Enovise Challenge 2 CTF Write-up

~l3s7r0z

Intel Gathering

On booting up the Virtual Machine, in this case I am

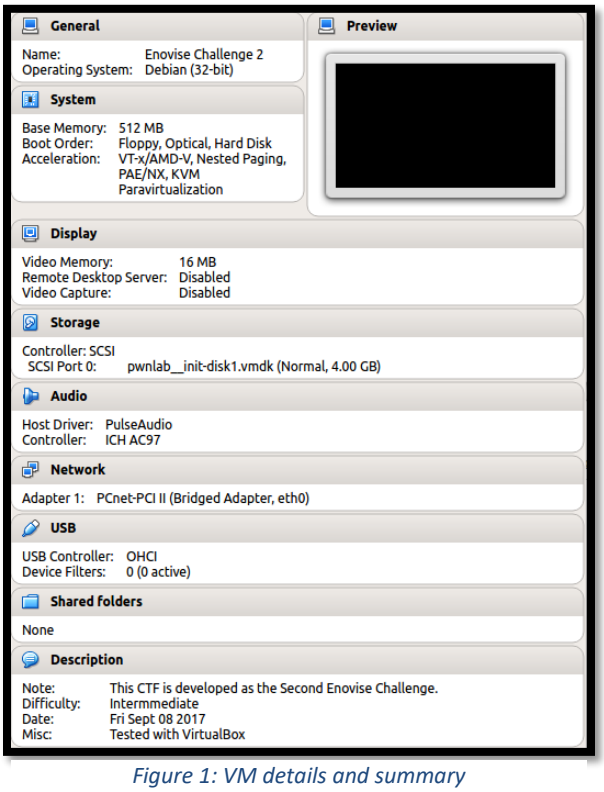

Figure 1: VM details and summary

using VirtualBox[1], I learn that it is a 32-bit Debian[2] installation, with the challenge difficulty set to intermediate.

I launched angry IP scanner (ipscan[3]) and identified a new IP within my network (192.168.0.67) as shown below:

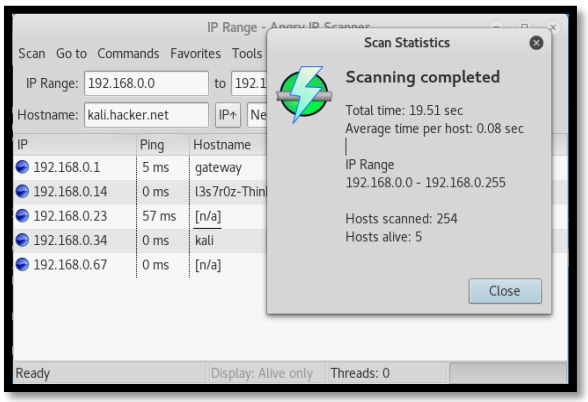

Figure 2: Range scan with angry IP scanner

I then performed an nmap[4] scan of the target in order to scan all the ports from 1-65535 (using the –p 1-65535 option) and:

a) Identify the service and operating system versions using the –sV switch
b) Perform an aggressive timed scan by making use of the –T4 switch
c) Perform a TCP scan using the –sS switch

Despite the –T4 switch intended to perform an aggressive scan, the results took quite a while to return, so patience was key.

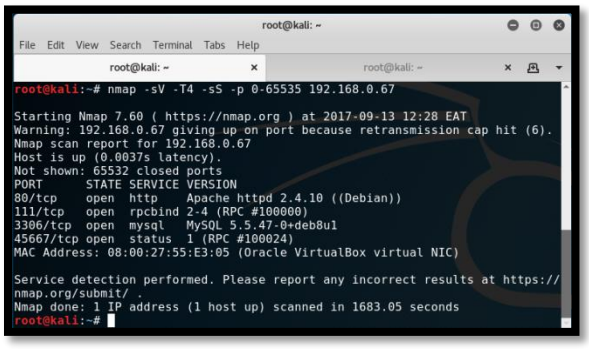

Figure 3: nmap enumeration for service version

[1] https://www.virtualbox.org/
[2] http://www.debian.org/
[3] http://angryip.org/
[4] https://nmap.org/

Through nmap, I was able to determine that port 80 was open, so visiting it through the browser the following is observed:

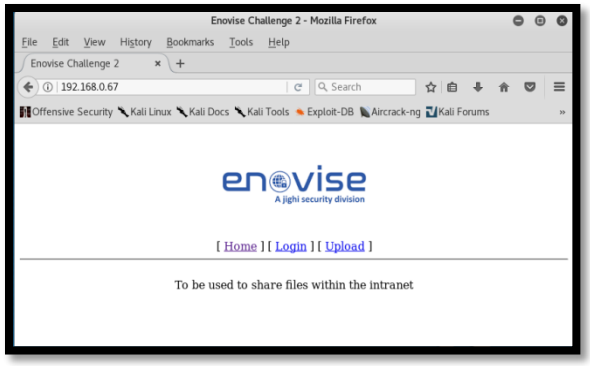

Figure 4: Port 80 details on browser

I decided to run nikto[5] against the web server to see what I would identify.

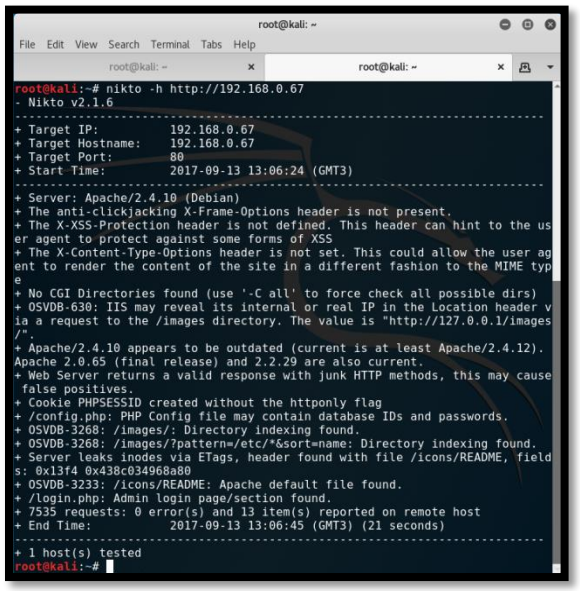

Figure 5: nikto web server scan results

A web server!! Good stuff! I clicked on the buttons to see the URL behavior, clicking on the "Login" button changes the URL to http://192.168.0.67/?page=login which looks likely vulnerable to LFI[6].

Injection Point Identification

We can see that **/config.php** and **/login.php** might contain database IDs, passwords and a login page respectively, so we visit these and see the response on the browser.

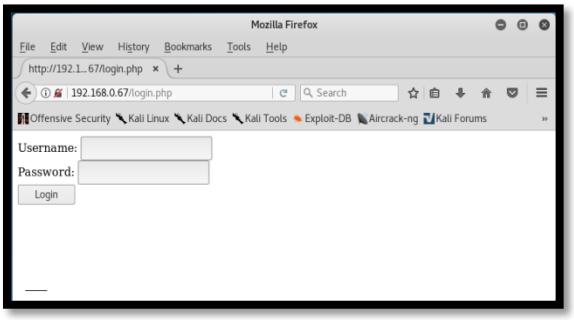

Figure 6: Login form

The **/login.php** page contains a username and password field and loading the **/config.php** shows a blank page.

After attempting SQLi tests on the text fields unsuccessfully, I decided to go back to that URL and test for LFI. I tried the following:

http://192.168.0.67/?page=/etc/passwd

http://192.168.0.67/?page=/etc/passwd

http://192.168.0.67/?page=../../../../../../../etc/passwd

http://192.168.0.67/?page=../../../../../../../etc/passwd

But they never worked, so, after googling[7], I discovered that I could try some php filter manipulation as below, and it worked:

http://192.168.0.67/?page=php://filter/convert.base64-encode/resource=index

[5] http://sectools.org/tool/nikto/

[6] https://www.acunetix.com/blog/articles/local-file-inclusion-lfi/

[7] http://kaoticcreations.blogspot.co.ke/2011/12/lfi-tip-how-to-read-source-code-using.html

Exploitation

As seen below, I was able to bypass the execution of the **/index.php** page by using PHP filters.

Figure 7: Index page information disclosure

This allowed me to obtain the contents of the **/index.php** file, encoded in base64.

Decoding[8] this resulted in PHP code that revealed improper sanitization of user input.

```
1. <?php
2. //Multilingual. Not implemented yet.
3. //setcookie("lang","en.lang.php");
4. if (isset($_COOKIE['lang']))
5. {
6.         include("lang/".$_COOKIE['lang']);
7. }
8. // Not implemented yet.
9. ?>
```

In order to prevent these types of attacks, the line 4 above should have been coded this way:

```
$_cookie = str_replace('../','',$_GET['file']);
```

Since it is not, I decided to use burpsuite[9] to change the cookie and attempt to load a file within the system. Notice the cookie, I change it to a file within the file system (in this case **/etc/passwd**).

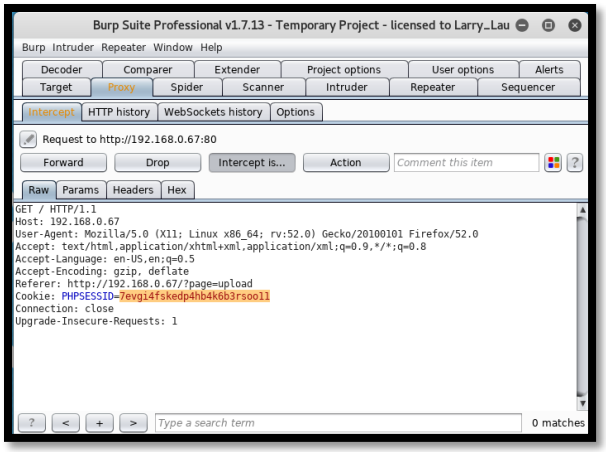

Figure 8: Index page cookie examination

Note: I change the cookie from:

PHPSESSID=7evgi4fskedp4hb4k6b3rsoo11

To become:

lang=../../../../../../etc/passwd

Replacing the cookie value I was able to view the contents of /etc/passwd as shown below:

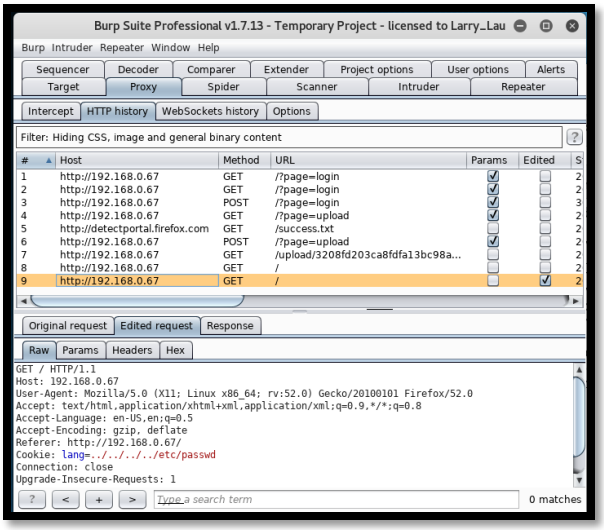

Figure 9: Local inclusion of /etc/passwd

I carefully thought of the most ideal method of exploiting this LFI vulnerability, I needed a means of uploading a payload.

[8] https://www.base64decode.org/
[9] https://portswigger.net/burp/

Recalling that nikto had earlier on identified a **config.php** file, I attempted to load the file to attempt to obtain the configurations (in case of any).

I abused the php filters to load /config.php as shown below:

http://192.168.0.67/?page=php://filter/convert.base64-encode/resource=config

That results in a base64 hash, which I then decoded to display the clear text PHP code. See below:

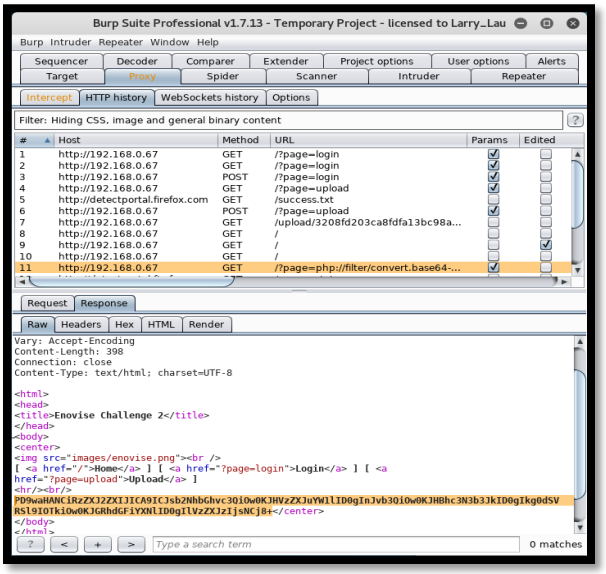

Figure 10: Base64 hash of /config.php

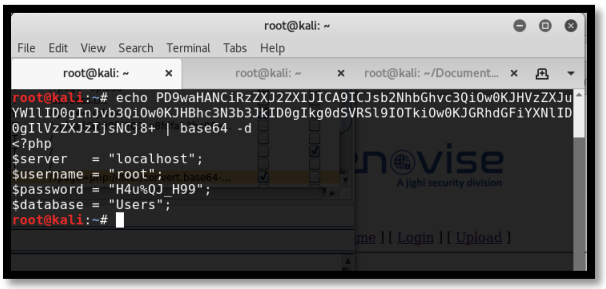

Figure 11: MySQL connection details

Sure enough, these were MySQL database connection details. Trying them out, I was able to successfully login to the database instance. While inside, I managed to identify the database along with the table and data within the table (users along with their hashed passwords – again in base64).

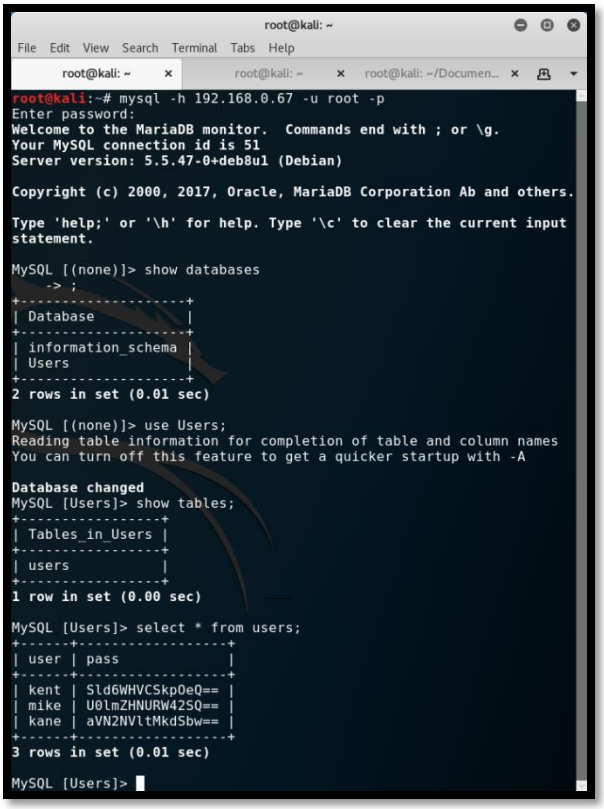

Figure 10: Remote MySQL database administration

I decrypted the passwords and attempted to log into the login interface using the login button within the **/index.php** page.

The decrypted passwords were as shown below:

user	pass
kent	JWzXuBJJNy
mike	JWzXuBJJNySIfdsTEn6I
kane	iSv5Ym2GRo

Table 1: System users clear-text passwords

Logging in with the username kent and his corresponding password worked well. The upload functionality did not allow me to upload a php reverse shell (with a **.php** file extension), so I had to convert it into an image, or rather, make it look similar to one. A little googling helped.

I altered the beginning of the php payload[10] to read **GIF98** and uploaded it.

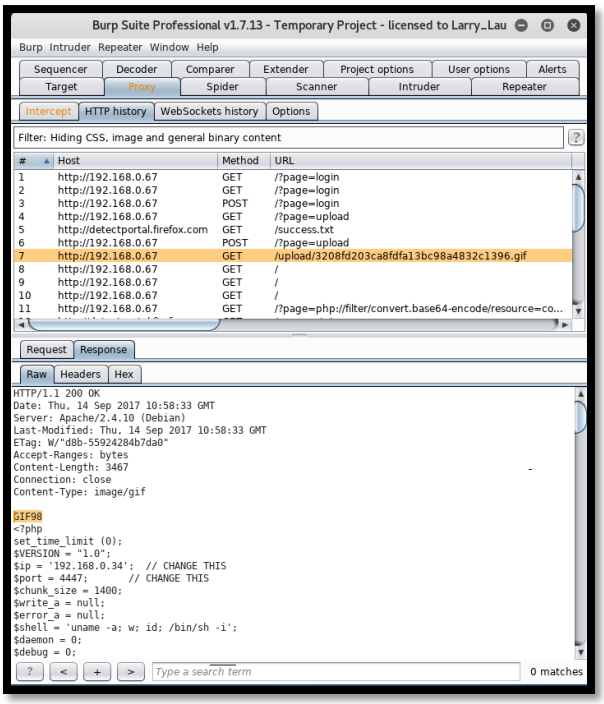

Figure 13: Php payload upload

Uploading the payload went well but then I had to identify the location where it had been uploaded to.

To do this, I navigate within the Proxy section of burp and go to the HTTP history section where I can clearly see the location that the uploaded payload went to. As shown below, this was the "**/upload/**"directory.

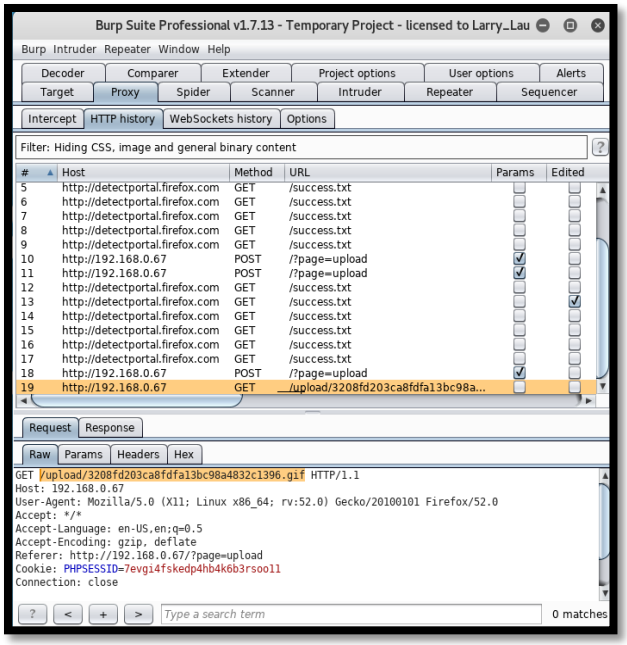

Figure 14: Payload location

I had to have a netcat[11] listener running before visiting the path and activating the payload.

Doing that allowed me to receive a reverse connection to my local machine.

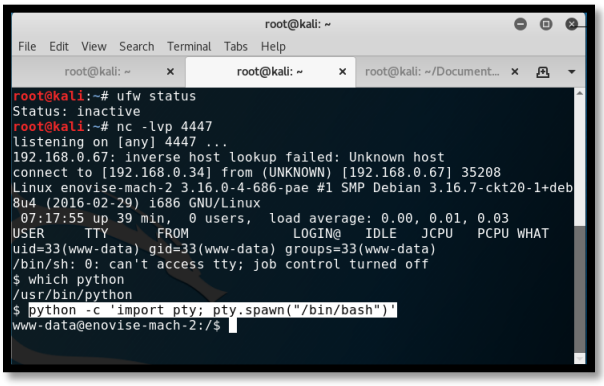

Figure 15: Reverse connection

I was now inside the machine, and had to escalate privileges to root.

[10] http://pentestmonkey.net/tools/web-shells/php-reverse-shell

[11] http://netcat.sourceforge.net/

Notice that the shell initially received has “job control” turned off. That means that I cannot manipulate jobs (check their running status, push them to the background or foreground) or even use utilities such as **ssh** or **su**. To gain a tty[12] shell, I run the command below:

python –c ‘import pty; pty.spawn(“/bin/bash”)’

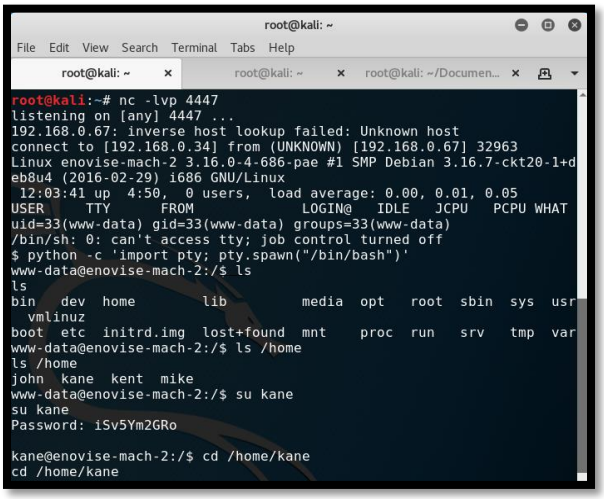

Figure 16: Login as user kane

I tried to log into the three users within the system with the clear-text passwords obtained from the base64 decryption. User kane seemed to have some interesting stuff within his home folder.

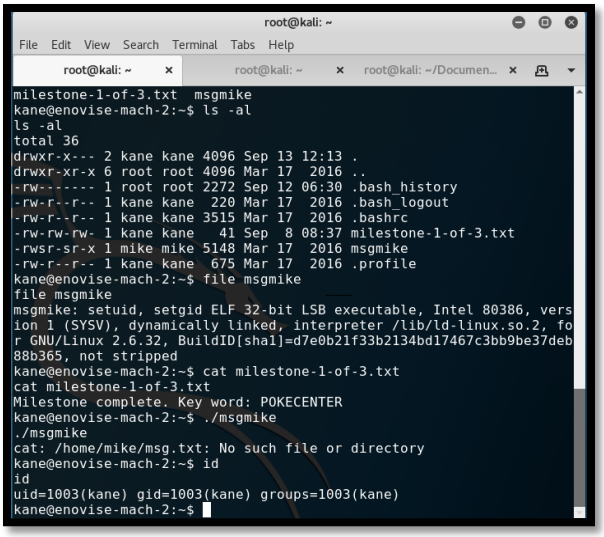

Figure 17: Exploration as user kane

I was able to obtain the key word within the first milestone as **POKECENTER** but noticed something interesting while examining the file **msgmike**.

I observed that msgmike was an executable file with the SUID[13] and GUID bits set. This executable also seemed to write to user mike’s home directory – into a file called msg.txt.

However, the executable complains that it never found the “cat” command. I decided to get creative and create my own “cat” script that could allow me to drop into mike’s home folder by invoking the bash[14] program.

[12] http://netsec.ws/?p=337

[13] http://www.linuxnix.com/suid-set-suid-linuxunix/

[14] https://en.wikipedia.org/wiki/Bash_%28Unix_shell%29

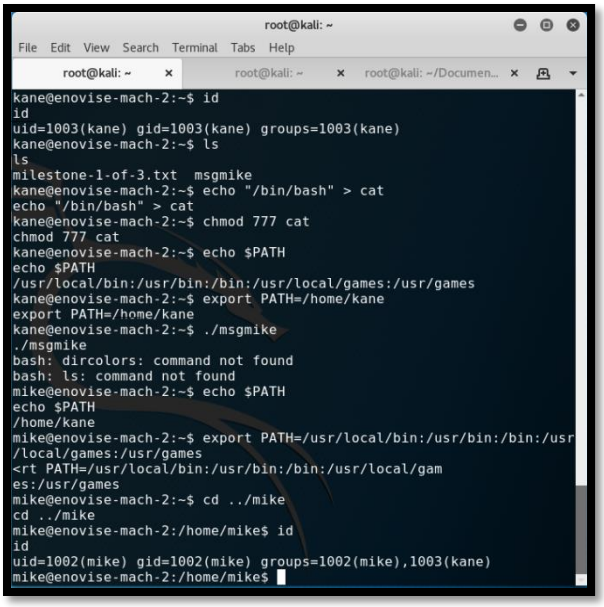

Figure 18: Elevating to user mike

I create a script to allow me execute a shell as user mike, altering my PATH[15] variable accordingly.

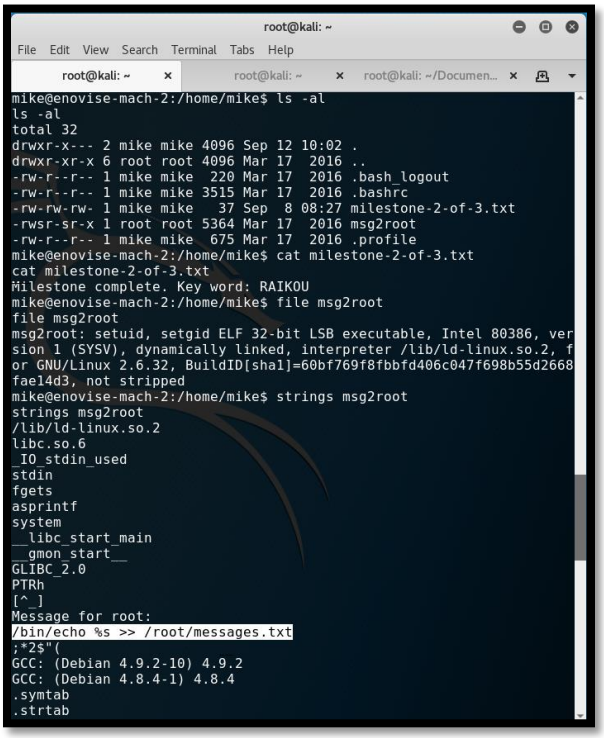

Figure 19: Exploration as user mike

While under mike's home folder, I discover the second milestone and cat the file contents. The second key word I find is **RAIKOU**. I also notice an executable within the directory called msg2root. I run a strings command on it and discover some text to allege that it attempts to use the echo command to append messages to a text file within the root directory.

I think of a means of abusing this, and after several attempts, discover that I can pass a ";" in between commands, and by doing so, pass a "/bin/sh" that drops me onto yet another shell, but this time with root[16] privileges. This is shown below:

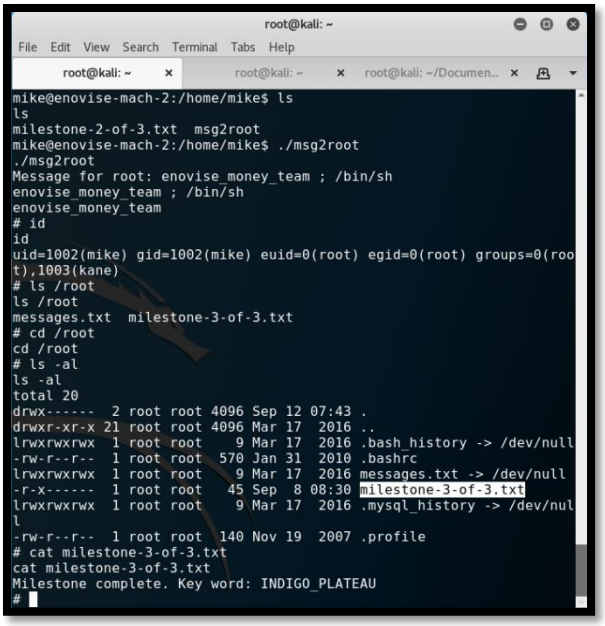

Figure 20: Elevating to root

I obtain the third key word as being **INDIGO_PLATEAU** by performing a cat on the third milestone file.

At this point I can confirm that I am root by performing an "id" and "whoami" to verify my root-ship.

[15] http://www.linfo.org/path_env_var.html

[16] http://www.linfo.org/root.html

Enovise Challenge 2 Quiz

1. What is the MySQL clear-text root password? – (10 Marks)
   H4u%QJ_H99
2. What is user kent's clear-text password? – (10 Marks)
   JWzXuBJJNy
3. What point at the PHP code does the vulnerability lie? – (20 Marks)
   include("lang/".$_COOKIE['lang']);
4. What is the key word in the first milestone? – (20 Marks)
   kane@enovise-mach-2:~$ cat milestone-1-of-3.txt
   cat milestone-1-of-3.txt
   Milestone complete. Key word: POKECENTER
5. What is the md5 hash for the file milestone-1-of-3.txt? – (5 Marks)
   kane@enovise-mach-2:~$ ls milestone-1-of-3.txt | md5sum
   ls milestone-1-of-3.txt | md5sum
   6dccfee257efcea701e52d8996cdfa0b
6. What is the build id for the file msgmike? – (10 Marks)
   d7e0b21f33b2134bd17467c3bb9be37deb88b365
7. What is the keyword within the file milestone-2-of-3.txt? – (20 Marks)
   mike@enovise-mach-2:/home/mike$ cat milestone-2-of-3.txt
   cat milestone-2-of-3.txt
   Milestone complete. Key word: RAIKOU
8. What is the build id for the file msg2root? – (10 Marks)
   60bf769f8fbbfd406c047f698b55d2668fae14d3
9. What is the key word within the milestone-3-of-3.txt? – (20 Marks)
   bash-4.3# cat milestone-3-of-3.txt
   cat milestone-3-of-3.txt
   Milestone complete. Key word: INDIGO_PLATEAU
10. What is the password hash for the user root? – (15 Marks)
    # cat /etc/shadow
    cat /etc/shadow
    root:6aYZMZ3V0$qAYwiR7aanVmKSWyV5IbRffspdjFx4xhLrm8kbHhh1DG16Bdb0/ptImcDK2uT.6xc/FZotacYr0X4dB0SurjD/:16877:0:99999:7:::

Conclusion

The challenge was great and it taught me various skills, especially, exploiting PATH misconfigurations and using burpsuite to tamper requests so as to abuse php filters, thus taking advantage of improperly sanitized php functions.

Credit

[i] This CTF is based on the PwnLab CTF found on Vulnhub.

[ii] Credit to @claor for designing such an interesting machine for beginner CTF players.

---

[i] https://www.vulnhub.com/entry/pwnlab-init,158/
[ii] https://www.vulnhub.com/author/claor,331/