CYSE 211

Mahish Mahendarkar

7/7/2021

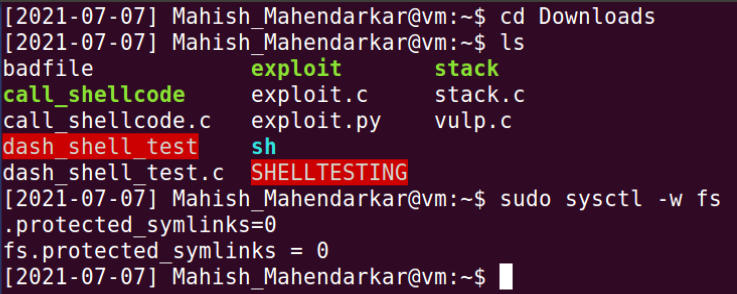
Race Condition Vulnerability Lab 5

Lab 5 purpose:

The purpose of this lab is to develop the right environment and files to carry out a race condition-based attack. The lab’s main task involves escalation of privileges to root as well as disabling counter measures built into the OS, shell, and compiling programs. There are three key parts to the lab: successful exploitation of race condition vulnerability, symlink protection, and the principle of least privilege. The tasks involve the successful race condition attack being carried out, as well as a set number of failures due to certain protections that will be enabled either by default or by the user as a test of the protection.

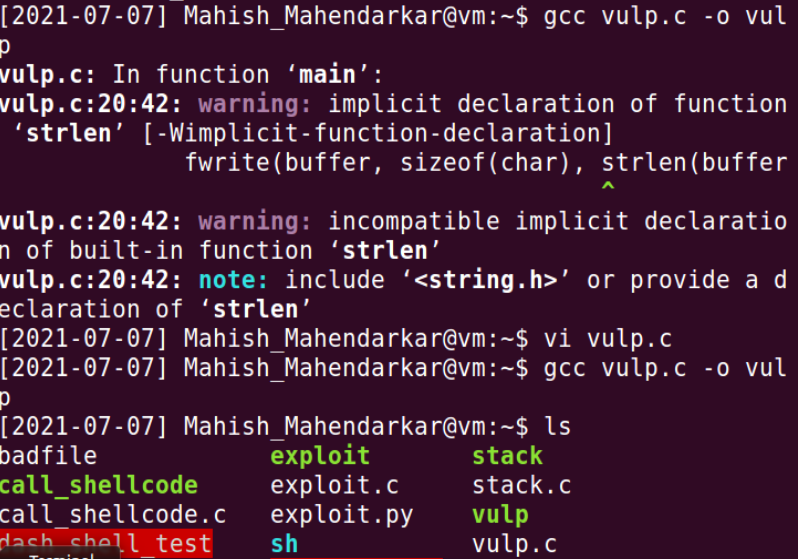
---------------------------------------------------------------------------------------------------------------------

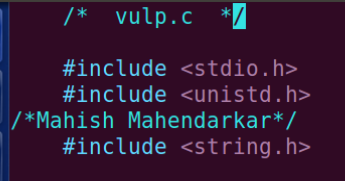
Pre-Lab configuration settings:

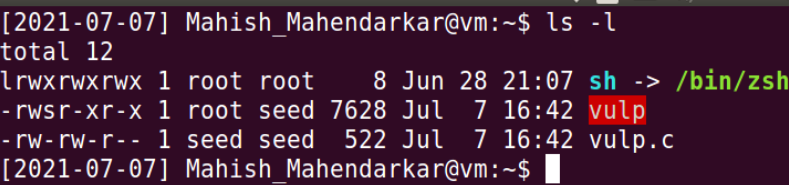


**Set up the Set-UID program:**

This is the setup required for vulp.c to be given root privileges. We will be interacting with the compiled vulp.c named vulp. Initially the strlen function is not recoginized so the string.h library is added to [vulp.c]. The third screenshot shows the permissions associated with vulp. NOTE initially I was using a clone of Lab4 that is why there are random unrelated files in this screenshot.





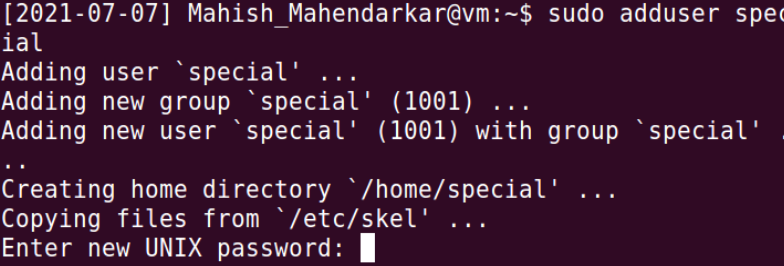


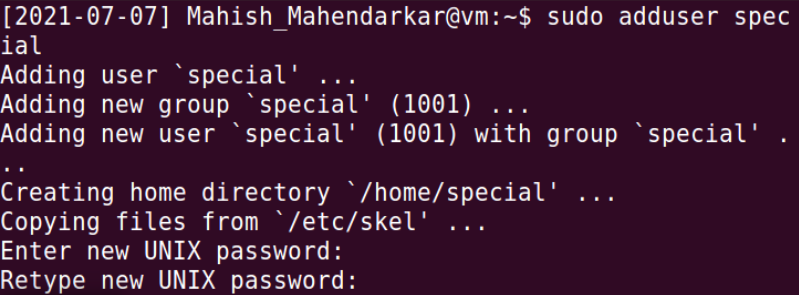
Lab tasks:

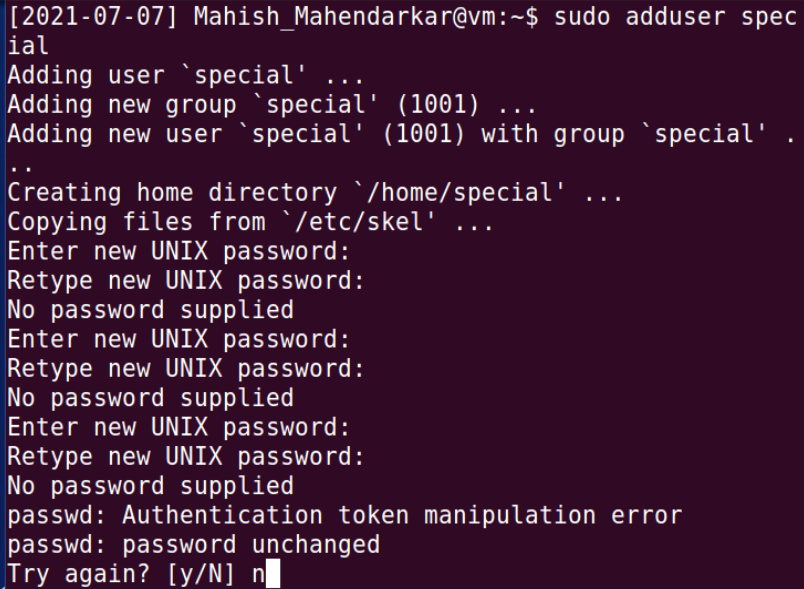
**Task 1: Choosing Our Target**

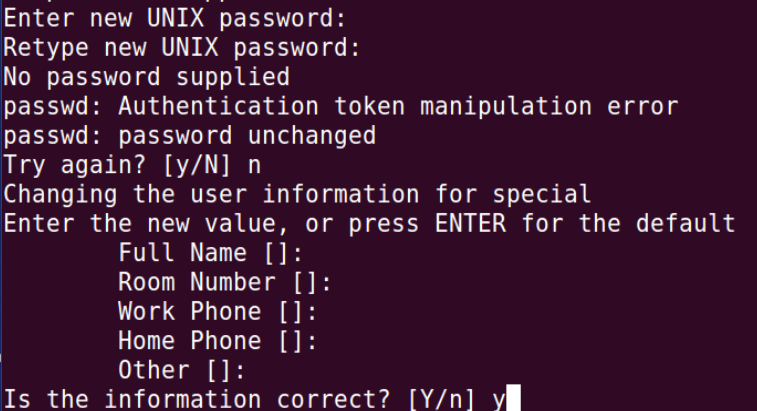
This task explains how to specify a password for an account in Linux with a root Process User ID of 0, This is the third field from the left [root:x:0:0:root:/root:/bin/bash]. We manually add a password instead of the OS referring to the [/etc/shadow] directory. This can be done by hashing [dees] with the encryption scheme of the system (SHA512), or by copying the hashed password generated into the [/etc/shadow], or by using a special Ubuntu magic value that sets the password to “nothing” which allows the pressing of the enter key to be a valid password.

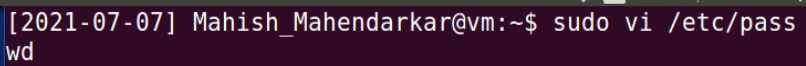
Observations: The first part of this task is to create a new user that will have root level privileges. I first create a new user. As I create the user named [special] I hit the enter key when prompted for a new Unix password. The system notices the lack of a password and raises an error and I proceed with [enter key] as my password three times, finally I can confirm this as my chosen password. I use the default values for all the fields in screenshot 4 and confirm my choice. I then use [sudo vi /etc/passwd] to edit the [/etc/passwd] file according to the specifications in the lab instructions (IMPORTANT NOTE: the instruction have a new user named test I used special). I then attempt to login to the user account [special] and as the last screenshot shows no characters have been entered in the password prompt because I only hit the enter key. This is successful meaning the magic value I added worked appropriately. The last commands confirm my root status and directory location that has been defined in the vulp.c source code. Once this has been shown to work, I remove the [special] entry entirely and move on to the next task.

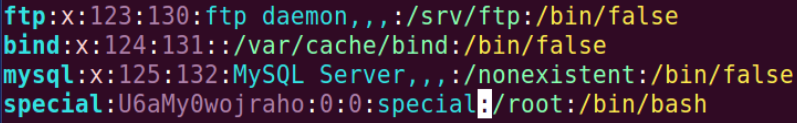


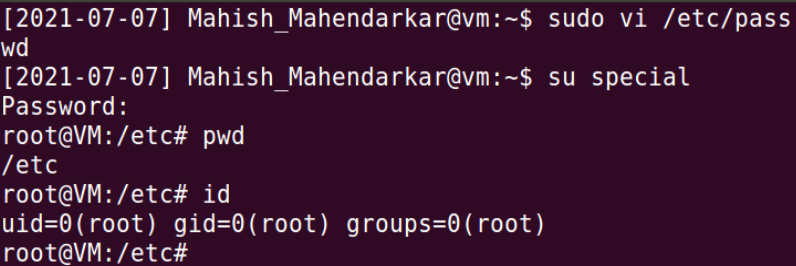


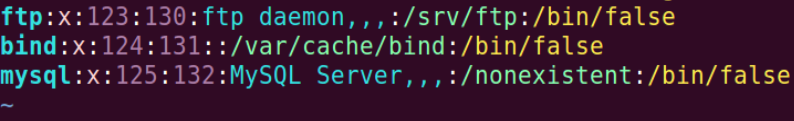










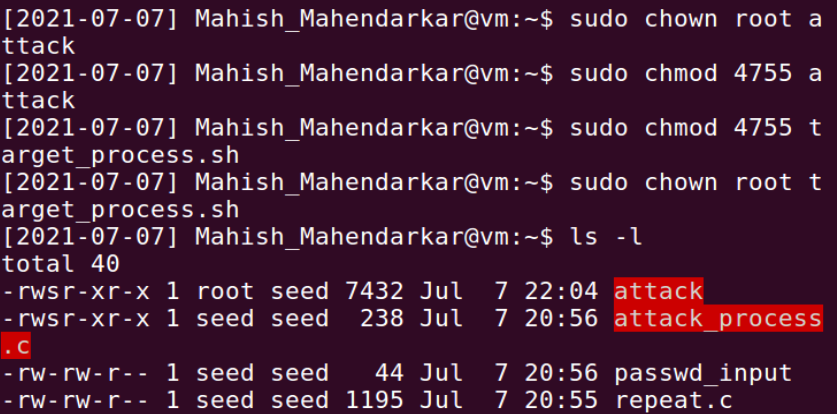


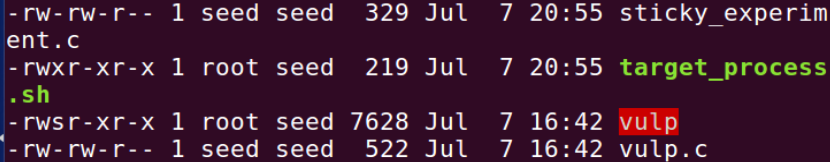
**Task 2.A: Launching the Race Condition Attack**

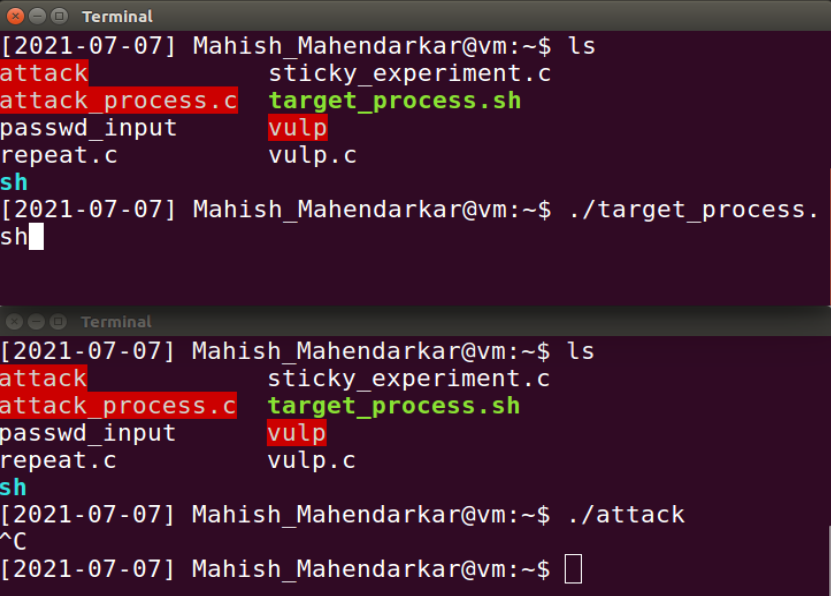
This task is concerned with launching a successful race condition attack. The first step is to download the pre constructed files provided via github. The contents of these files do not need to be modified. The permissions do need to be upgraded and the full permission [vulp.c] file must be brought to the same directory location as well. Then the attack can be carried out.

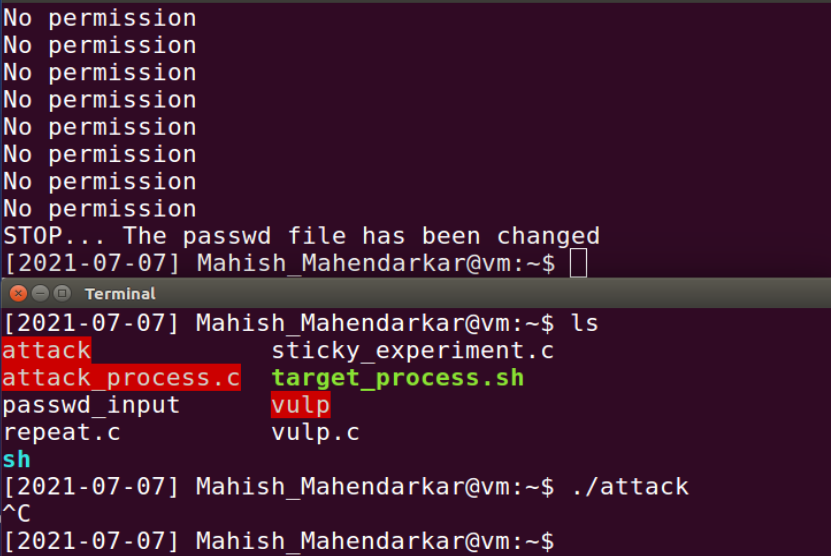
Observations: Initially I was unsure of how to carry out the attack until I realized the files had been provided. The contents of the files can be summarized as: TARGET.sh opens and reads a file, and loops through this action many times, ATTACK changes the contents of a file that is being read right before it has been read. The passwd\_input file holds the magic key. This key is fed to the reading process when the attack process can replace the location being read from /tmp/XYZ to /etc/shadow. This reads off the password stored in passwd\_input. These details are present when going through the provided files in detail. It can make the directory change because of the full permissions given. This enables the attack to successfully use the magic number password and gain elevated privileges.







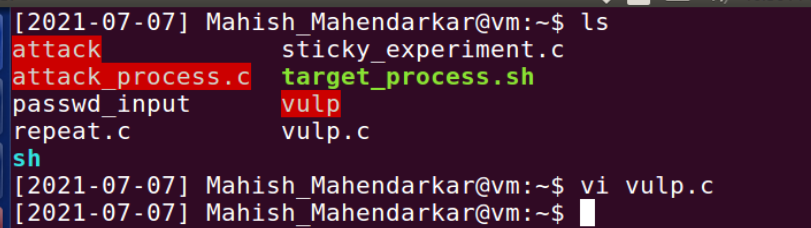


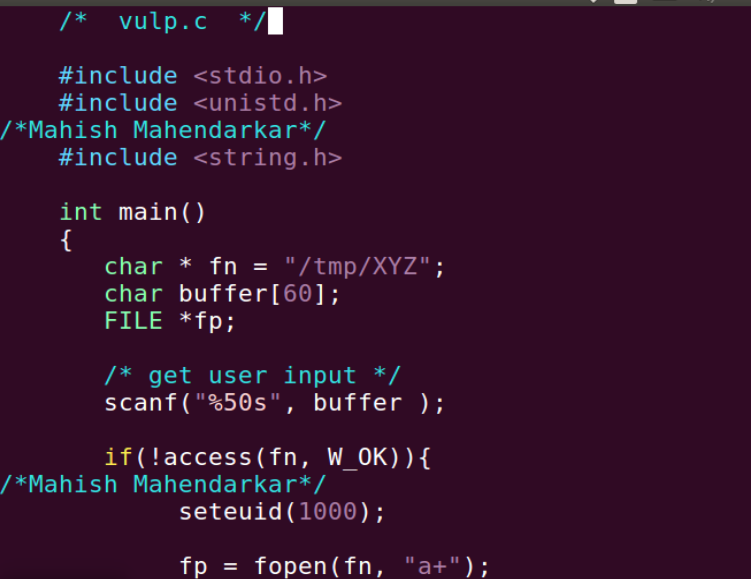


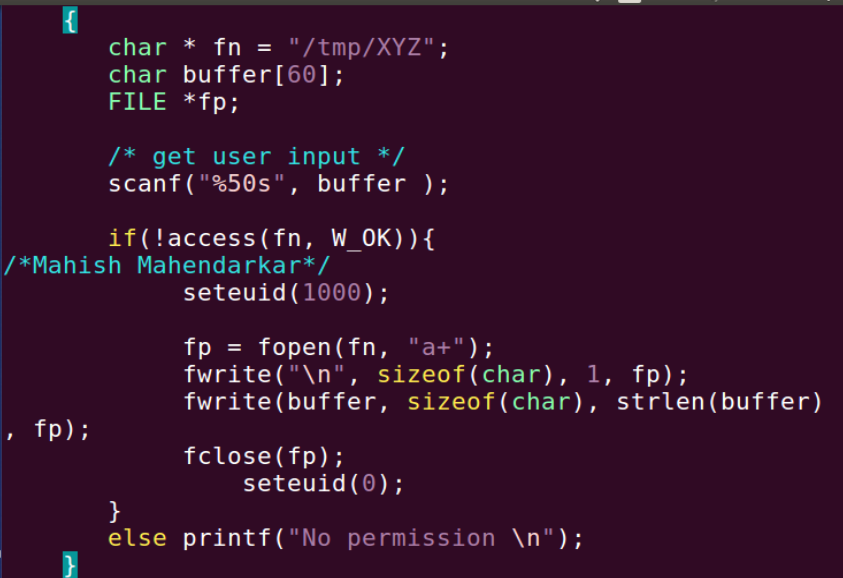
**Task 3: Countermeasure: Applying the Principle of Least Privilege**

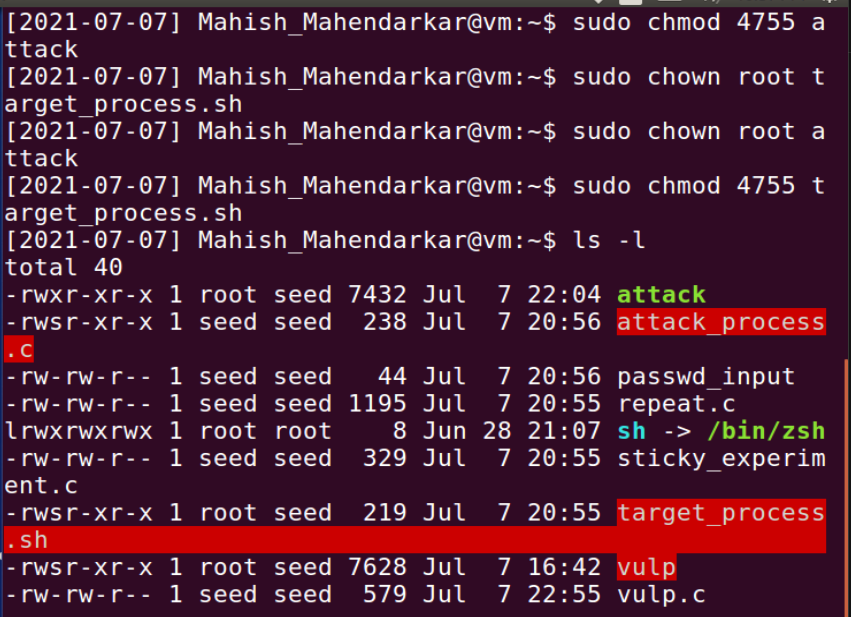
This task puts specific system call level commands meant to disable root privileges into the vulnerable program to be compiled.

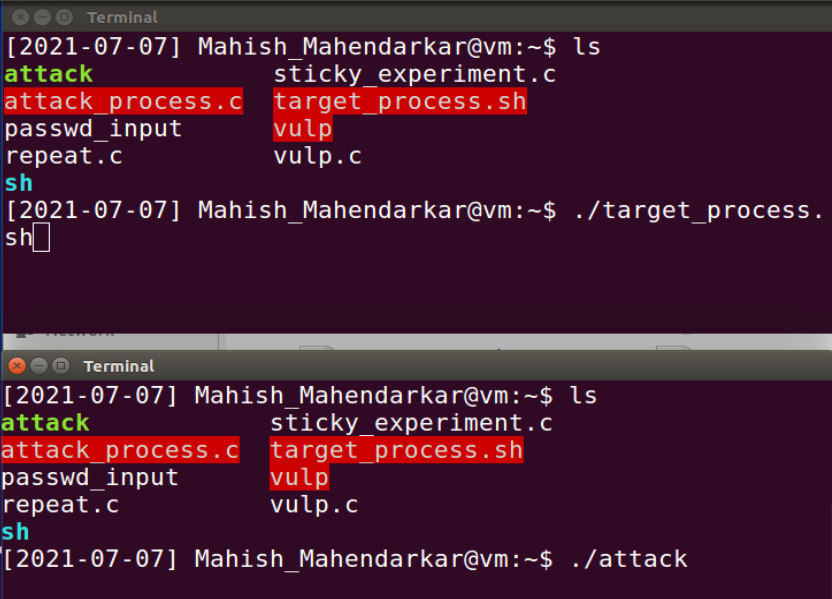
Observations: The first part of this task involves changing the vulnerable file[vulp.c] to use the Linux system command of [seteuid] to ensure that root permissions would not be enabled during the attack process for this vulnerable file. Then the attack and target executables were given their full permissions (Screenshot 4). The attack was carried out and there was no printed output. The exit command of [ctrl+c] had to be used and this attack was unsuccessful as expected, a success!

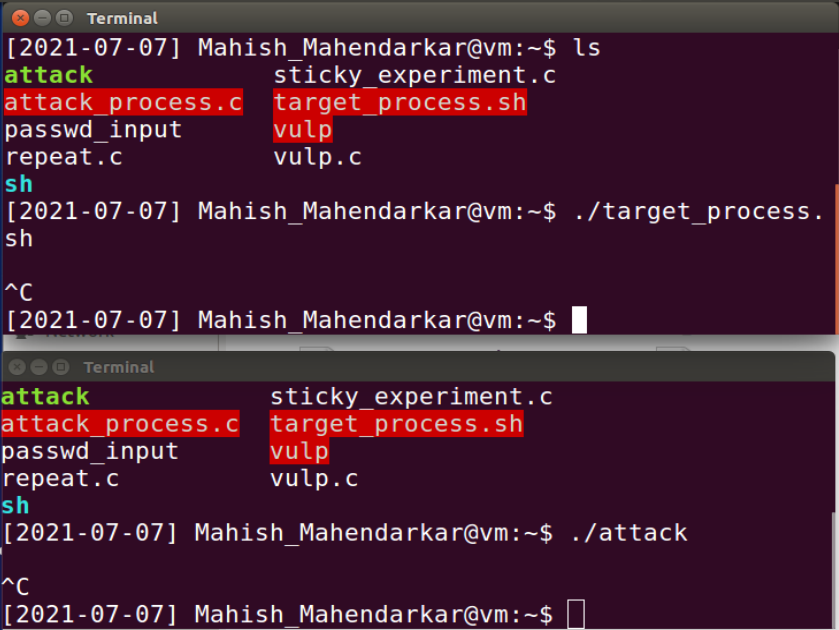








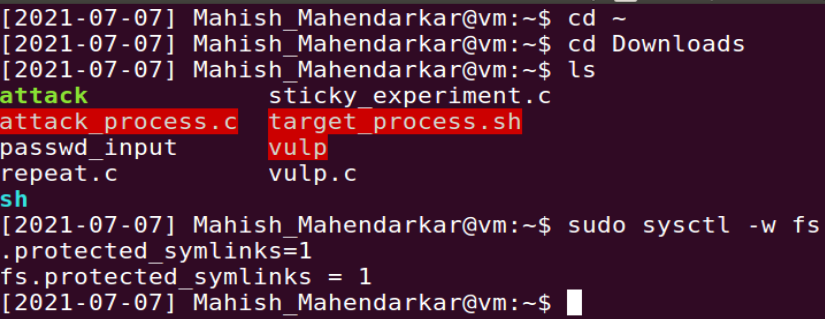


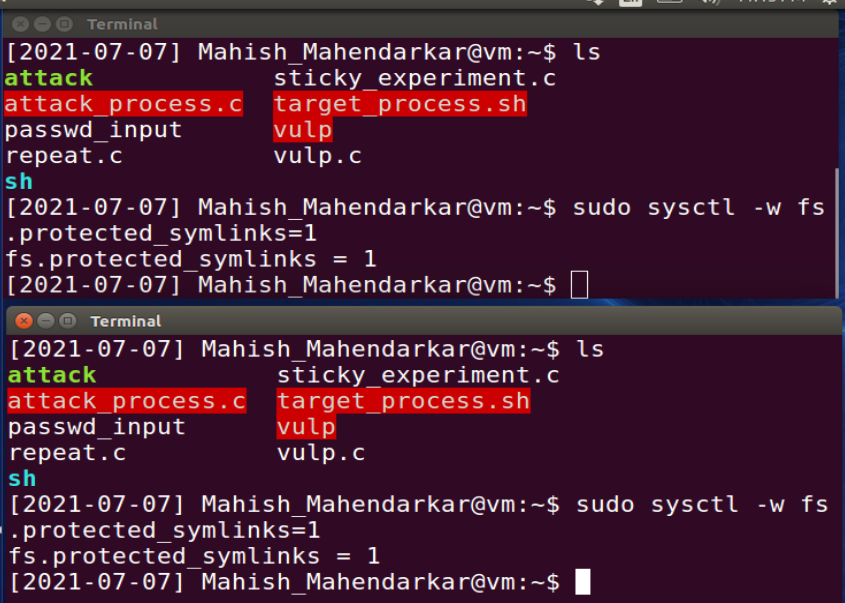


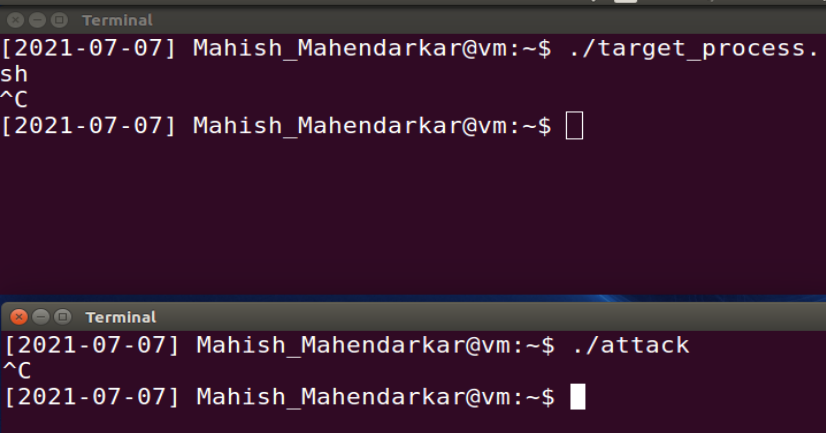
**Task 4: Countermeasure: Using Ubuntu’s Built-in Scheme**

This task involves enabling the Ubuntu protection initially disabled in task 1. Then the attack is conducted with the OS level protections enabled.

Observations: The setup prior to this attack was similar to the previous tasks. The full execute permissions are enabled for the necessary files as indicated by the red highlight. The attack is unsuccessful as expected.







**Questions:**

How does this protection scheme work?

The last protection scheme works by not allowing a new symbolic link action to be carried out. This stops our attack executable from changing the file being read in a loop by our target script. The attack file works by changing the symbolic link to a file that is in a protected directory[/etc/shadow].

What are the limitations of this scheme?

This scheme only works when the directory has sticky bits enabled. In a directory owned by root the race condition might still manage to be successful.