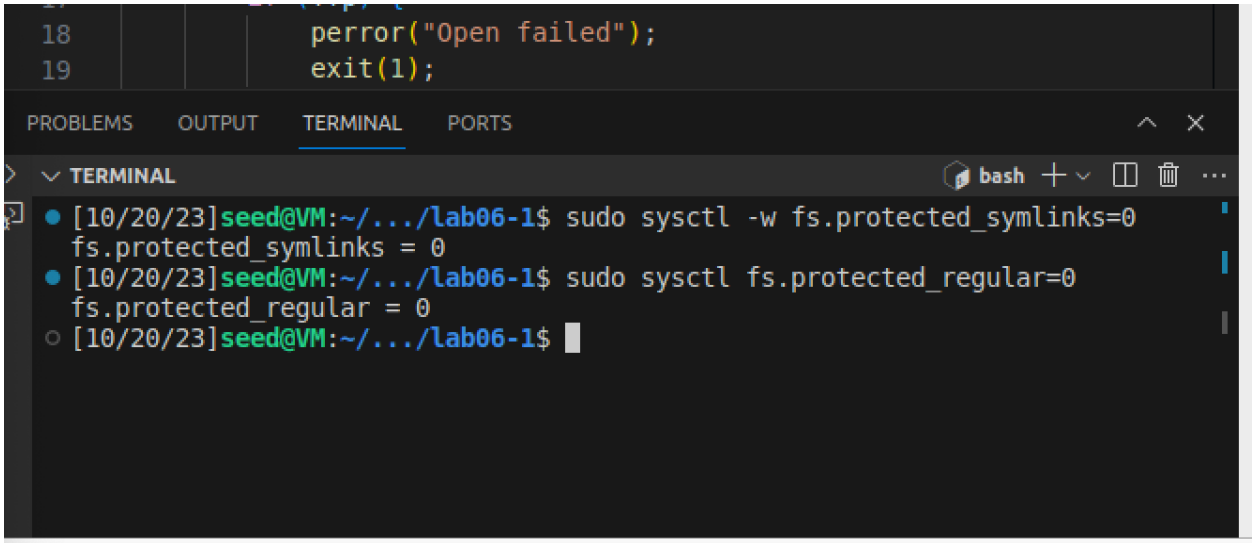
ICSI424 Lab6 Race Condition.

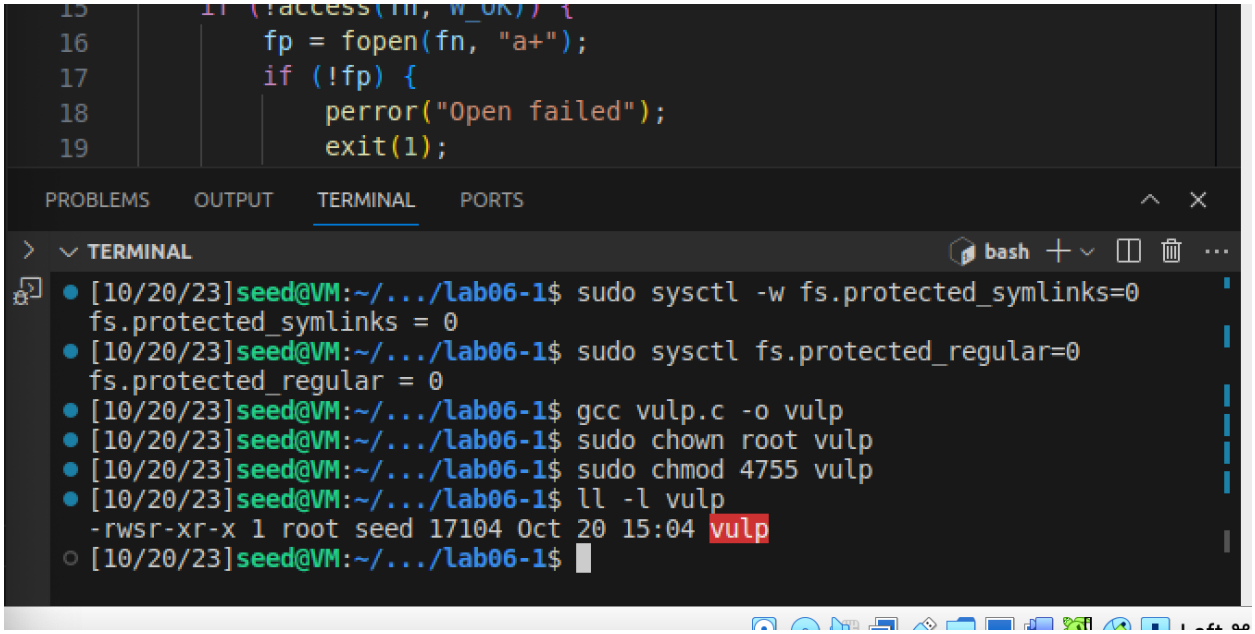
Seoyeon Choi

2.1 Turning Off Countermeasures



// Turning off Ubuntu build-in protection against race condition attacks.

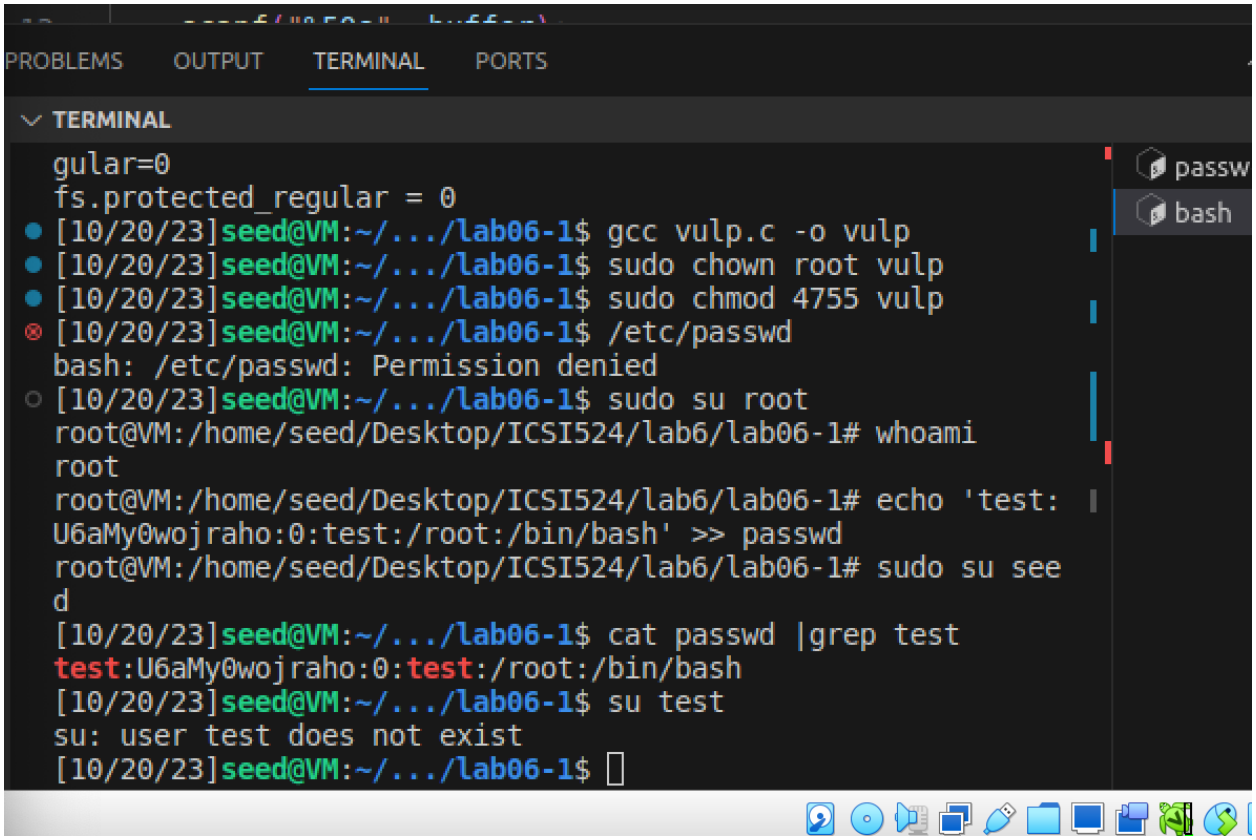
2.2 A Vulnerable Program



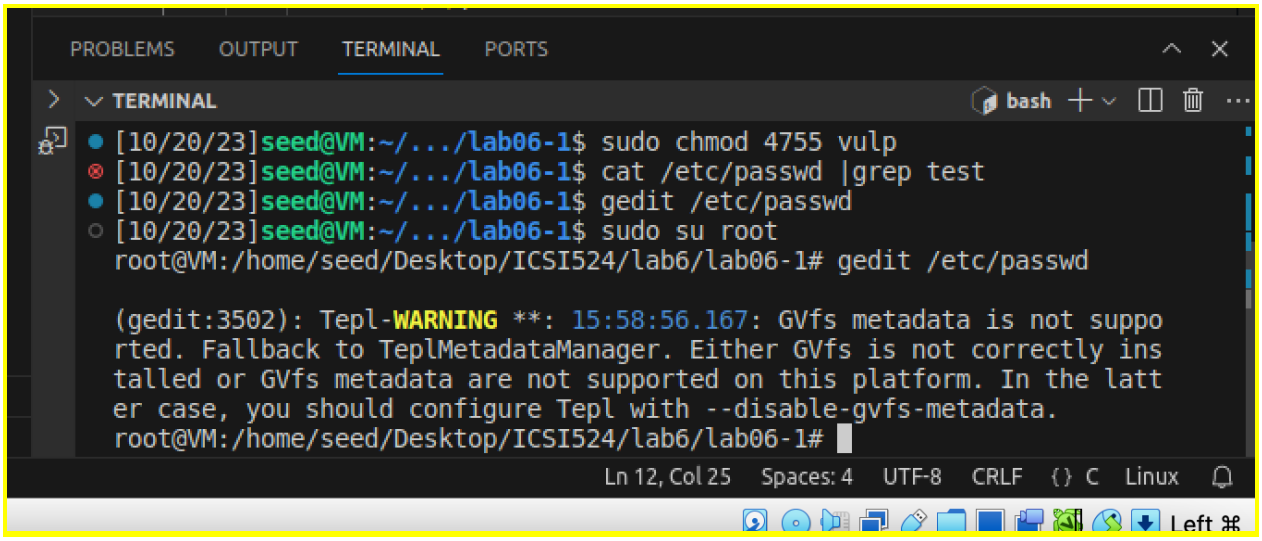
// Change owner to the root and set-uid privilege for vulp.c program.

3 Task 1: Choosing Our Target

Task. To verify whether the magic password works or not, we manually (as a superuser) add the following entry to the end of the /etc/passwd file. Please report whether you can log into the test account without typing a password, and check whether you have the root privilege.

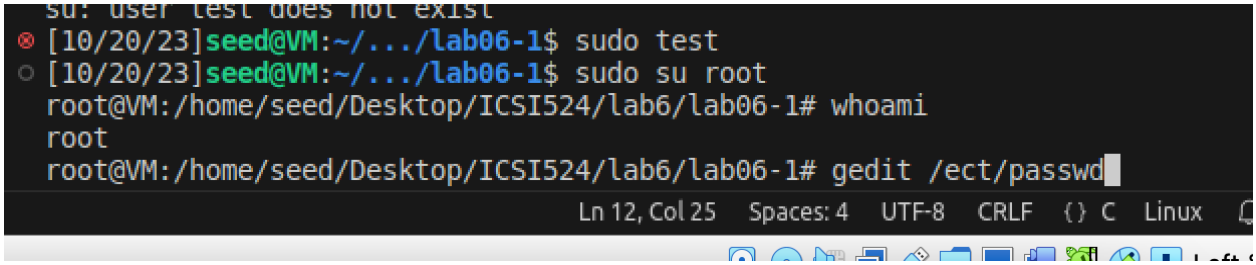


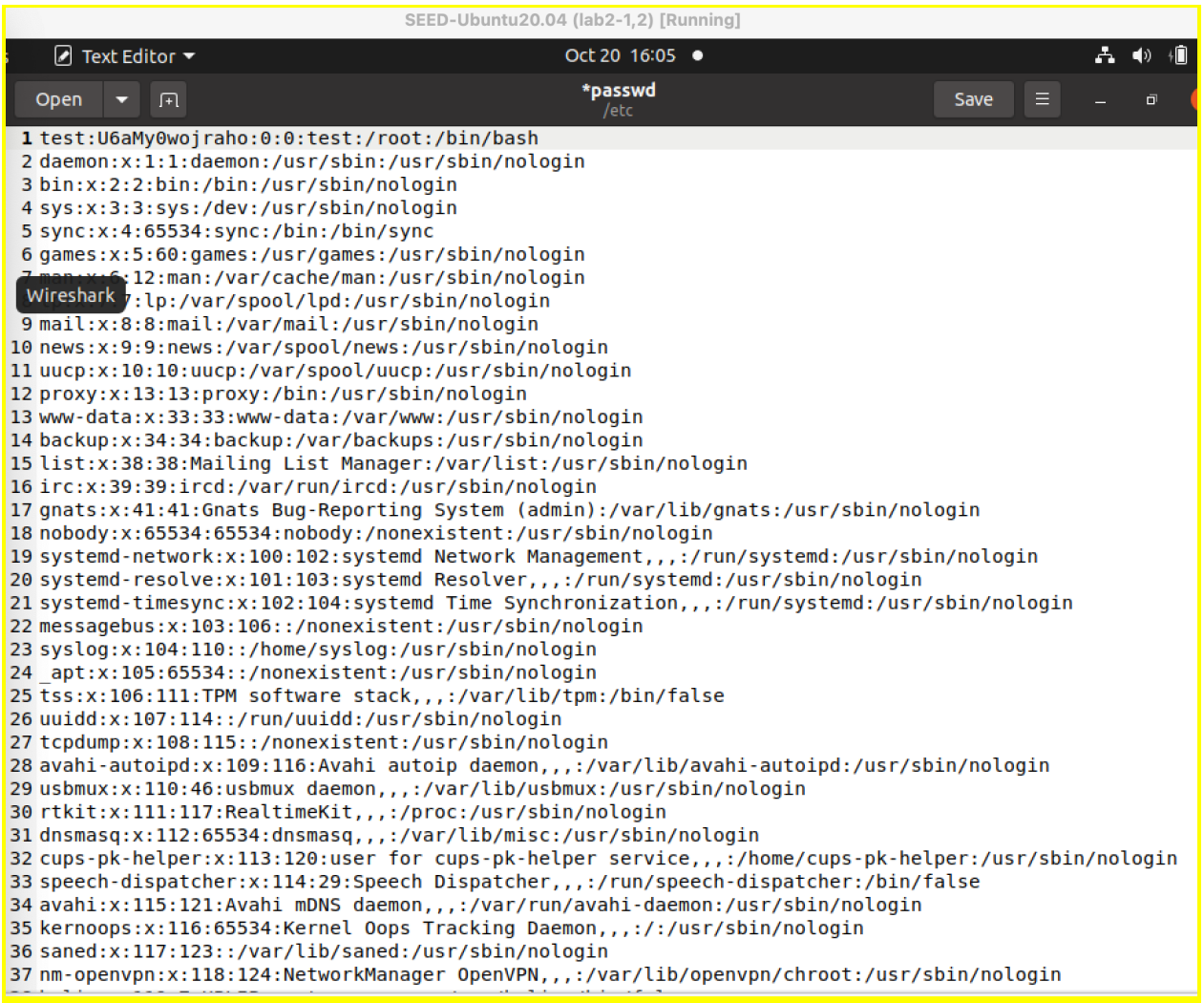
First I enter the root using sudo su root, check whether I am root by using whoami. I used the echo command line to put a special password into passwd.
And then I used “cat passwd | grep test” to print out password for test.
I can see that 2nd session is changed to the magic value for the password, but I was not able to login as test.



I have to open /etc/passwd using gedit keyword and I have to open that file with root privilege so that I could modify/change the passwd file and be able to save. I tried

without root privilege, I was able to open /etc/passwd by just using gedit and able to write, but it did not allow me to save changed file.

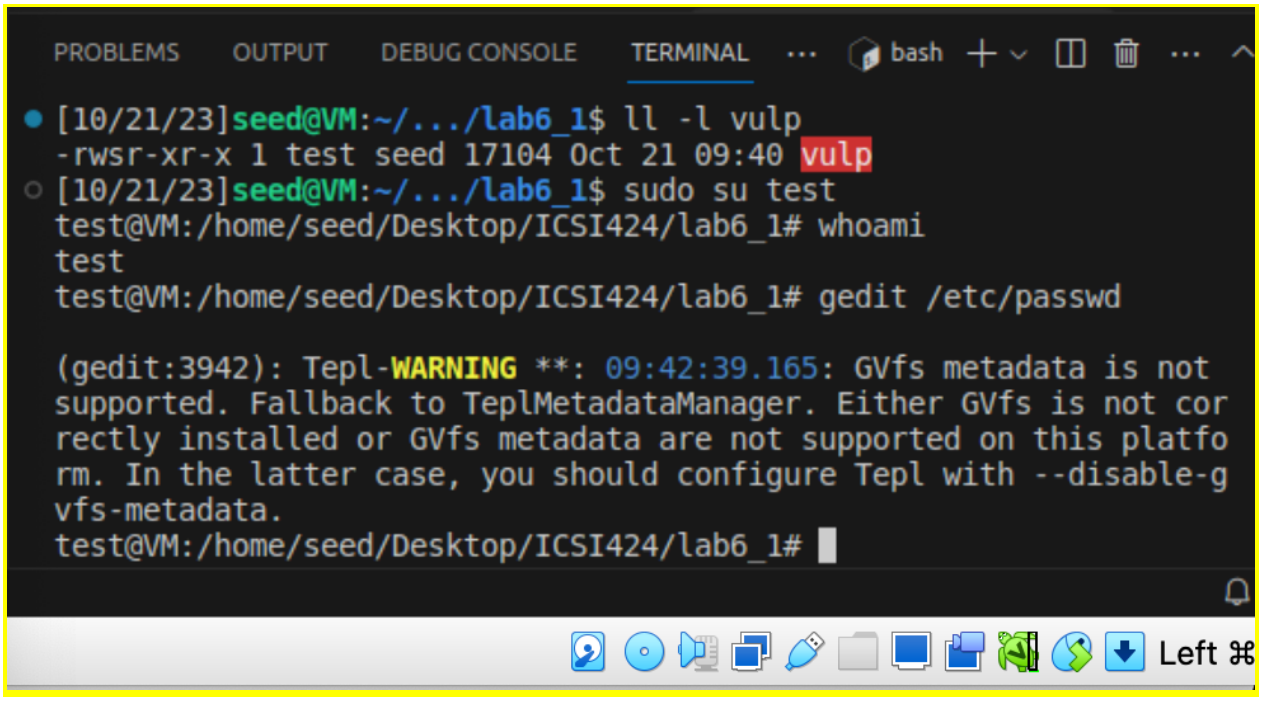




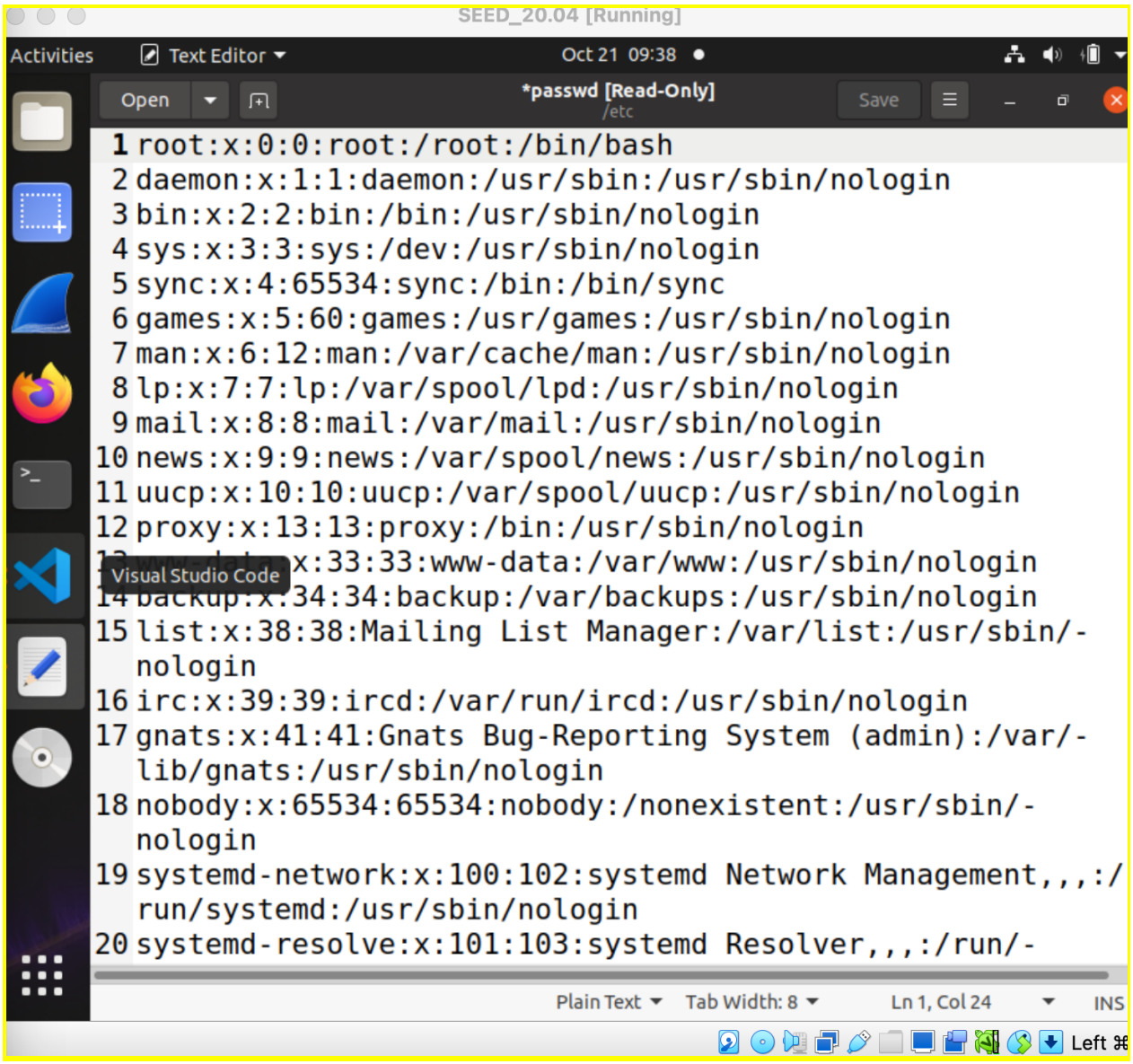
// I was able to login as a test with a magic password.
I used “gedit” to change the “/etc/passwd” file. Since I was a root user I was able to write and change the “etc/passwd” file.

With magic password, I was able to login as a test and have root privilege without typing password.

Change password and user as root.



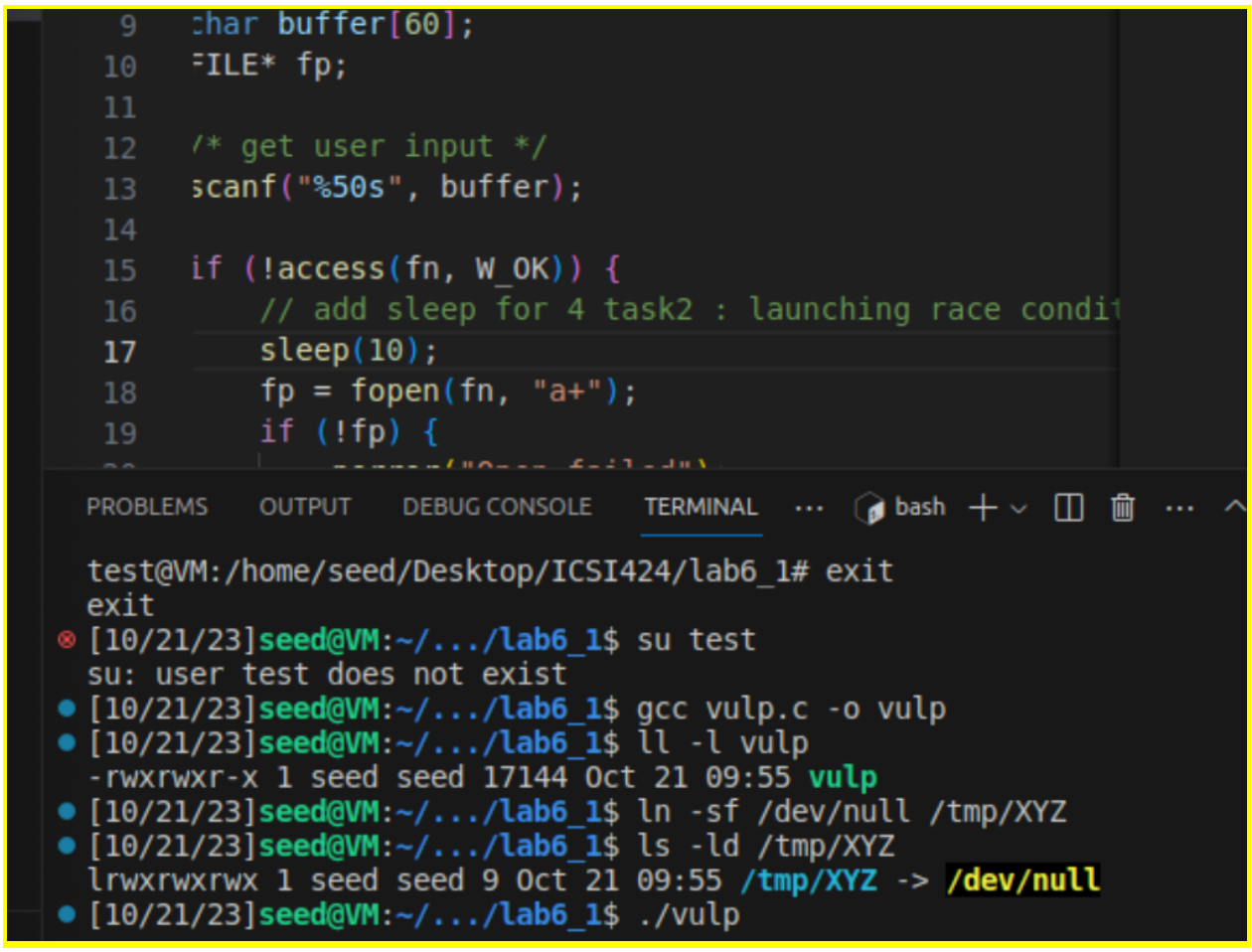
Now root is test, so I logged in as test by typing “sudo su test” and opened the /etc/passwd to change test to root and magic password back to x (second column)



// change password to the root and x for the second raw which indicates pin number. (changed test to root). Save and exit. And try to login again as a test. I shouldn’t able to login

This shows that I changed back to root therefore the test is no longer registered and unable to get into the root shell.

4.1 Task 2.A: Simulating a Slow Machine



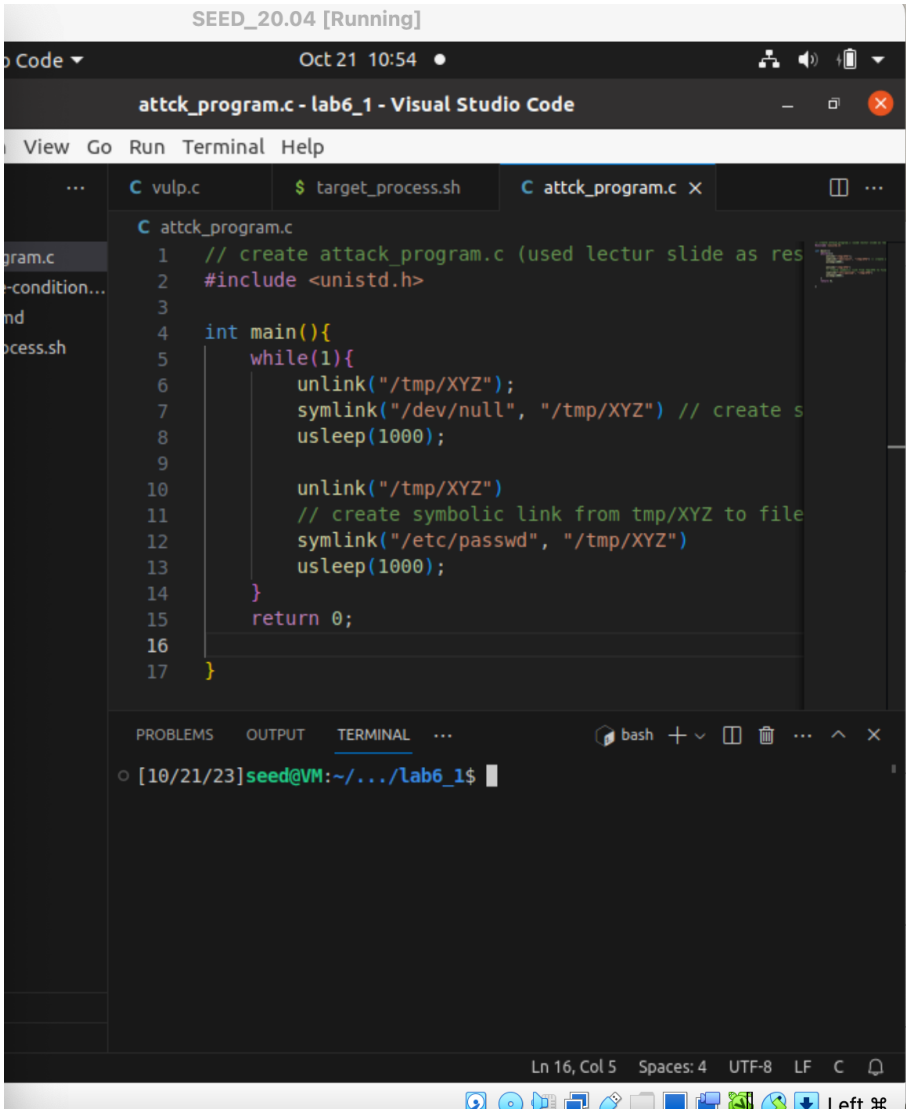
I added sleep(10) in vulp.c and compiled. I changed symlink, I pointed /tmp/XYZ to /dev/null and using -sf command line and we can see that the symbolic link is pointing to /dev/null.

Your job is to manually do something, so when the program resumes after 10 seconds, the program can help you add a root account to the system. Please demonstrate how you would achieve this.

-> The race condition vulnerability is due to the simulated time delay window between the check (access) and use (fopen). There is a possibility that the file used by access is different from the file used by fopen, even though they have the same filename /tmp/XYZ.

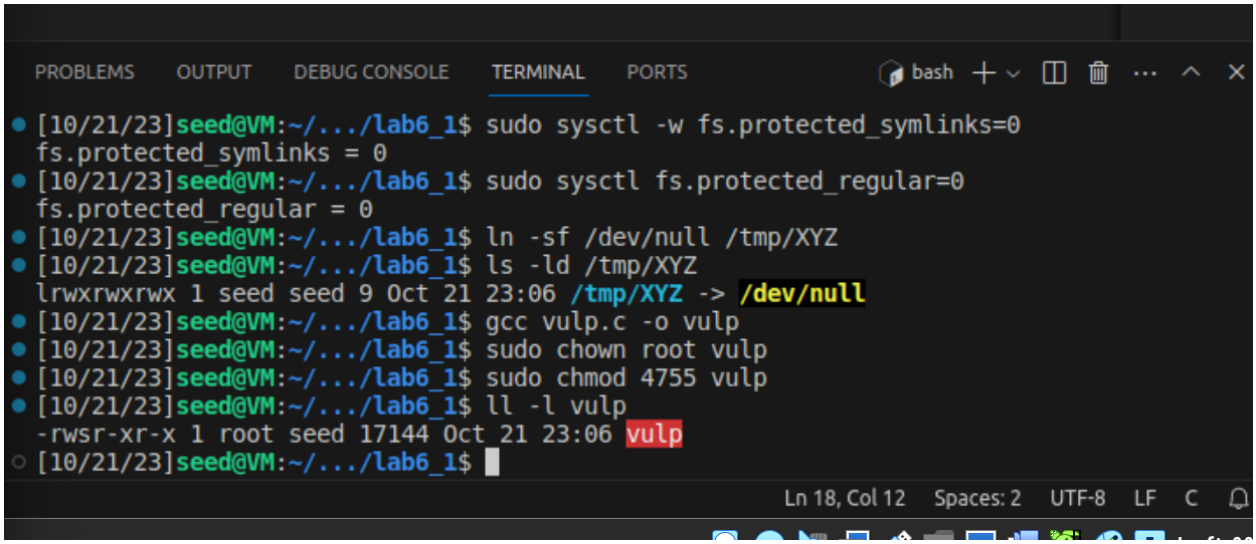
4.2 Task 2.B: The Real Attack

// created attack_program.c



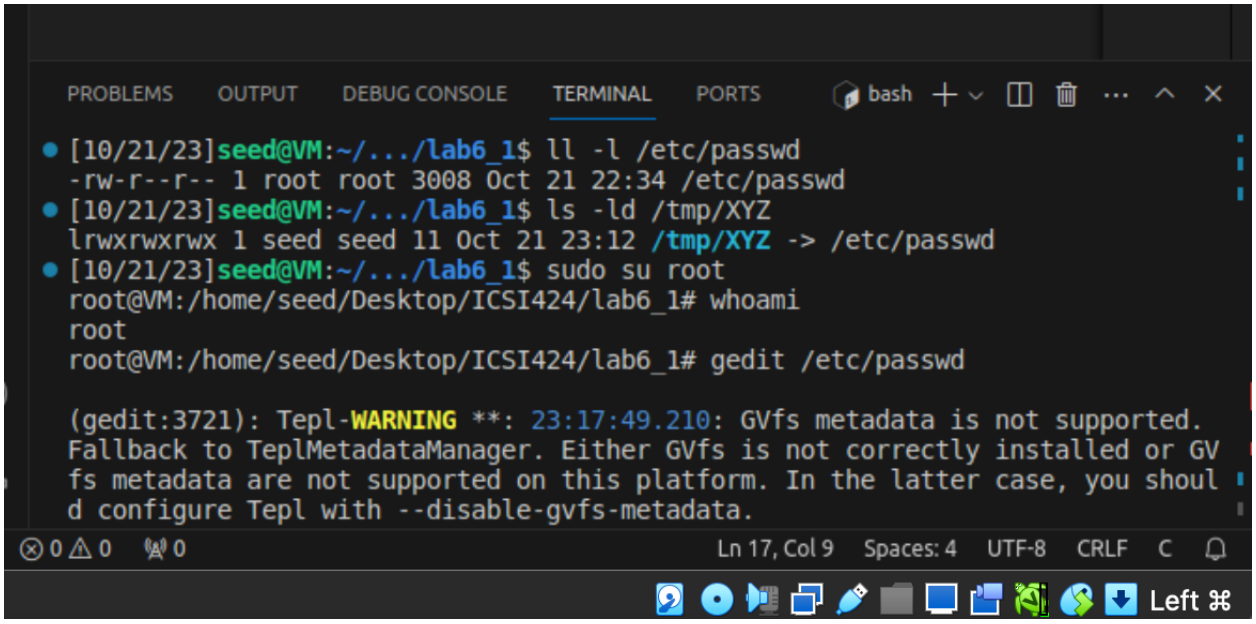
// attack_program.c , unlink “/tmp/XYZ” and then create /tmp/XYZ a symbolic link to the /dev/null file. Some file that you owned and slo anybody can write on /dev/null. And then make a process to sleep for a certain amount of time and then create /tmp/XYZ a symbolic link to the /etc/passwd where we want to attack. So that we can run target_process and attack_program in parallel to create race conditions and modify the target file which is /etc/passwd file.

// tast 4 attack A

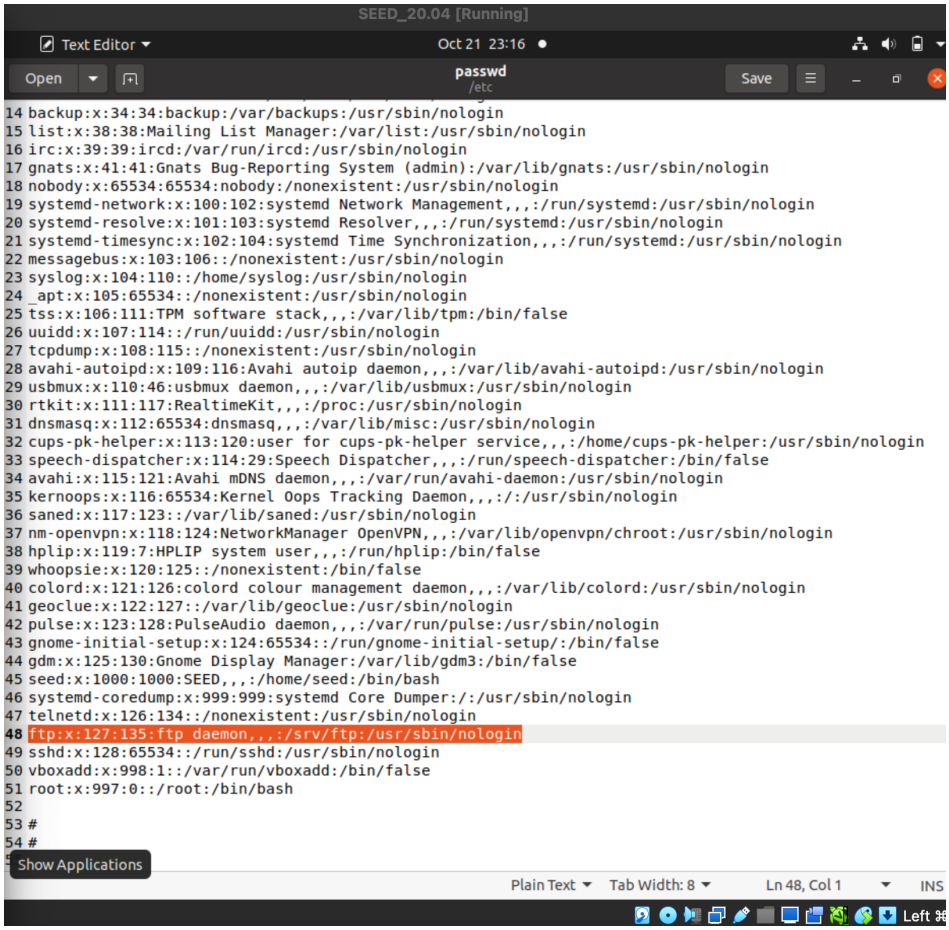


// create symbolic link of /tmp/XYZ to points to /dev/null
Check the /tmp/XYZ symbolic link by typing “ls -ld /tmp/XYZ”
We an see that /tmp/XYZ is pointing to /dev/null

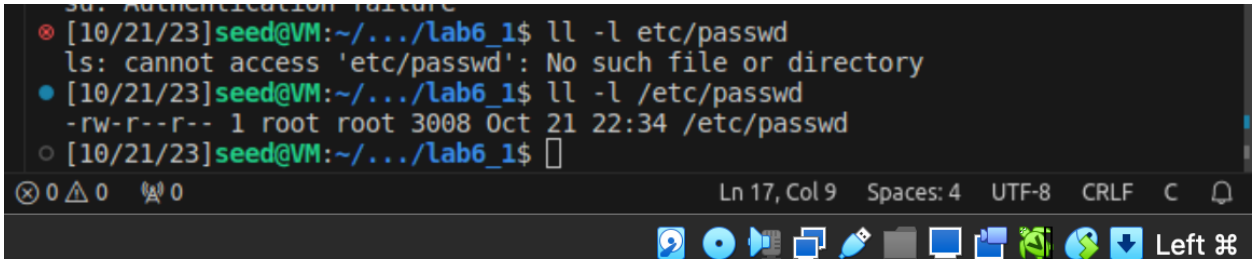
Compile vulp.c (make sure you add sleep(10) between access() and fopen() so we can launch an attack between 10 sleep periods) , change owner to root and set-UID. Check the permissions. By typing ll -l vulp.



// first I checked /etc/passwd file’s last modified time. It was October 21th 22:23. I can see that /tmp/XYZ is pointing to /etc/passwd after compiling vulp.c
I typed gedit /etc/passwd to see whether attacks were successful. If race condition attack was succeeded then there will be newly added line in /etc/passwd file.



// after pointing /tmp/XYZ to /etc/passwd. I opened /etc/passwd and I can see that some lines are added to the passwd file.

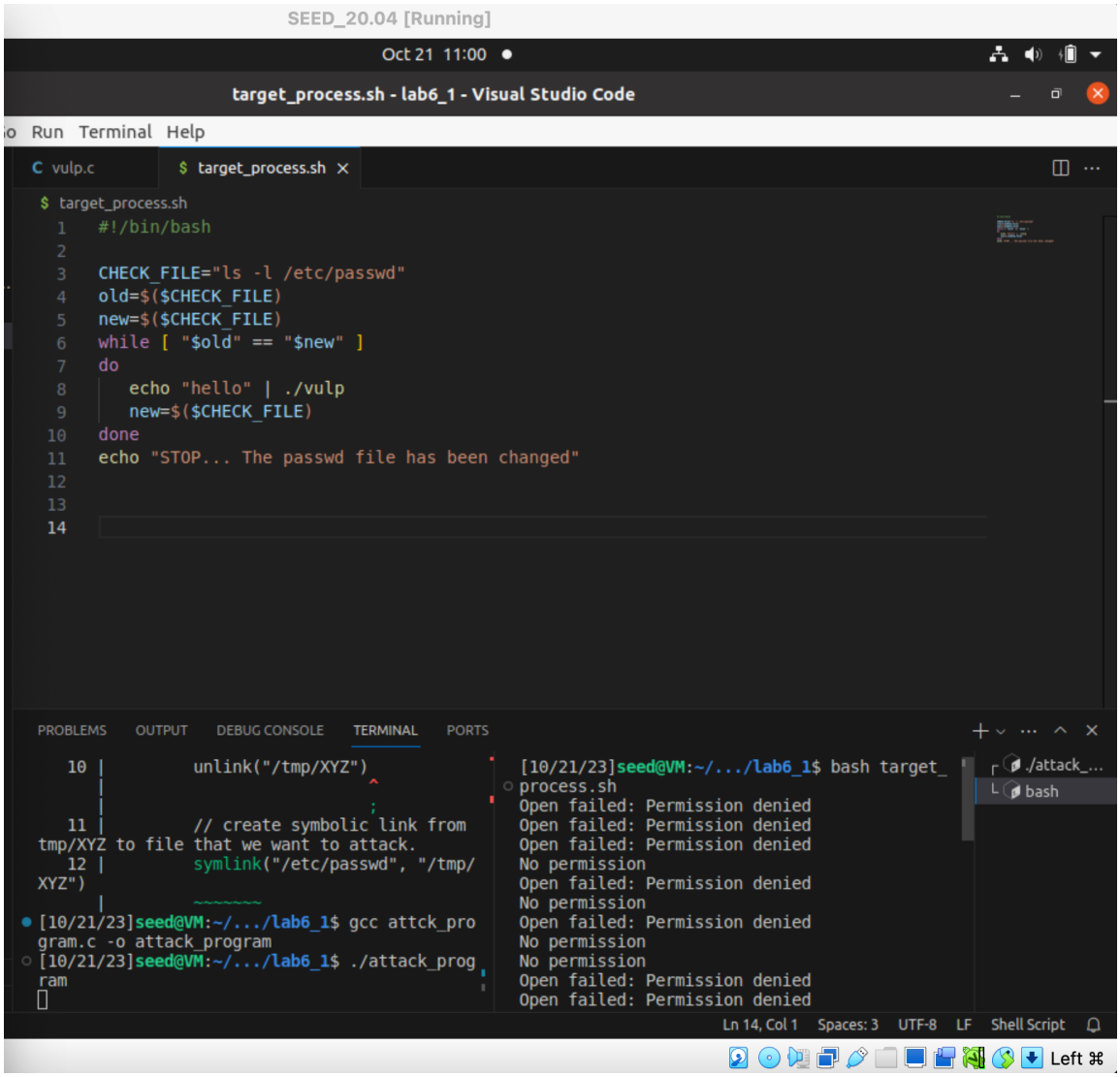


// after finishing the attack, I checked the last modified time for /etc/passwd file. It was 22:34 which means the /etc/passwd file was modified.

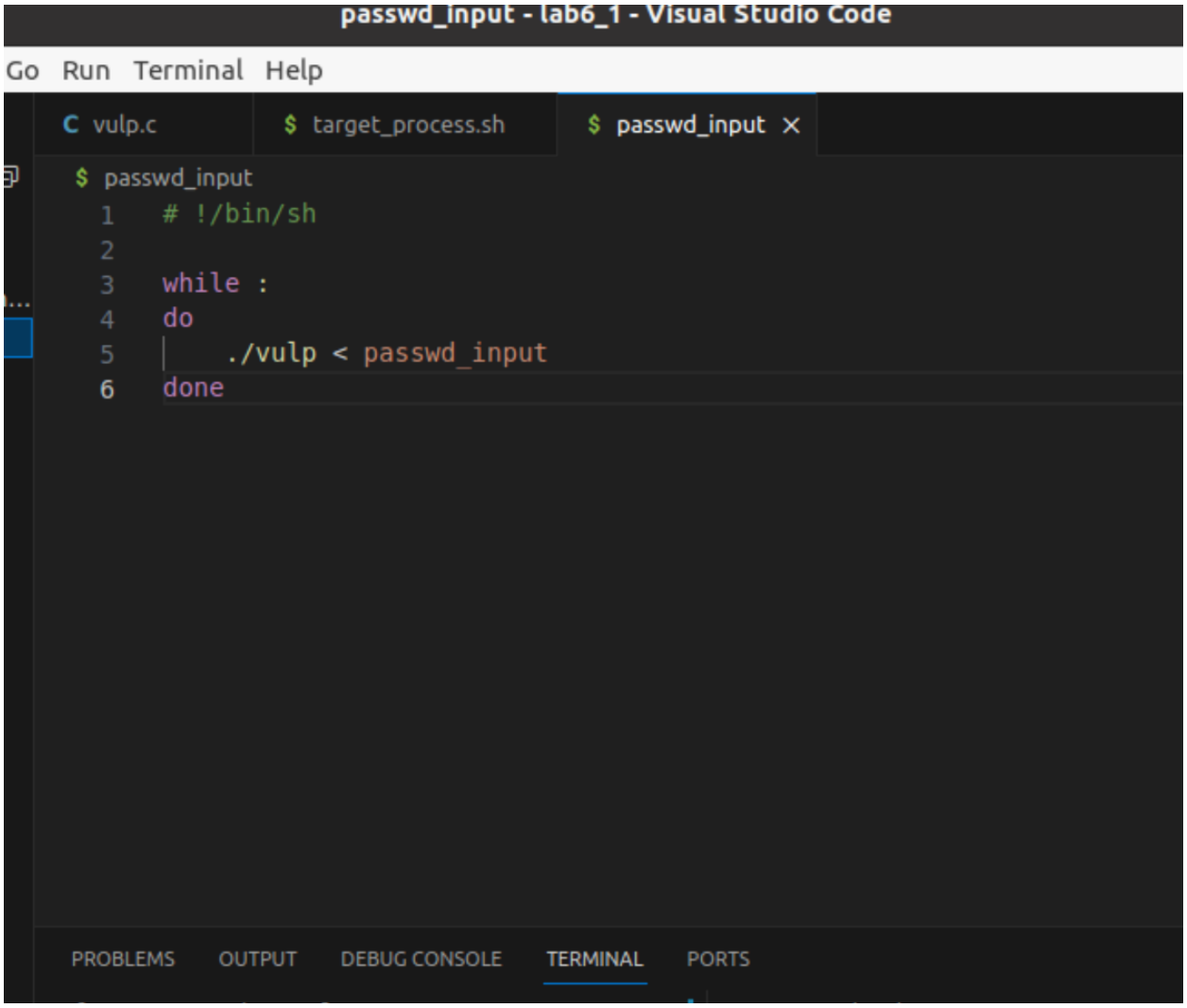
Overall 4.1 Task A Simulating a slow machine

I linked /tmp/XYZ symbolic link to point to /dev/null and checketd /tmp/XYZ symbolic link pointer. I compiled the vulp.c file and changed owner root and set-did. I compiled ./vulp and within 10 seconds (between access() and open()) I changed /tmp/XYZ symbolic link to points to /etc/passwd file. and checked whether /etc/passwd files were modified. It was modified since there were new lines in /etc/passwd and modified time were updated (we can check this by typing ll -l /etc/passwd)

4.2 Task 2.B: The Real Attack

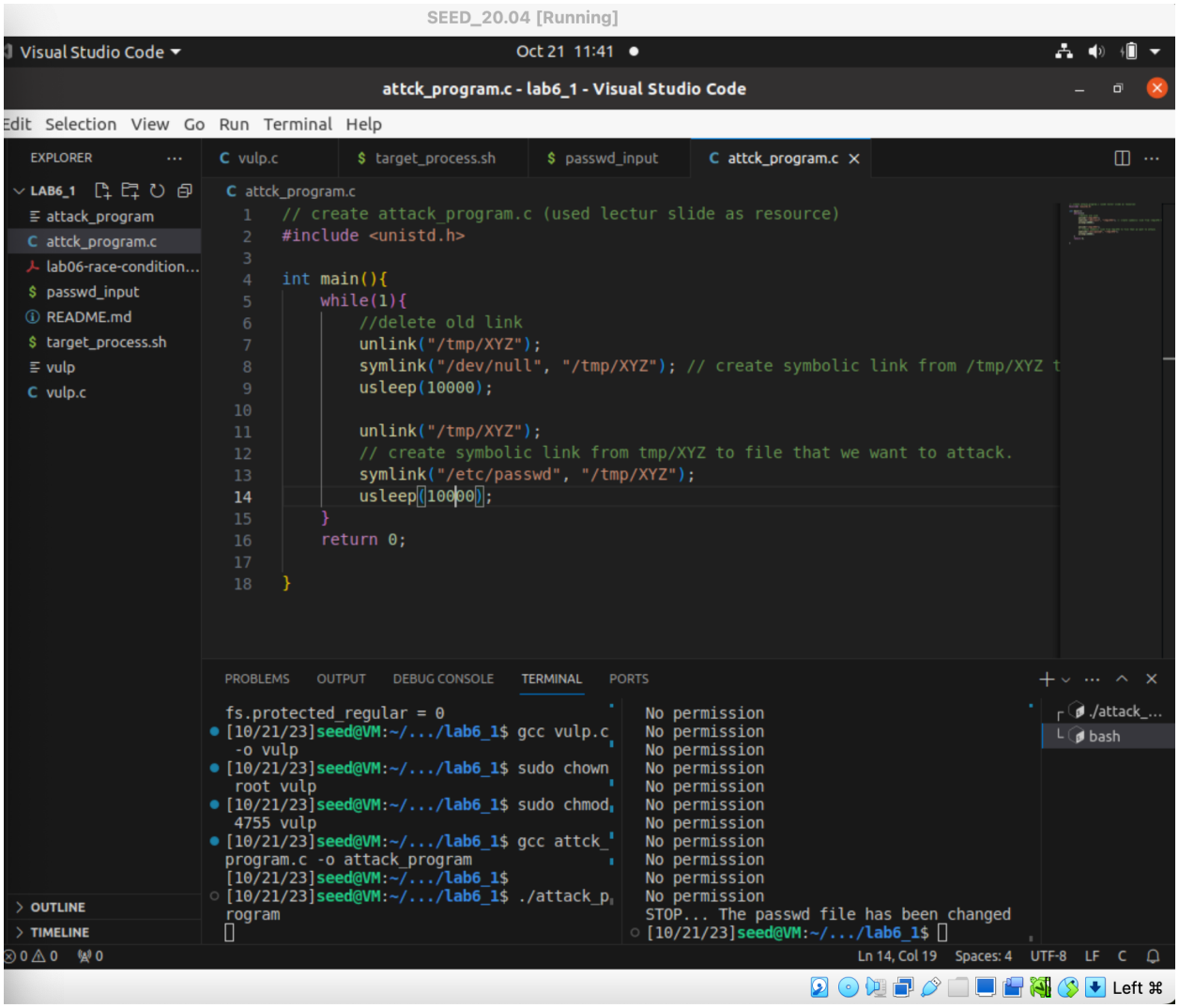


I compiled attack_program and ran the targetprocess.sh in parallel to see whether I could attack.



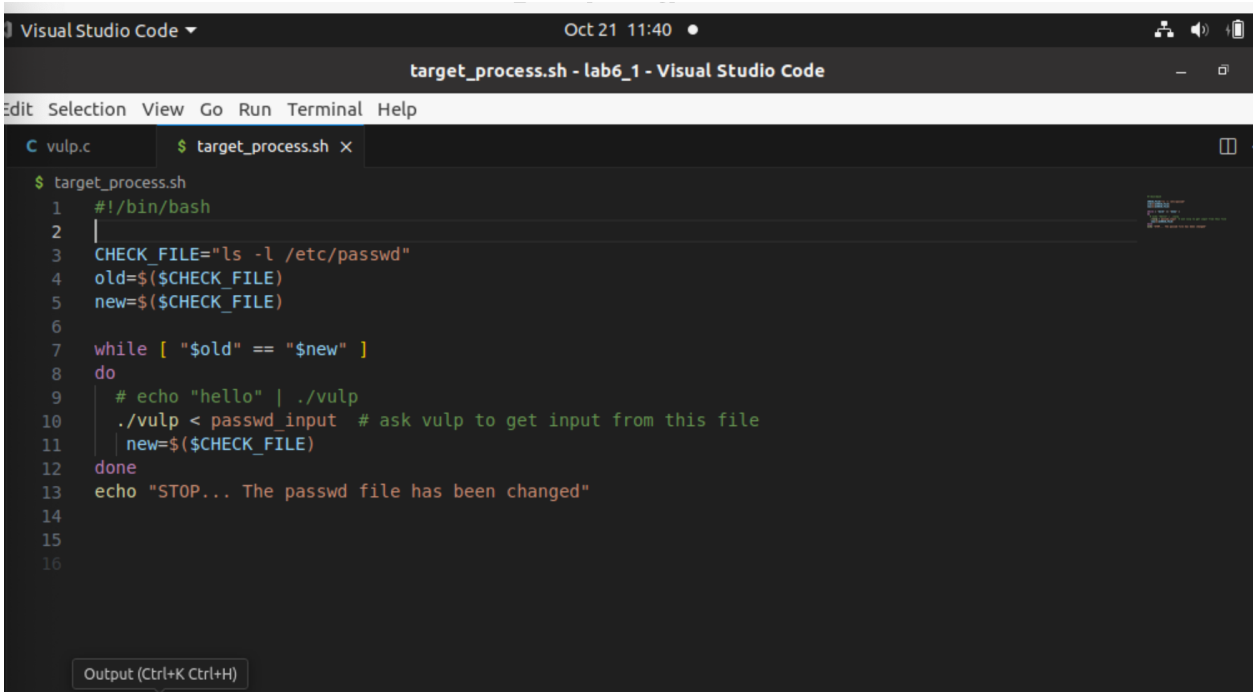
// add passwd_input screenshot

I wasn't able to succeed at launching an attack and changing the/ etc/passwd file so I created "passwd_input" which makes a vulnerable program (./vulp) run infinitely in while loop and contains strings into the passwd_input.

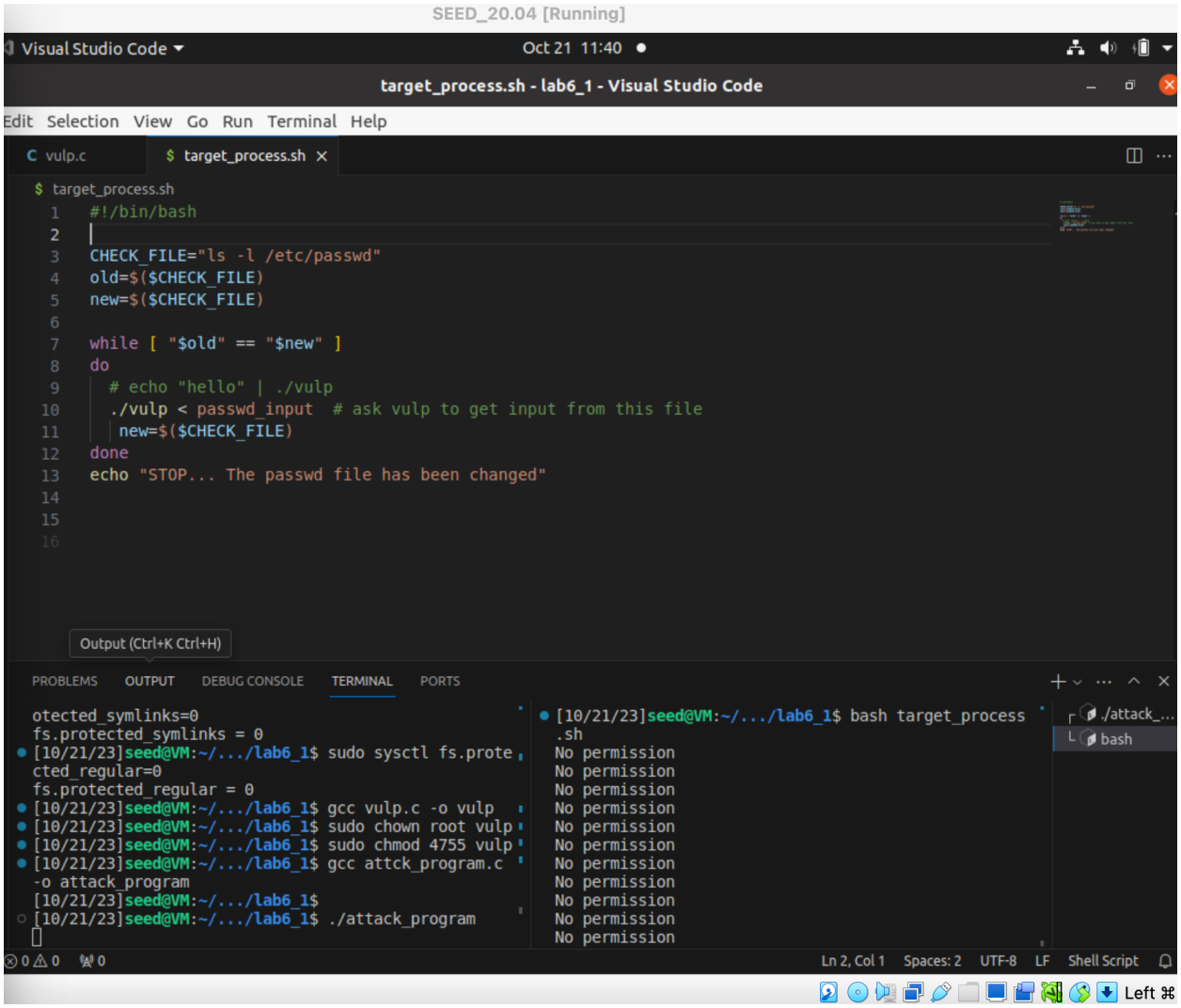


// add attck_program.c

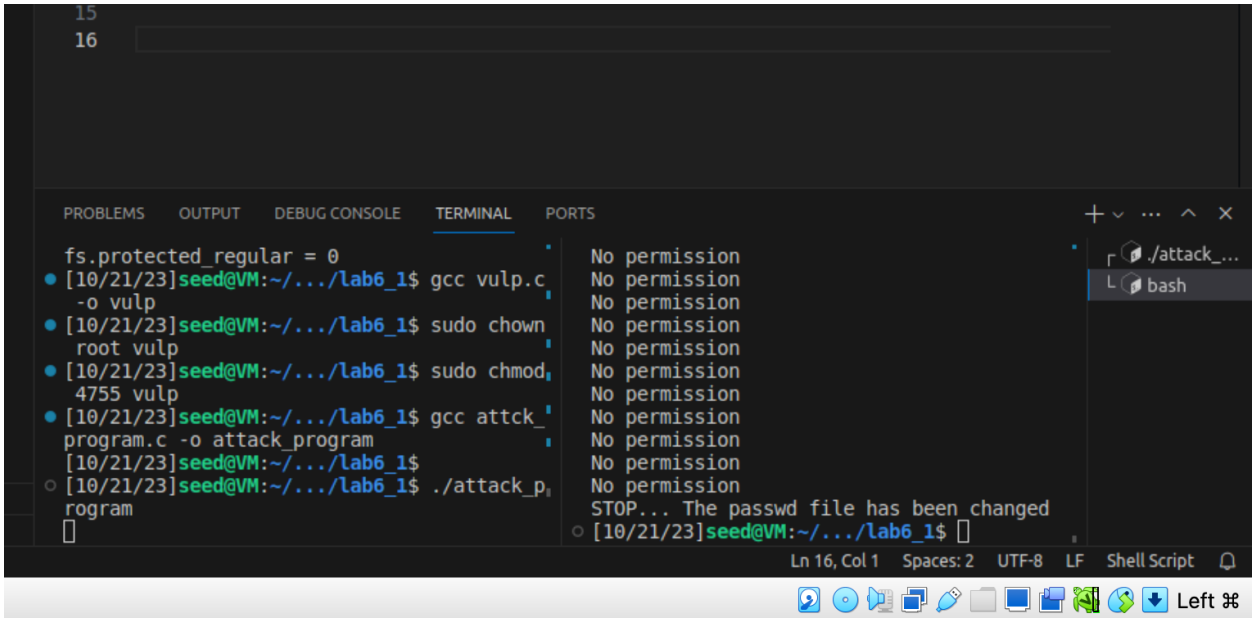
// I also created attack_program.c file that launches attack by unlink(/tmp/XYZ) and create symbolic link to the /dev/null which is file that anybody can write and then I unlinked(/tmp/XYZ) and create symbolic link to the file that I want to attack/ modify which was /etc/passwd. Also I added sleep ex) usleep(10000) which makes the process to sleep for certain amount of time.



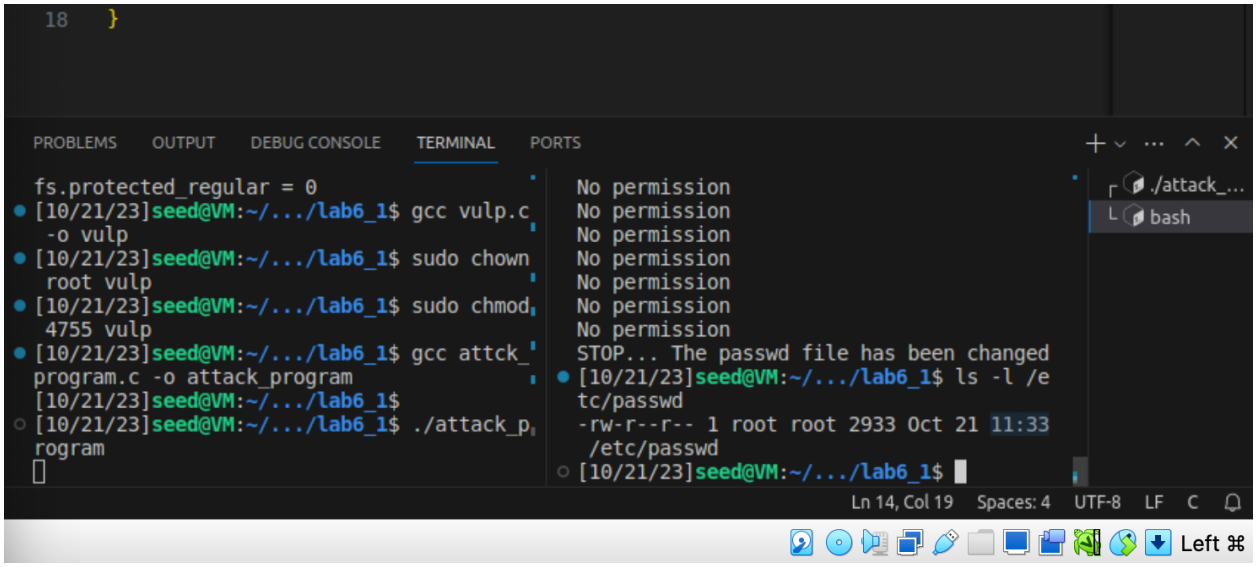
// add screenshot of target_process.sh
// Lastly I added one line in the target_process.sh. Which was " ./vulp < passwd_input
So this line can run the vulnerable program.



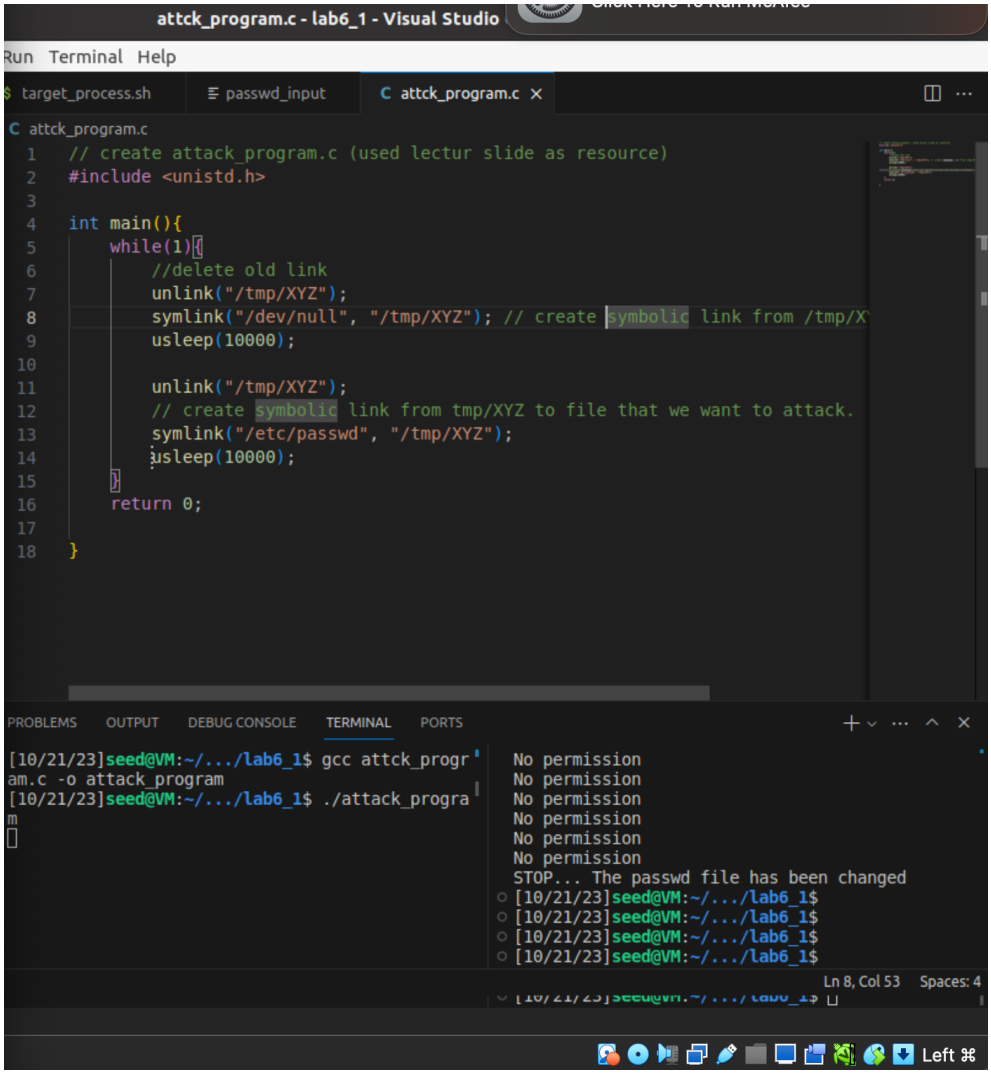
// Waiting for target_process side terminal to print out “STOP … The passwd file has been changed. “



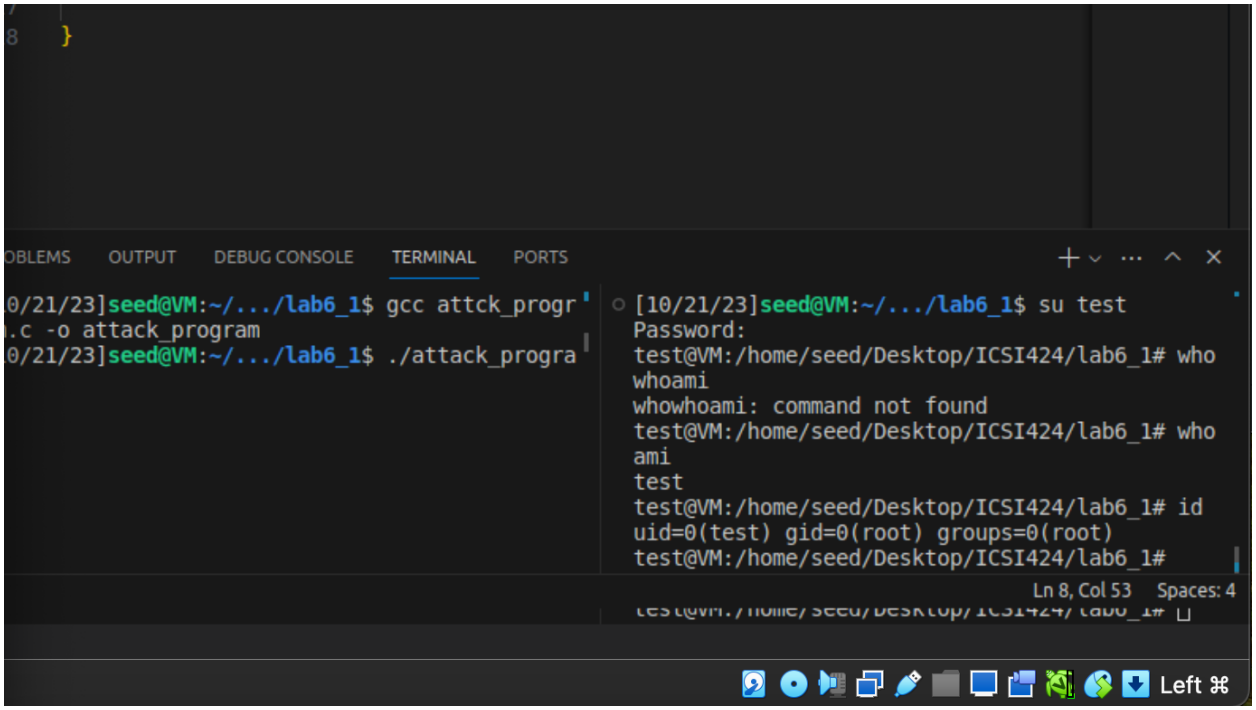
And then I compiled attck_program.c file and created separate two terminals so that I could run ./attack_program and bash target_process.sh in parallel.
Once, /etc/passwd modified, the terminal that I runed bash taraget_process.sh will stop and print out that passwd file has been changed. We can see that /etc/passwd file has been changed.



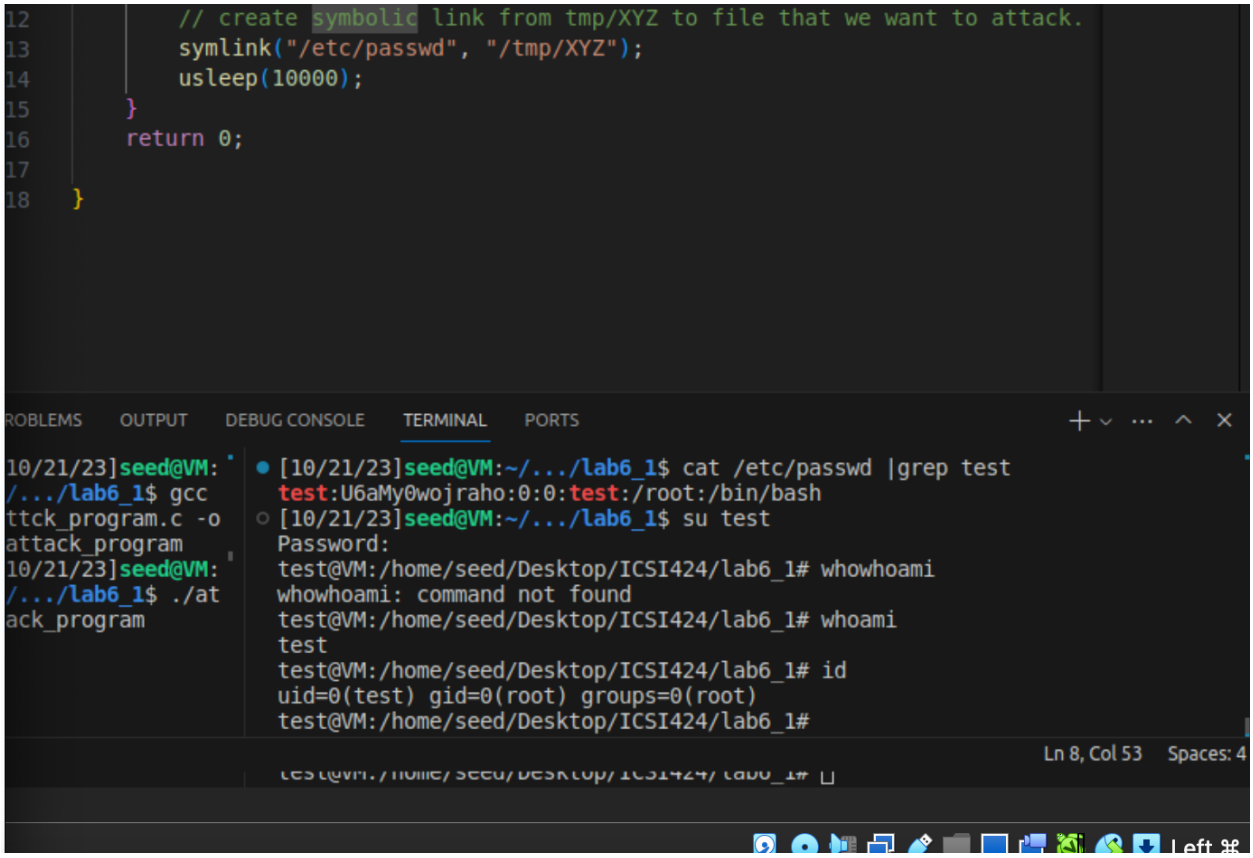
// I usd ls -l /etc/passwd to see whether it was actually modified. It shows that 11:33 Am /etc/passwd was modified.



// additional attack output screenshot.



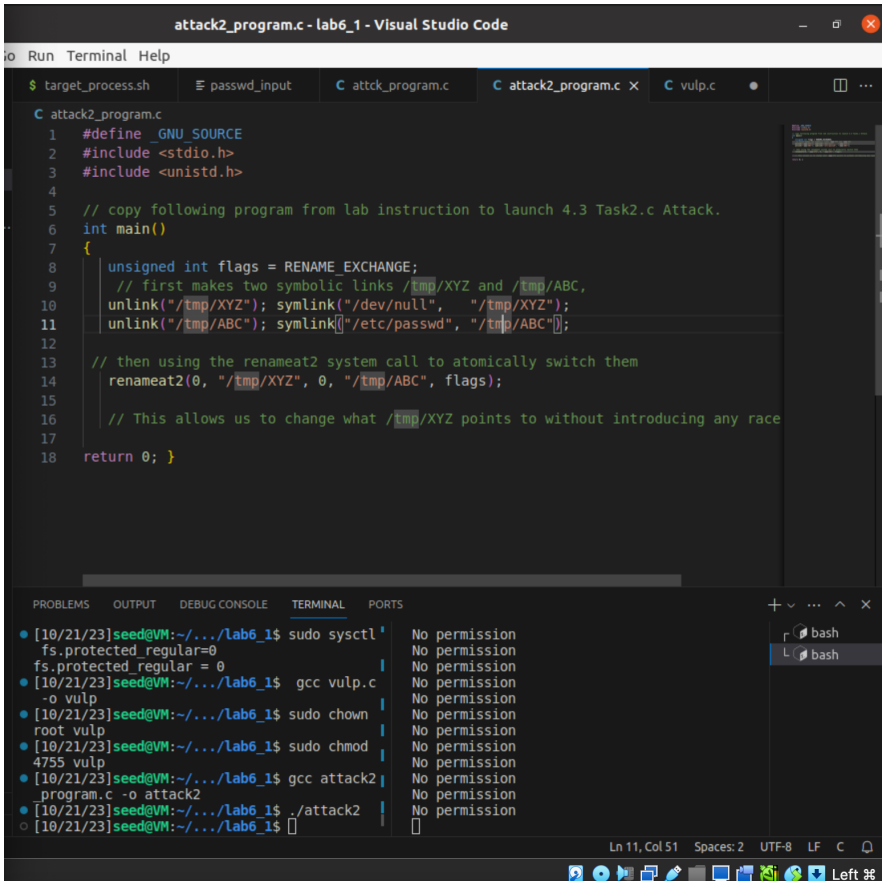
// This shows that after launching an attack, test user able to get into root.

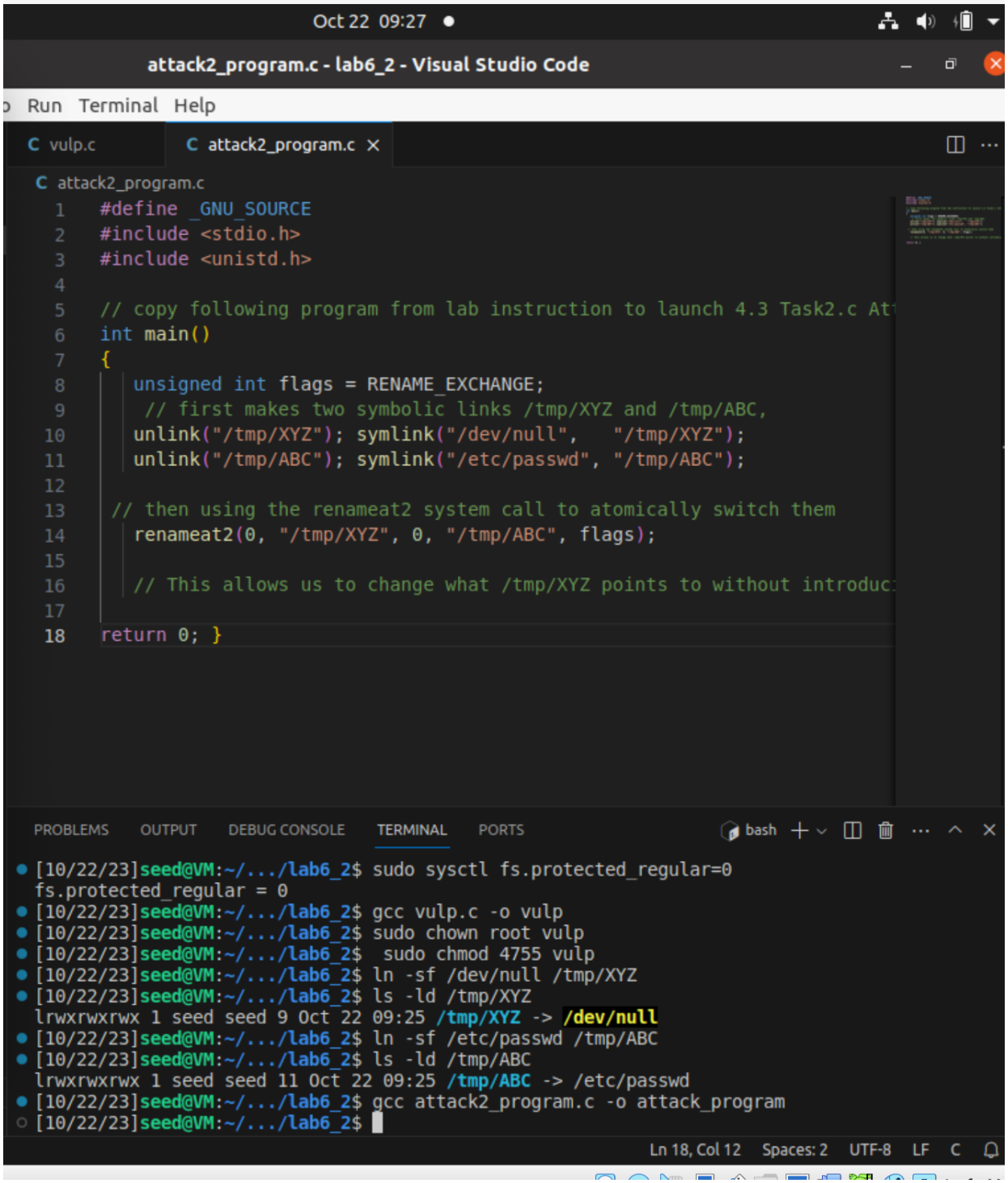


4.3 Task 2.C: An Improved Attack Method

I created attack2_program.c. This program unset symbolic links to /tmp/XYZ and /tmp/ABC. Andthen it sets symbolic link form /tmp/XYZ to /dev/null. And then I set a symbolic link to point form /tmp/ABC to /etc/passwd. Since system calls do not automatically switch those symbolic links, we use renameat2() to switch those. By using renameat2() we can prevent our program from being race conditioned because of unlinke() and symlink().

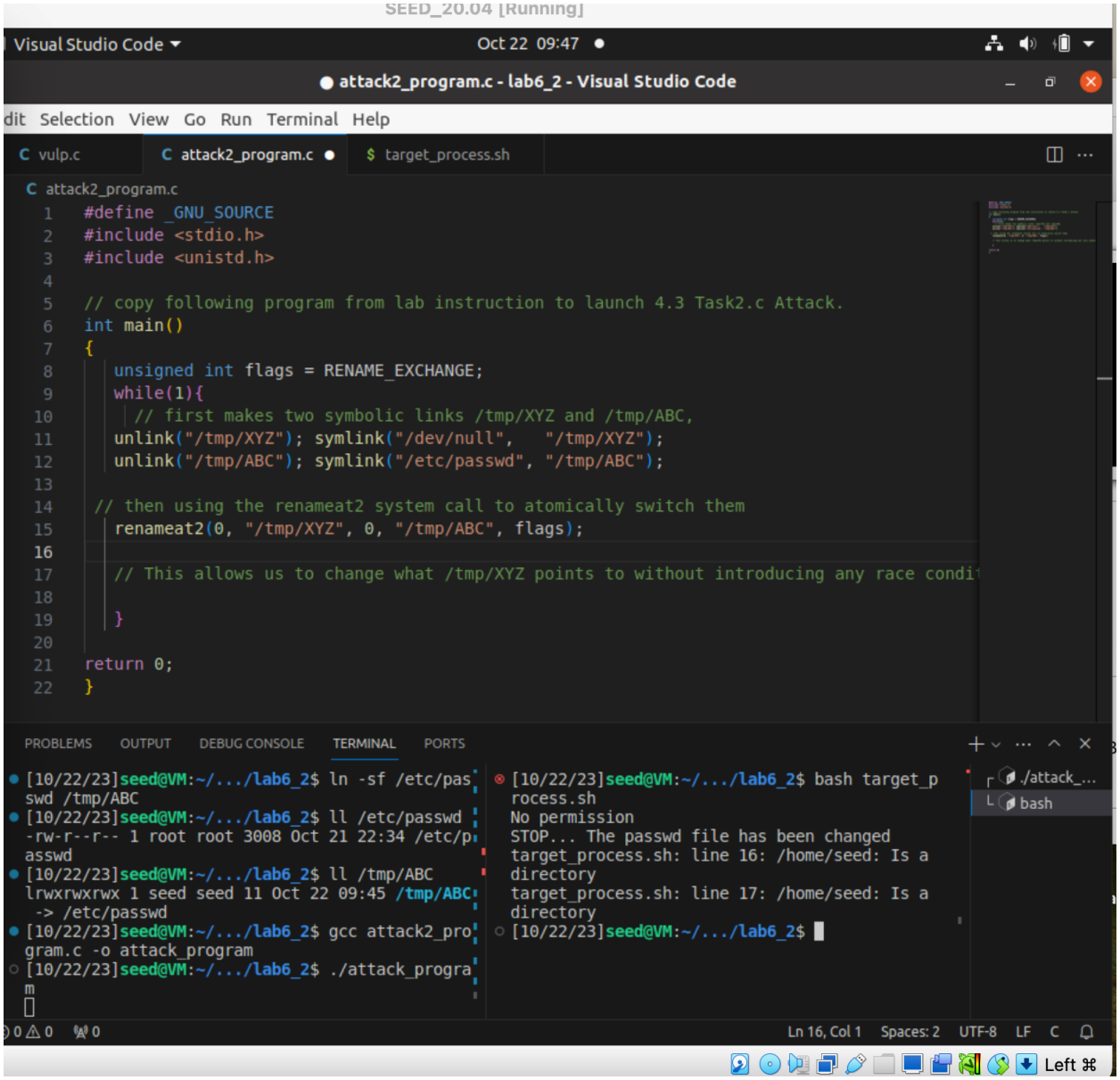
I compiled attack2_program and ran./attack2 with bash target_process.sh. Compared to previous attack, I was not able to get the message that files were changed.



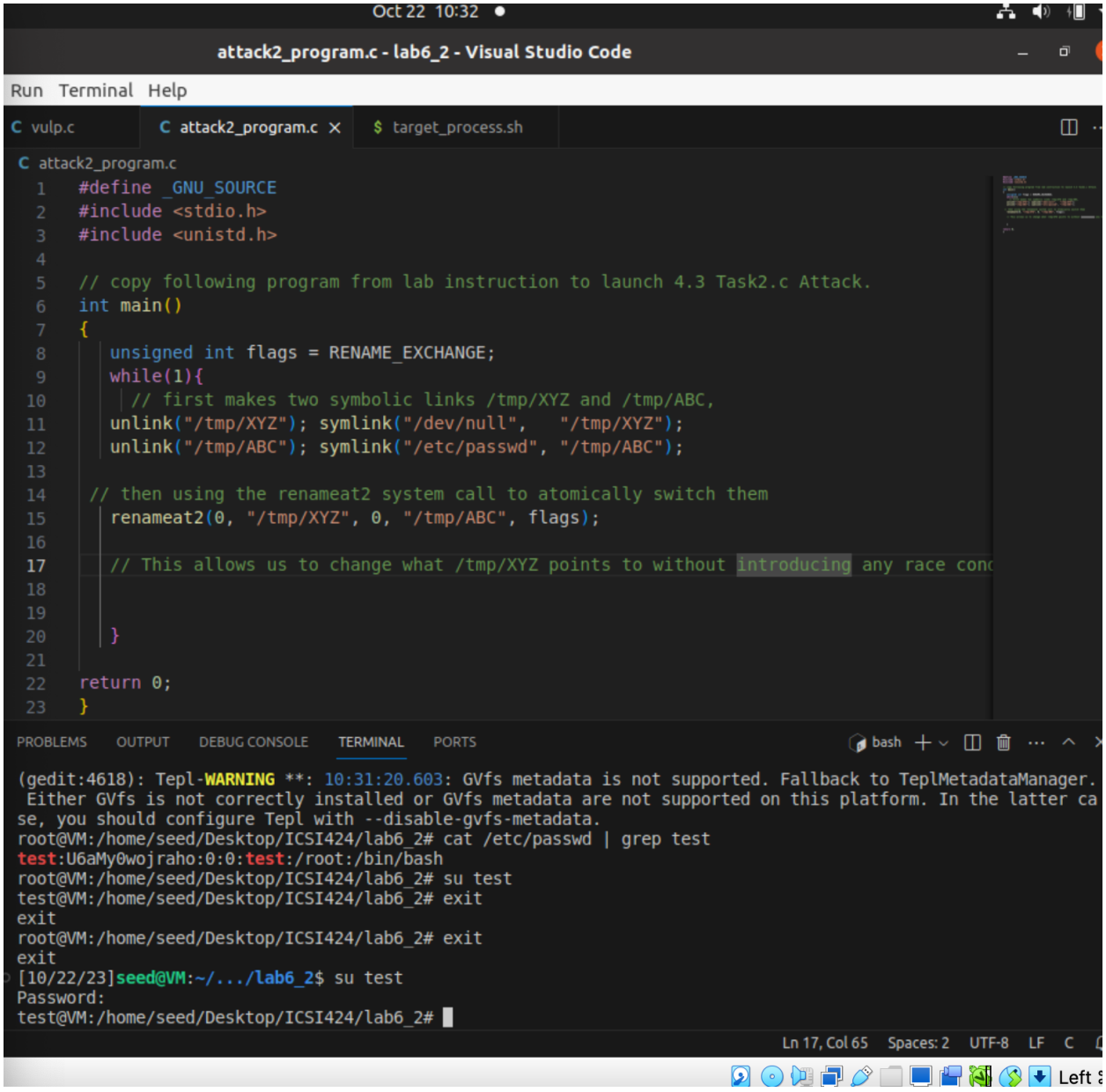


// compile vulp.c set owner to root and setuid privilege
Create symbolic link of /tmp/XYZ to point /dev/null (which every one or anybody can write down) and check symbolic link of /tmp/XYZ file by typing “ ln -ld /tmp/XYZ”
Create symbolic link of /tmp/ABC to points to /etc/passwd and check /tmp/ABC files symbolic link and wec can see that /tmp/ABC is pointing to /etc/passwd with symbolic

link. Compile attack2_program which has renameat that automatically switches above two symbolic links. (ex) /tmp/XYZ and /tmp/ABC)



// used attack2_program.c and created symbolic links and launched parallel attack.



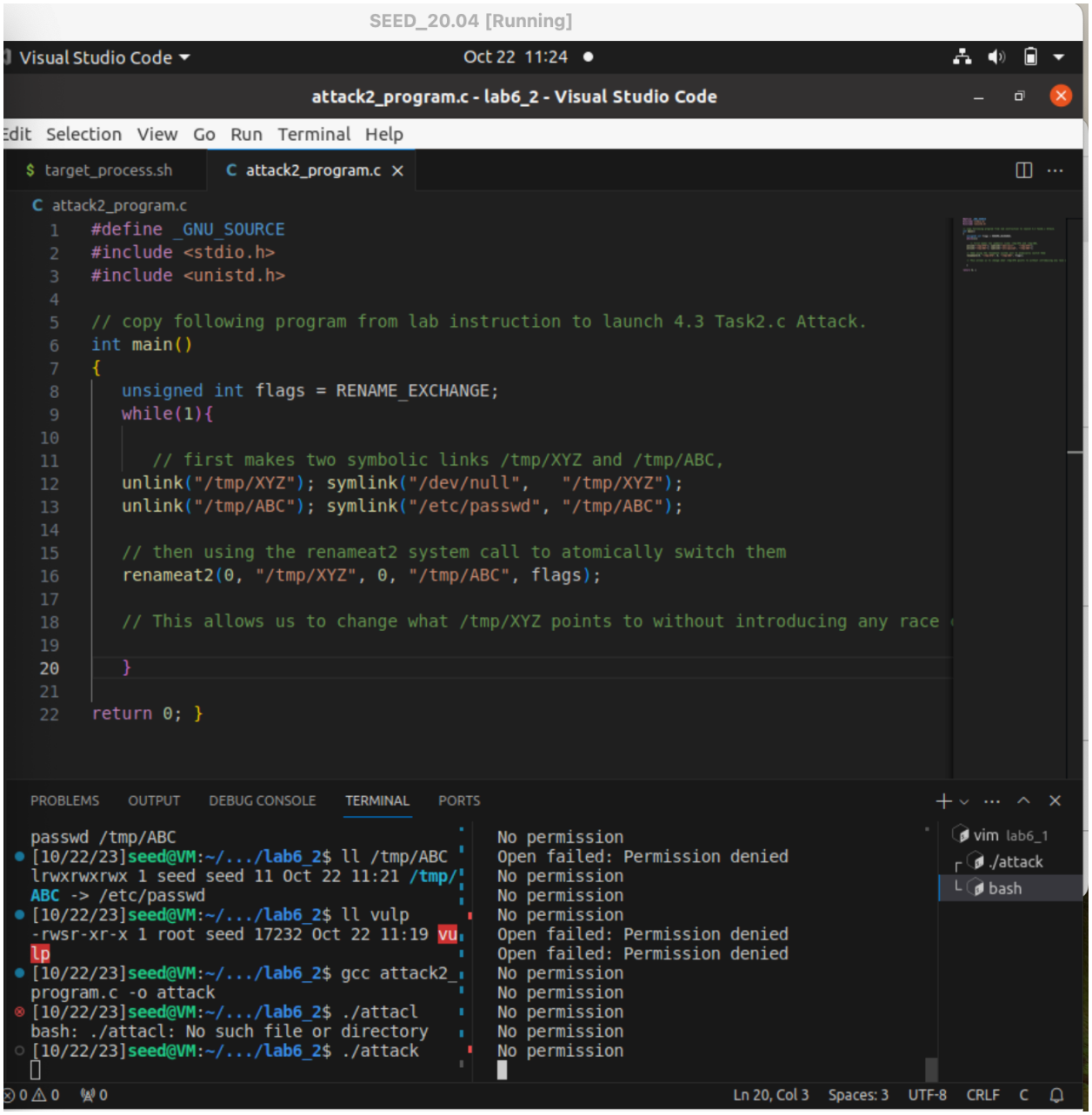
5 Task 3: Countermeasures

5.1 Task 3.A: Applying the Principle of Least Privilege

We can use set-uid system call to temporarily disable the root privilege, and later enable it if necessary. Please use this approach to fix the vulnerability in the program, and then

repeat your attack. Will you be able to succeed? Please report your observations and provide an explanation.

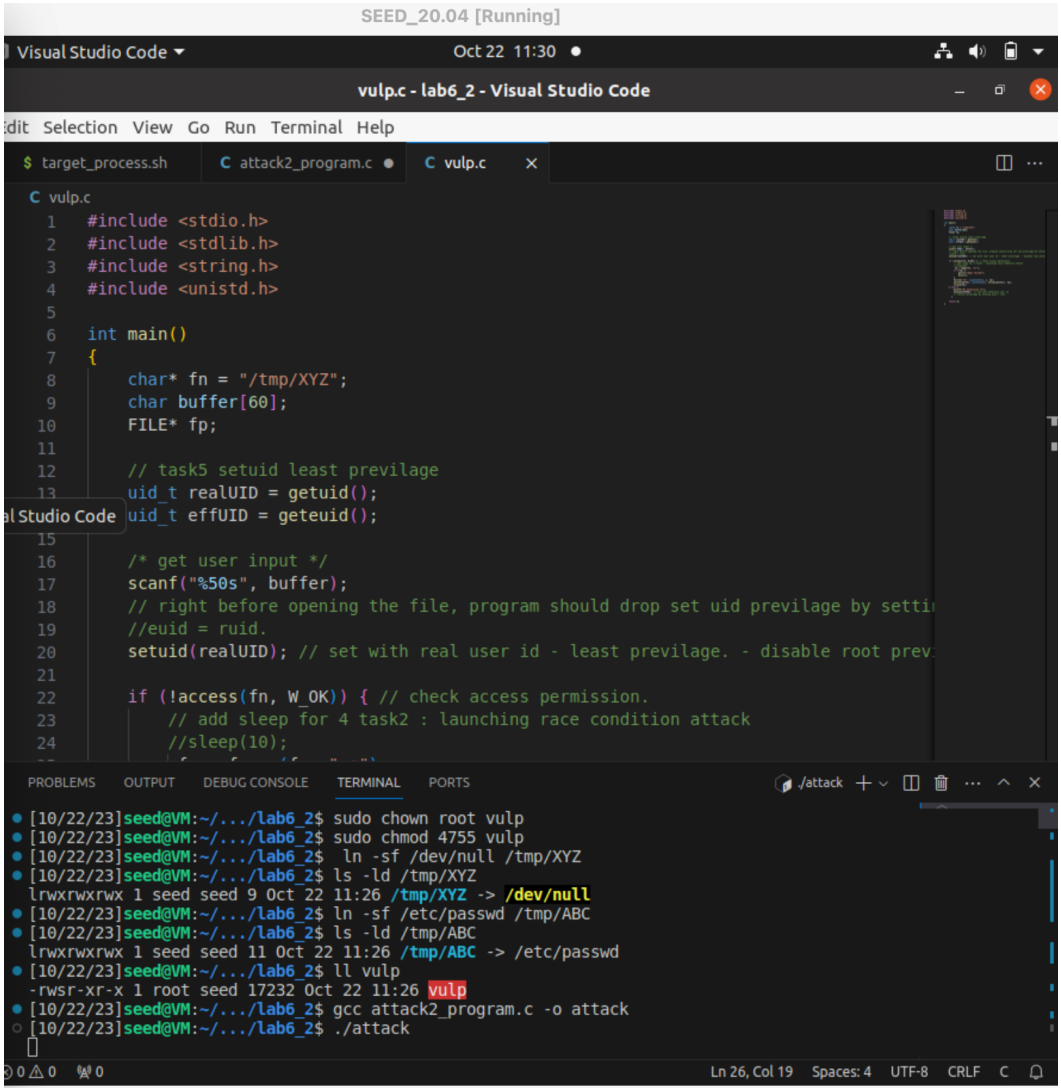
To enforce the principle of least privilege, remove permissions if the user does not need them. And after using the permission, if the user doesn't need to have the permission, then just delete it. This allows the user to grant minimal permissions whenever necessary. Therefore, I will set effective user id to be the same as real user id (EUID = RUID) before access() and then once we are done with our task, I will make SETUID root owned program.



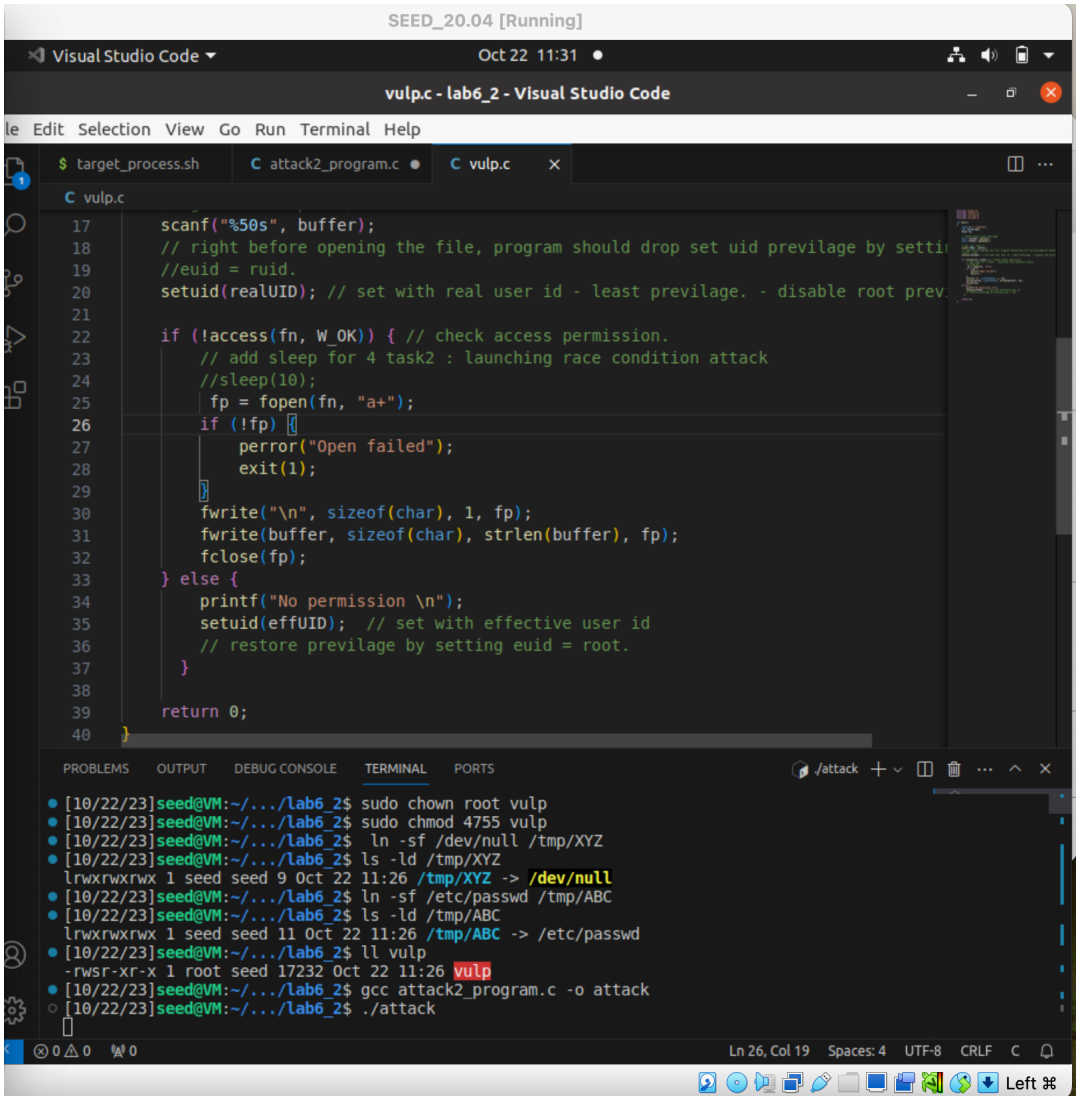
// Output shows that we won't be able to attack and modify /etc/passwd because of the least privilege principle.

// with dropping privilege before entering access() and fopen() we won't be able to attack. I was only able to get open and failed with no permission.

// code for vulp.c

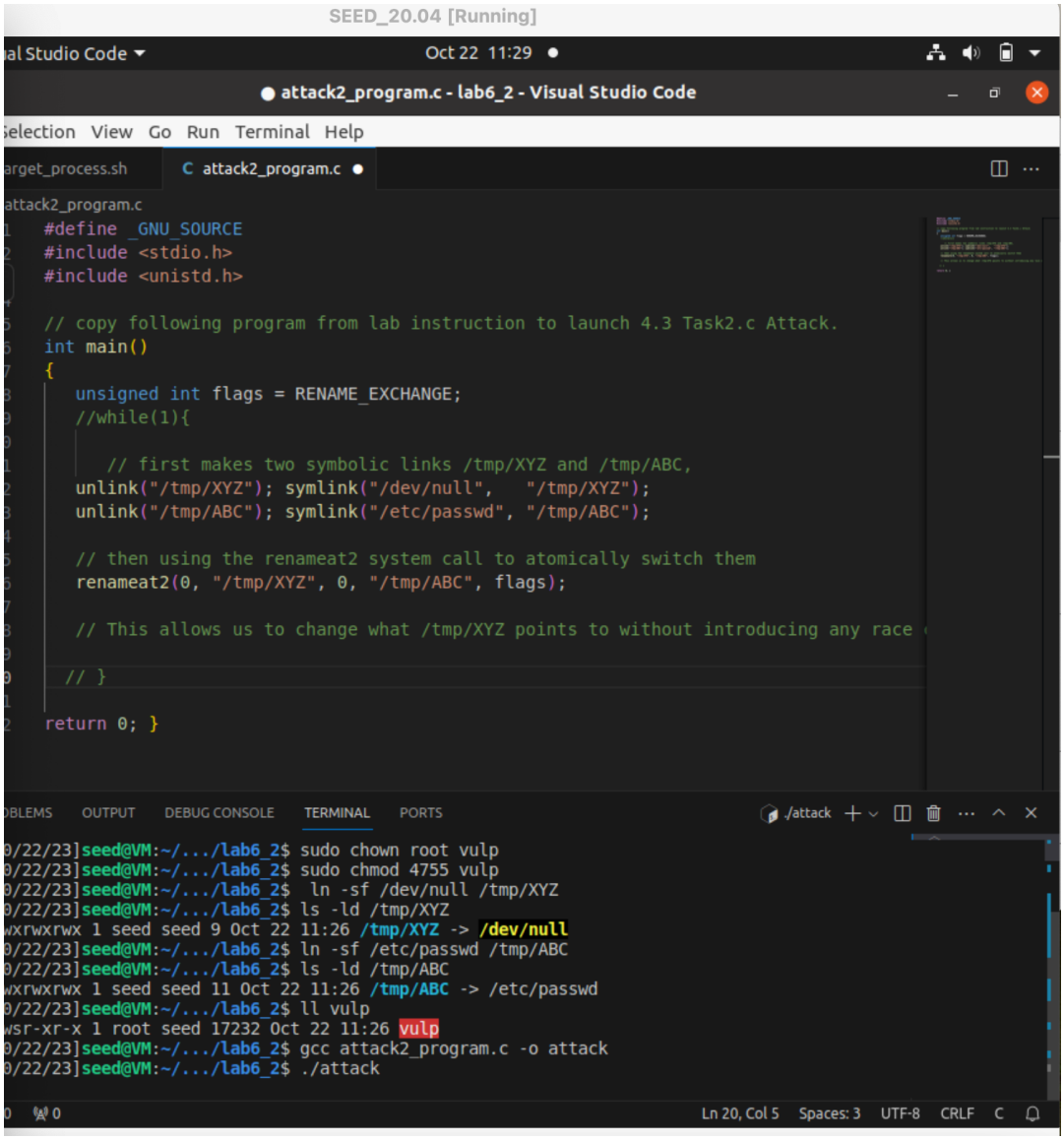


// This screenshot shows how I demonstrated the principle of least privilege. I studied this through lecture ppt. Line 20 : setuid(realUID) will set EUID = RUID and drop unnecessary privilege.



// code for vulp.c

// above command line shows that I added setuid() codes and recompiled. I set the owner to be root and set SET-UID. Created symbolic link /tmp/XYZ to point file that we owned and anybody can easily write which is /dev/null and then I check /tmp/XYZ files symbolic link by typing “ls -ld /tmp/XYZ”. Same thing for /tmp/ABC. Create symbolic links and points to /etc/passwd and compile the attack program (attack2_program.c). Launch attack.



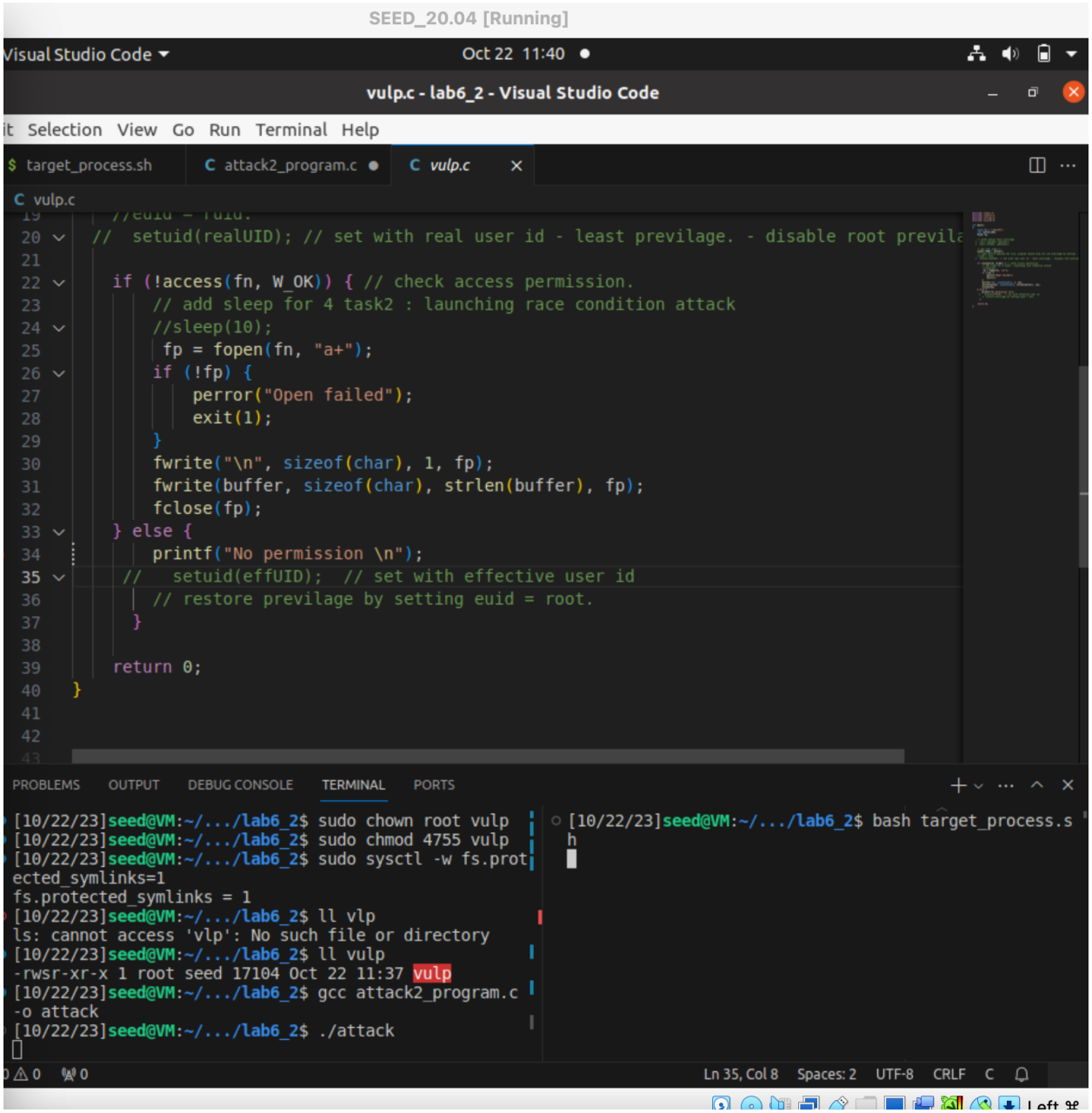
// code for attack2_program.c

5.2 Task 3.B: Using Ubuntu's Built-in Scheme

Ubuntu 10.10 and later come with a built-in protection scheme against race condition attacks. In this task,

you need to turn the protection back on using the following commands:

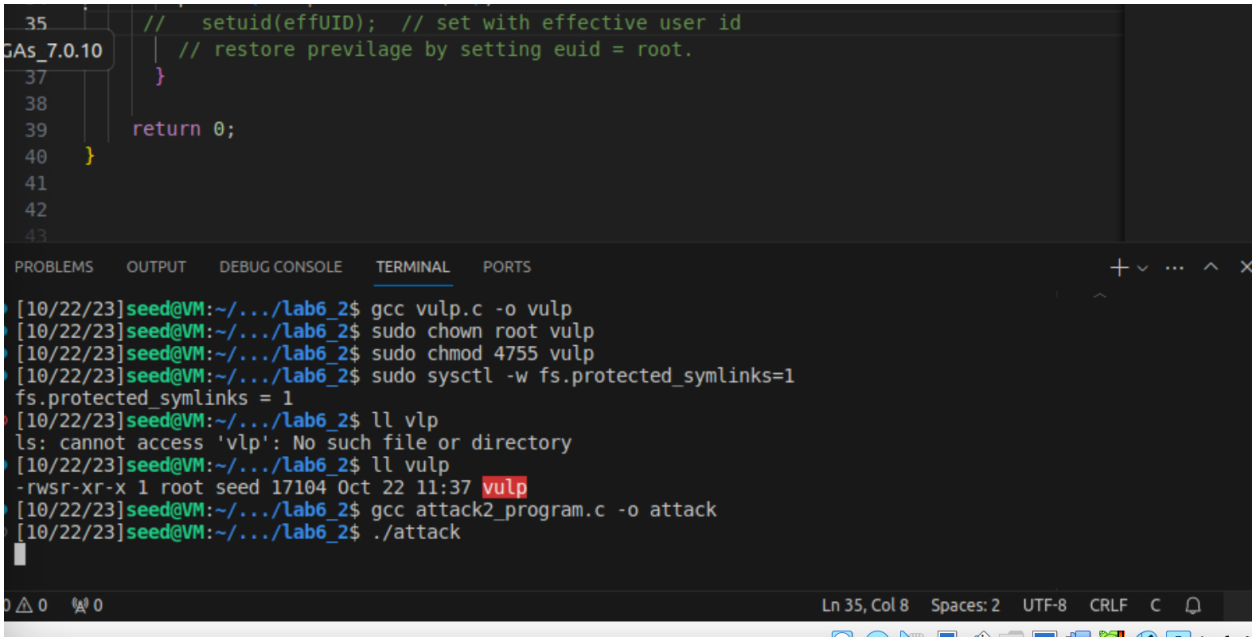
Conduct your attack after the protection is turned on. Please describe your observations. Please also explain the followings:



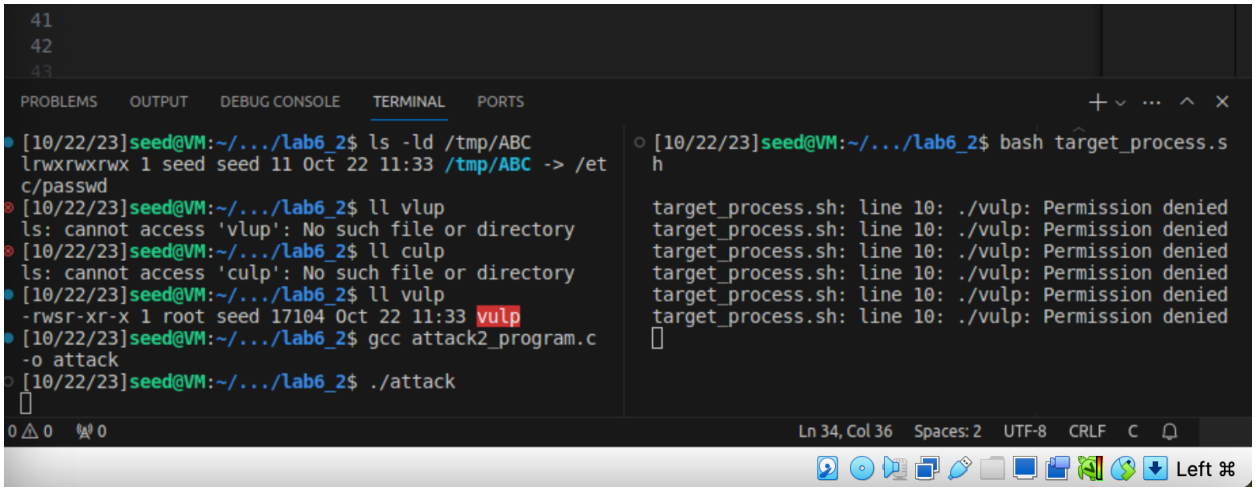
// I removed all the code that related to set uid, least privilege principle in vulp.c.

// I set owner to be root and set-uid and recompiled vulp.c

// I turned on Ubuntu's built-in protection scheme against race conditions by setting symlinks = 1. I compiled attack2_program.c and launched parallel attack .



// This terminal screenshot shows that I recompiled vulp.c, set owner to be root, changed to SET-UID privilege and turned on the Ubuntu built-in protection. And then launch an attack.



// As we can see I was unable to attack with Ubuntu's built-in Scheme.

(1) How does this protection scheme work?

Explanation about Sticky Symlink Protection for Ubuntu built-in Scheme.

Other users can not open vulnerable files or programs even if there are vulnerable race conditions because of Ubuntu built-in Scheme. Based on the lecture slide resource. When sticky symlink protection is enabled (e.g. fs.protected_symlnks=1) , symbolic links inside of sticky world-writable will only follow the symbolic links if and only if owner of symlink matches either the follower(euid) or the directory owner. For example, If follower(euid) = seed , directory owner(= seed) and symlink Owner = root then fopen() will be denied and print out an error message.

In our vulnerable program, follower (euid) is root. And /tmp/ directories are owned by root, but if symlink owner is not root, then the program will not be allowed to follow symbolic link pointers. Since symlink will be created by an attacker (not root!) Ubuntu built-in scheme won't follow symlink and prints out that the program crashed, but not attacked.

(2) What are the limitations of this scheme?

Based on the terminal output and above answer. Ubuntu built in Scheme will not allow to open fopen() files, if symlink owner is different with follower(euid) and directory owner. However, Scheme wasn't able to stop the race condition, because I kept getting output that we are unable to open files and it did not stop. I had to kill the terminals that I used to launch this parallel attack. Additionally, Ubuntu built-in scheme will only work with word-writable sticky directories (e.g. /tmp/ABC, /tmp/XYZ, /tmp). (not works for all directories).