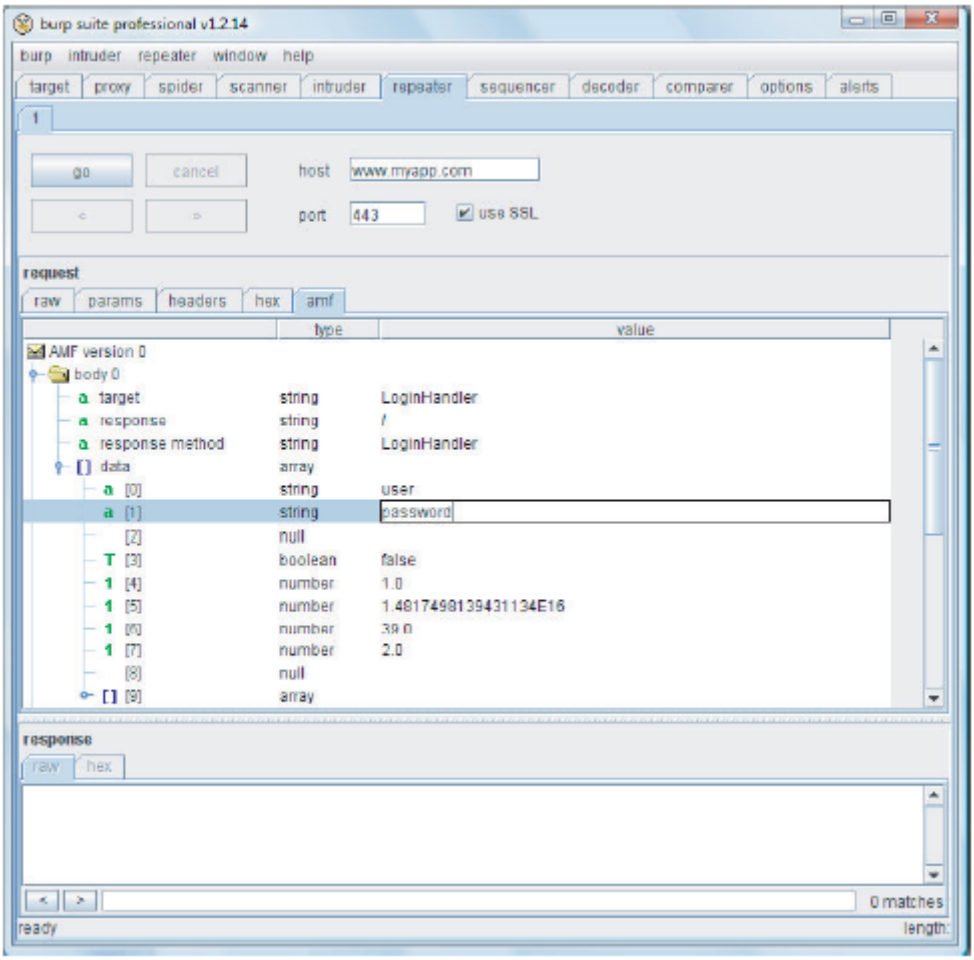
Po przechwyceniu nieprzetworzonych serializowanych danych za pomocą serwera proxy można je deserializować za pomocą samej Javy, aby uzyskać dostęp do zawartych w niej elementów danych pierwotnych. DSer to przydatna wtyczka do Burp Suite, która zapewnia platformę do przeglądania i manipulowania serializowanymi 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:

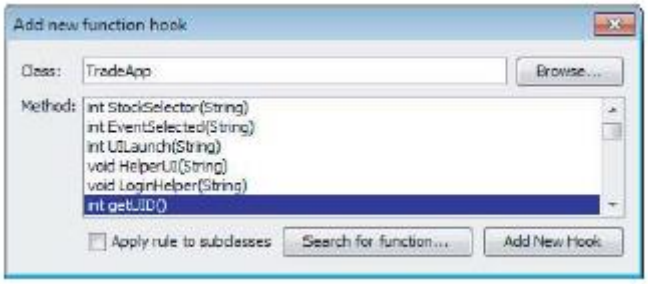
```
C:\tmp>javac CheckQuantity.java
C:\tmp>java CheckQuantity
4b282c510f776a455d425a7808015c555f42585460464d1e42684c414a152b1e0b5a520a
145911171609
```

Nasz zmodyfikowany komponent wykonał teraz niezbędne zaciemnianie naszej dowolnej liczby 999. Aby przeprowadzić atak na serwer, musimy po prostu przesłać formularz zamówienia w normalny

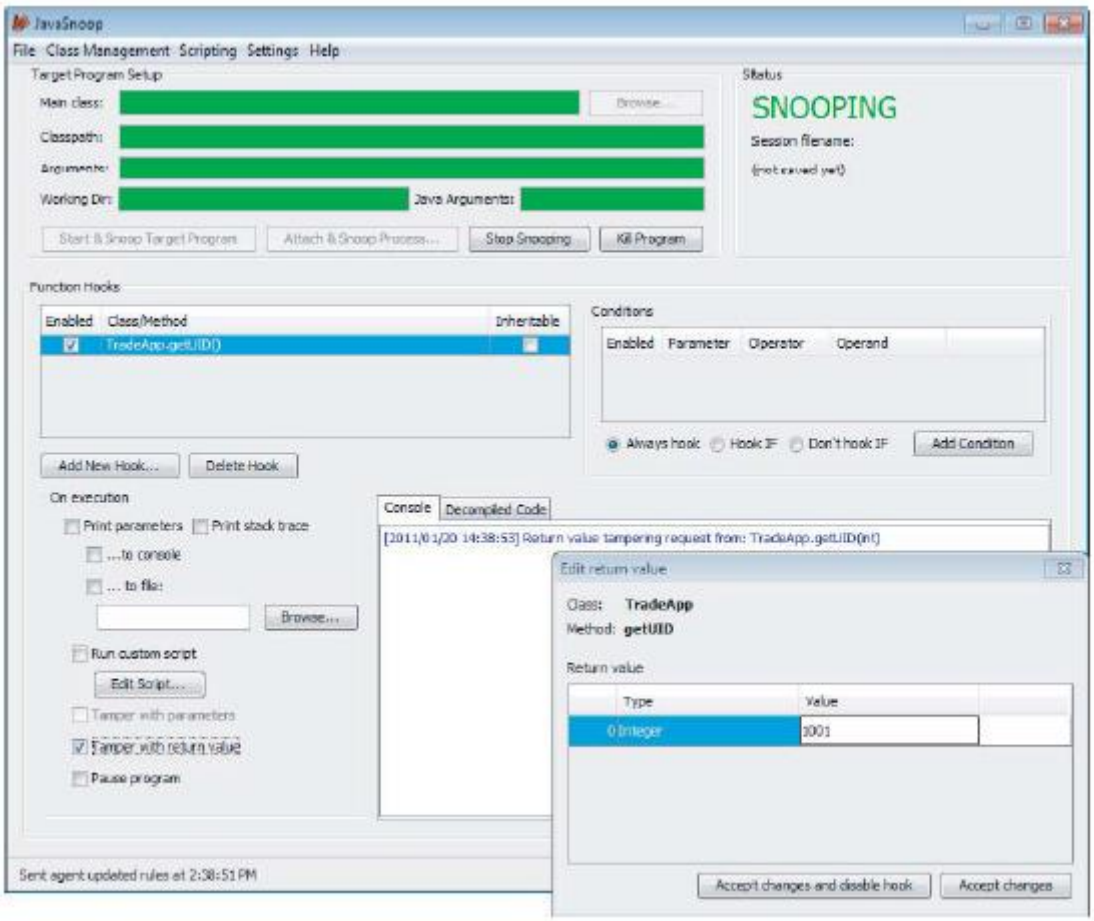
sposób, używając prawidłowych danych wejściowych, przechwycić wynikowe żądanie za pomocą naszego serwera proxy i zastąpić zaciemnioną ilość z tą dostarczoną przez nasz zmodyfikowany komponent. Należy pamiętać, że jeśli aplikacja wysyła nowy blok zaciemniający za każdym razem, gdy ładowany jest formularz 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?