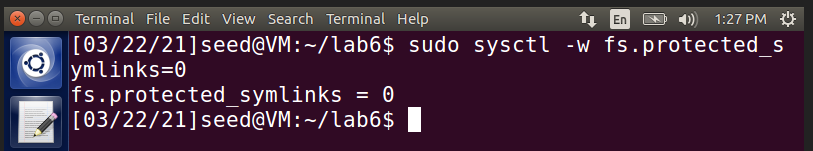
**ASSIGNMENT – 6**

Name: **Sudharsan Srinivasan**

UTA ID: **1001755919**

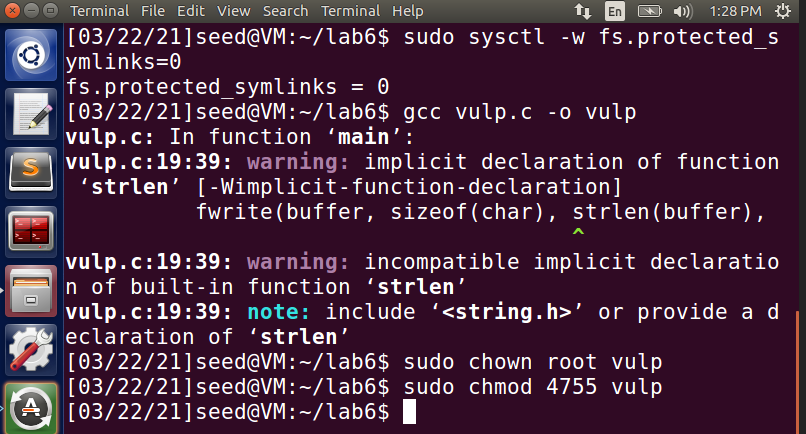
**Initial Setup:**

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



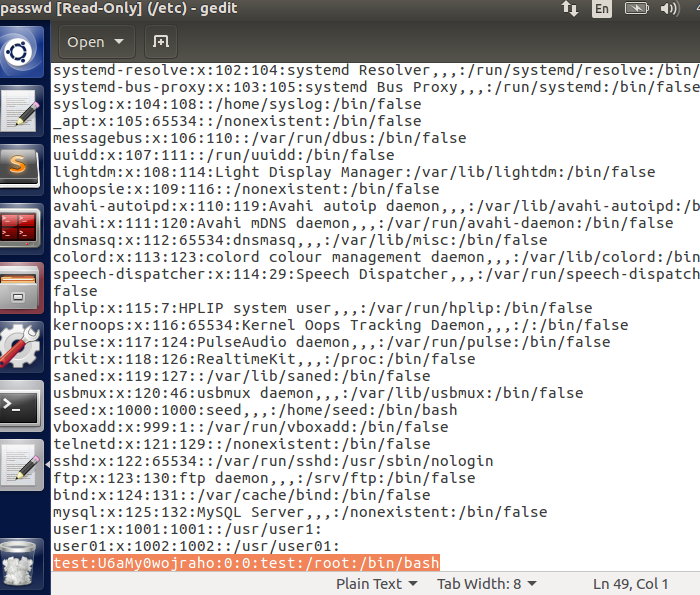
**Compiling and running Vulnerable program:**

* 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.

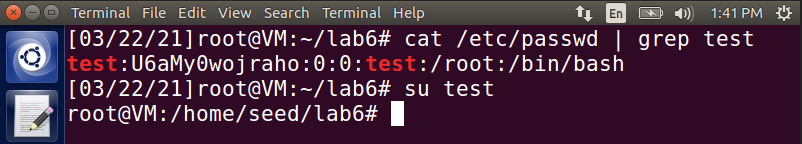


**Task 1 – Target Choosing:**

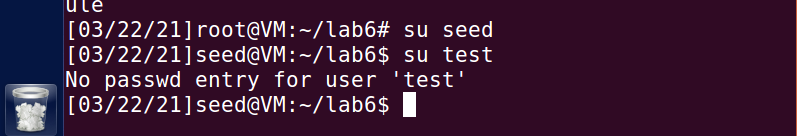
* The aim of this task to create a test user in the passwd file and use the user account to gain root access, which can be achieved using the vulnerable race condition program. For this, the passwd file is modified to add a test account.
* The passwd file can be found in /etc/passwd path and it can be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



* 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.



* 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.



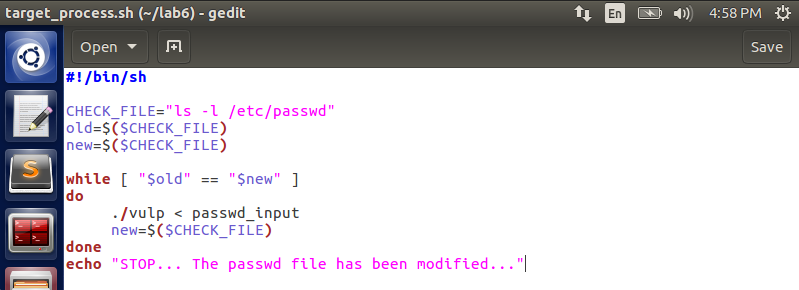
* Since the entry has been deleted, it can be seen that there is no password entry for ‘test’ account.

**Task 2 – Race Condition Attack Launch:**

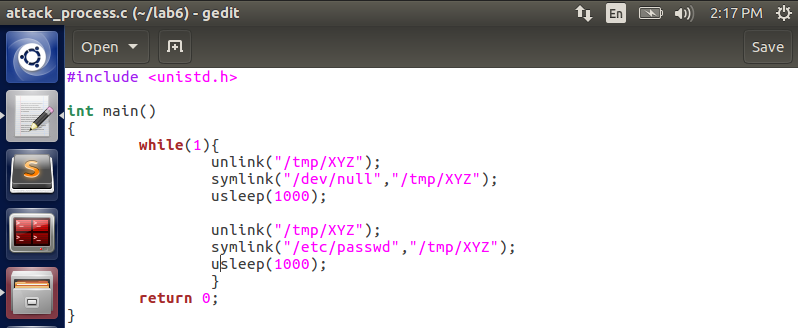
* The main thing we would like to accomplish in this task is to make use of the vulnerability in the race condition of the vulnerable program and get root access privileges. This can be achieved in 3 different ways.

**2.1 - Full version:**

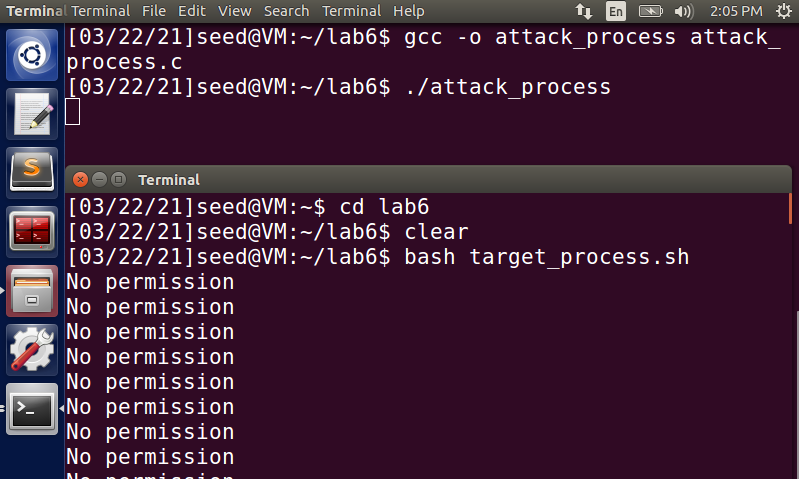
* This is done by setting up two processes, **attack\_process** & **target\_process** and run them in two separate terminals. There is also a **passwd\_input**  file created, inside which the test account and its password to be updated to the **/etc/passswd**  file is stored.



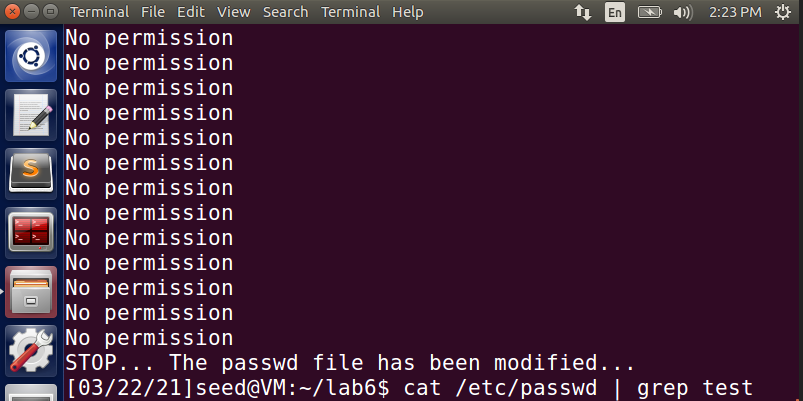
* In the above target\_process bash file, the password is fetched from the input file. Meanwhile, attack\_process file is set up in such a way that it points to the passwd file which we are trying to modify.



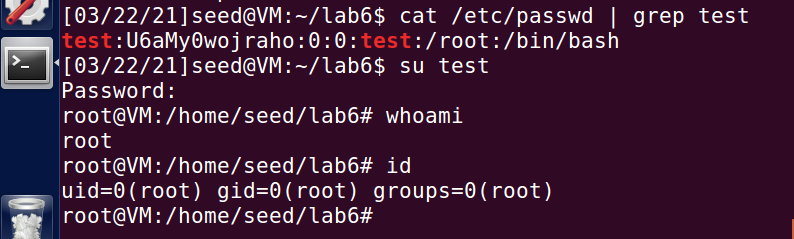
* In the image seen above, we see that /tmp/XYZ is pointed to /etc/passwd because that is the file we need access to modify the contents of it and append it with the input fetched from the input file, through the target\_process bash.
* **unlink** command is used to delete any old links that exists there. **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.



* The attack\_process file is compiled and run to try to launch the attack from one terminal. Meanwhile on the other terminal, the target\_process bash file is run using **bash** command.



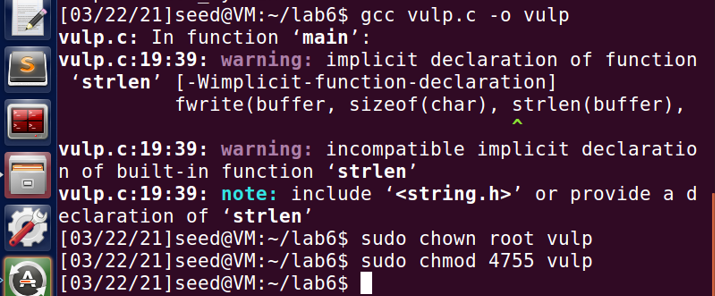
* The bash file stops when the passwd file has been successfully modified. The **cat** command that follows, is used to check if the entry is present in the passwd file.



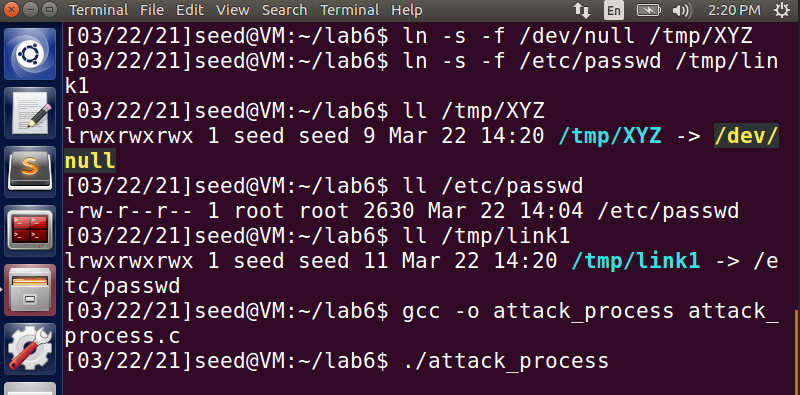
* As seen above, the test account entry is present in the passwd file. Also to be noted is that, if switched to the test account, it has root privileges, thus showing that the exploit on the vulnerability is successful.

**2.2 – Slow Deterministic version:**

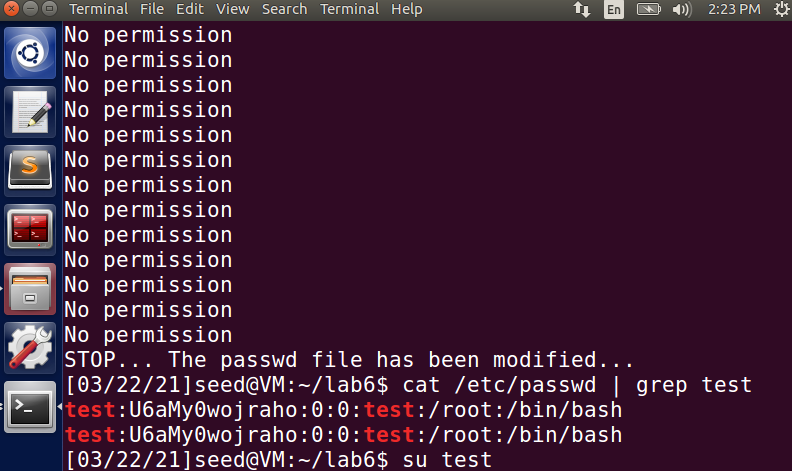
* In this version, sleep () is added to vulp.c program and it is recompiled. The use of this sleep () function is that, it pauses the execution and transfers the control to OS.



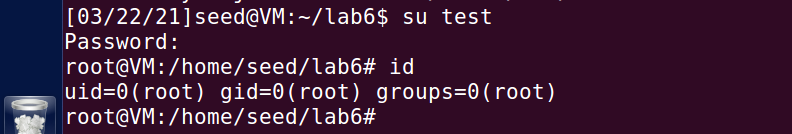
* During this sleep time, we will have to make the /tmp/XYZ point to our /etc/passwd file and then re-run the bash to see if we are able to get an entry into the passwd file.
* The indication to /etc/passwd means that passwd is the file that we are trying to modify. Then the **attack\_process** is compiled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



* On the other terminal, we run the **target\_process** bash, to see if we are able to modify contents of passwd file.

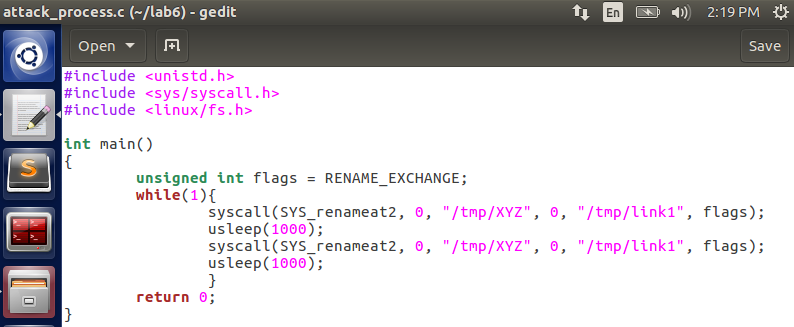


* The passwd file has been modified with another entry of test account, this time done using the slow deterministic version. On checking the privileges of the new test account created, it has root privileges indicated by **#**, meaning the attack succeeded.

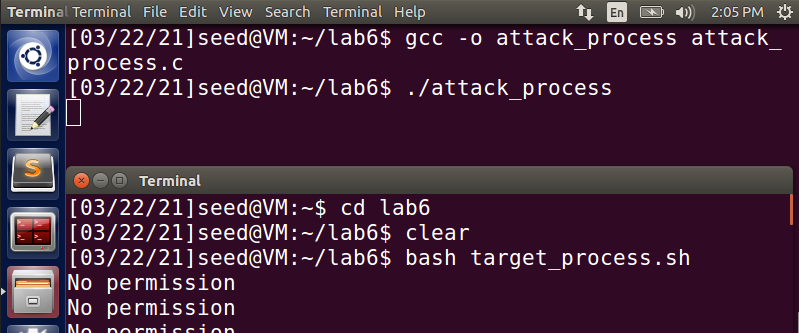


**2.3 – Improved Attack Method:**

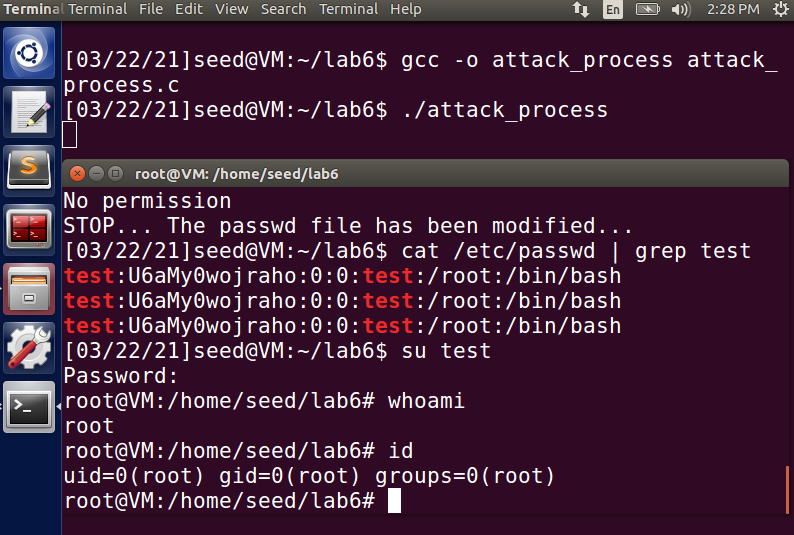
* The problem with the previous two methods is that if the target passwd file is opened before the **symlink** command is executed, then the file cannot be modified. This is because symlink points to the passwd and without it getting executed, we will not know the destination file to write into, thus 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.



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

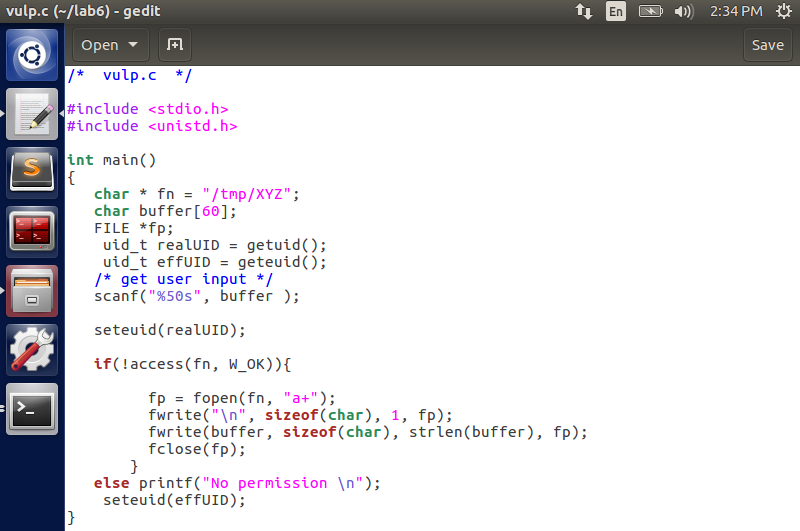


* As evident from the below image, the file has been modified using the vulnerability and root access privileges is also obtained for the test account, indicated by **#** proving a successful attack.

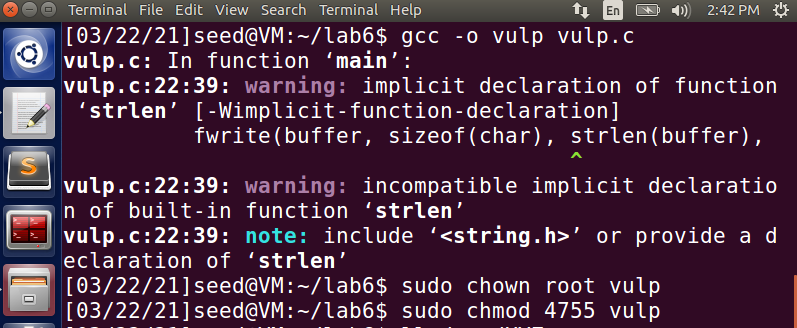


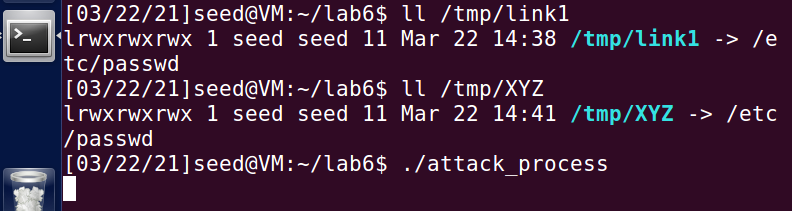
**Task 3 – Principle of Least Privilege – Countermeasure:**

* 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.

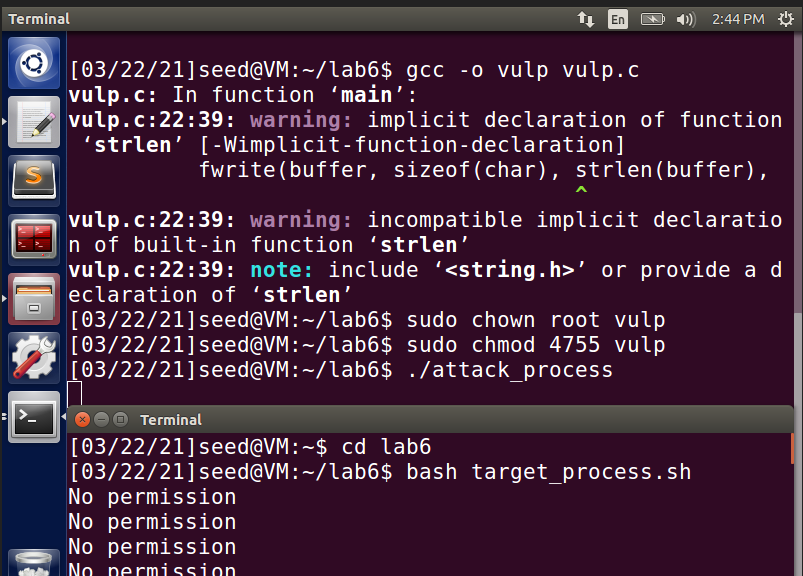


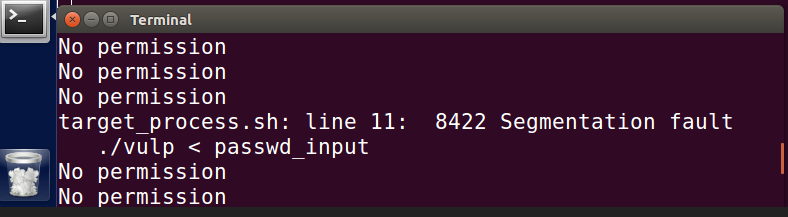
* In the above program, it can be noted that set UID is set to real UID before fopen () happens, typically before the passwd file is opened.
* Now, this updated **vulp.c** code is run and given root access to the compiled file, done in terminal 1. Then the attack\_process is run.





* 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.**

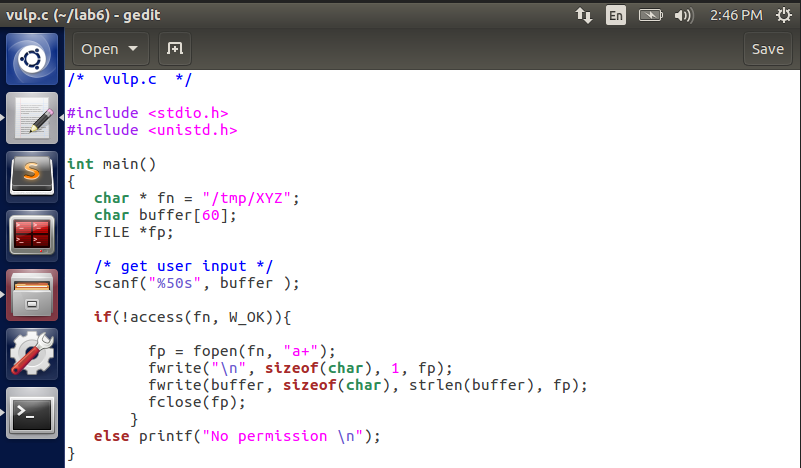




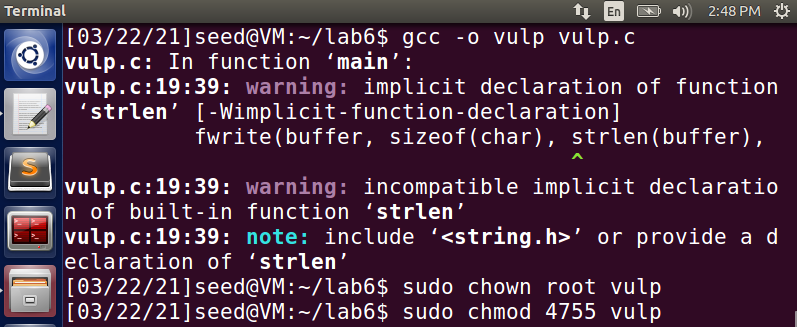
* The explanation for this issue is that we set the effective UID to be the same as that of real UID during execution time. Since we are running everything from the seed account, the real UID will be that of the seed account, during run time and that is stored in effective UID. The seed account does 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:**

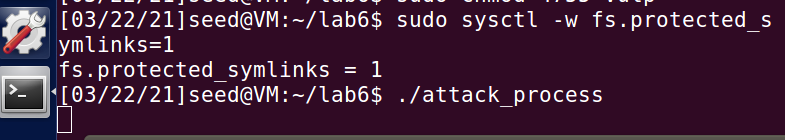
* To try this out, the changes made in the previous task are reversed and the vulp program is left as it was earlier for Task 1.



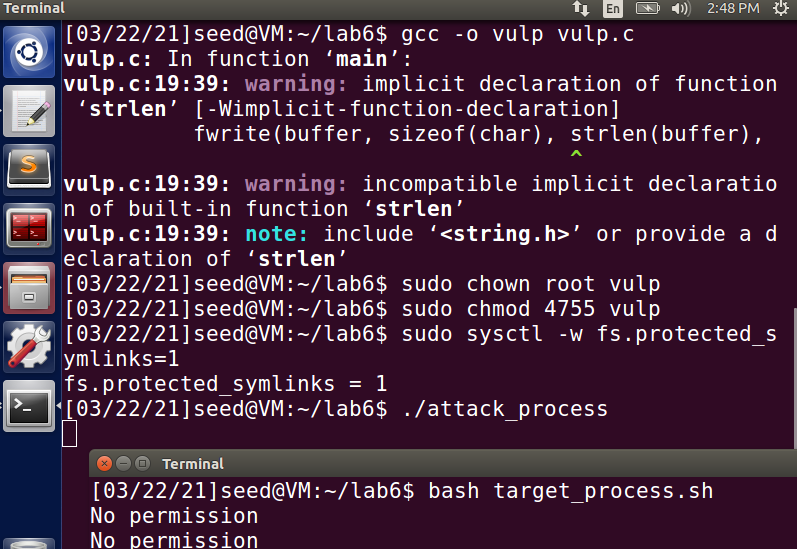
* Now, this updated vulp program is compiled into vulp file and root access privileges are given to it through the **chmod** and **chown** commands as shown below for reference.

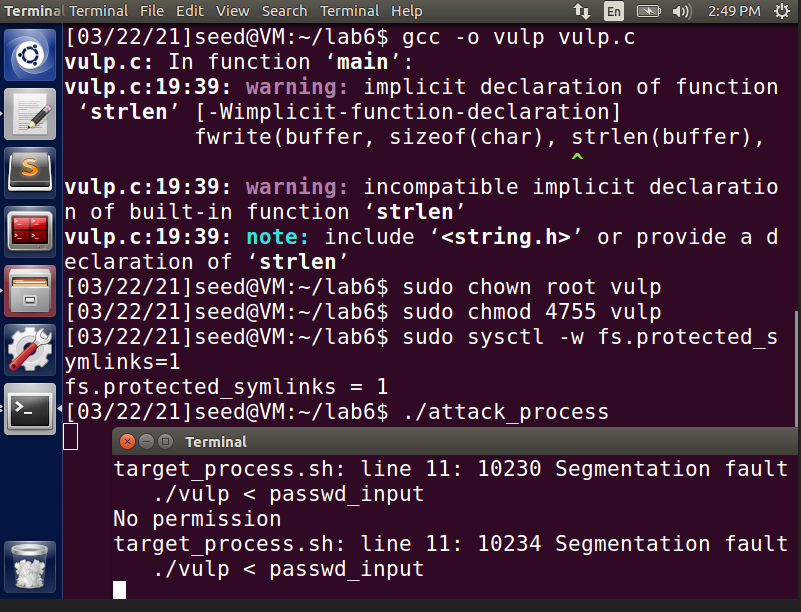


* Before running the attack process, we use the built-in scheme of Ubuntu to set the symlinks values to 1. Then the attack process is started in one terminal.



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





* 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.

1. **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.