Lab 3 Report
Comp 435
Joshua Trzcienski

Race Condition Vulnerability Lab

In this lab, the objective is to look at the race-condition vulnerability, which can happen when there is are multiple different processes that are trying to access the same data, and the order that they happen can effect the outcome. In the examples below, you can run multiple programs at the same time in order to try and exploit some of these potential weaknesses, and do something that should require root privilege without having it.

Task 1: Choosing Our Target

The purpose of this task is really to show that the "magic" password works correctly, and when prompted to login and type in a password, no password is needed. In Image One below, the very last line has an added line starting with "jtrzc", which is the name of the user that was added, and the magic password is added so no password is actually needed. In Image Two, su – jtrzc will switch the user to jtrzc, and ask for the password, which can be seen in the image, is not needed. After enter is pressed, a root shell appears. This shows that the magic password, but using root privileges in order to add this to the password file is not really an attack, but real attacks will be seen in the later tasks.

Image One

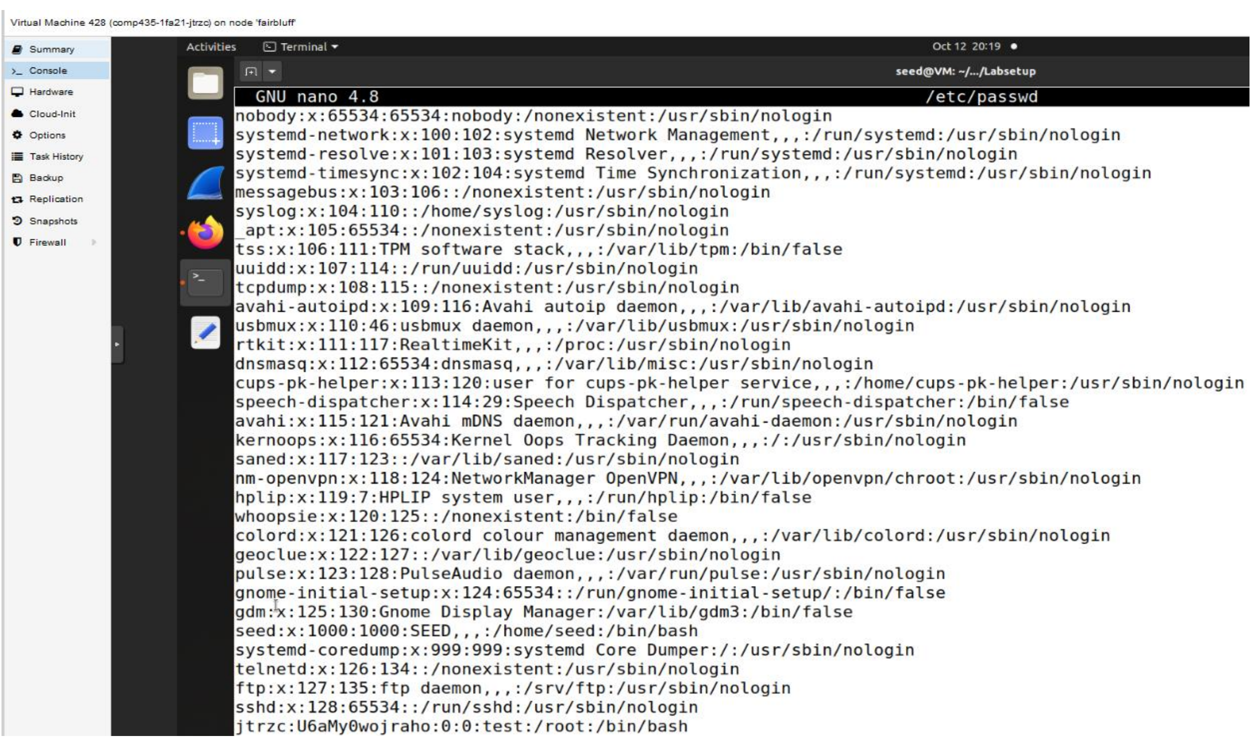
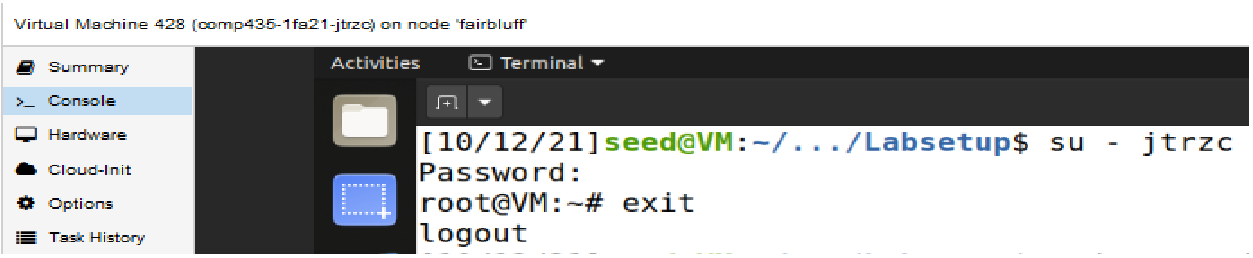


Image Two



Task 2A: Simulating a Slow Machine

Now instead of using sudo to amend the password file, a “real” race condition will be exploited in order to add the new user and gain root privilege. In vulp, sleep(10) is added to simulate a computer that is really slow, and this give time to manually make /tmp/XYZ a symbolic link to /etc/passwd in the 10 seconds the computer is waiting. Instead of vulp writing to /tmp/XYZ, it will now write to the password file, because during this time the computer is sleeping, aka simulating being slow, you can change the symbolic link of /tmp/XYZ to /etc/passwd. In Image Three, the third line where it says ./vulp is where the program first starts, and then the input is given right after in the line below. Right after the input is given, in Image Four in a separate shell, the symbolic link is changed during the time that the computer is sleeping. Then when it is finished, jtrzc is a user and is able to login in with root privilege.

Image Three

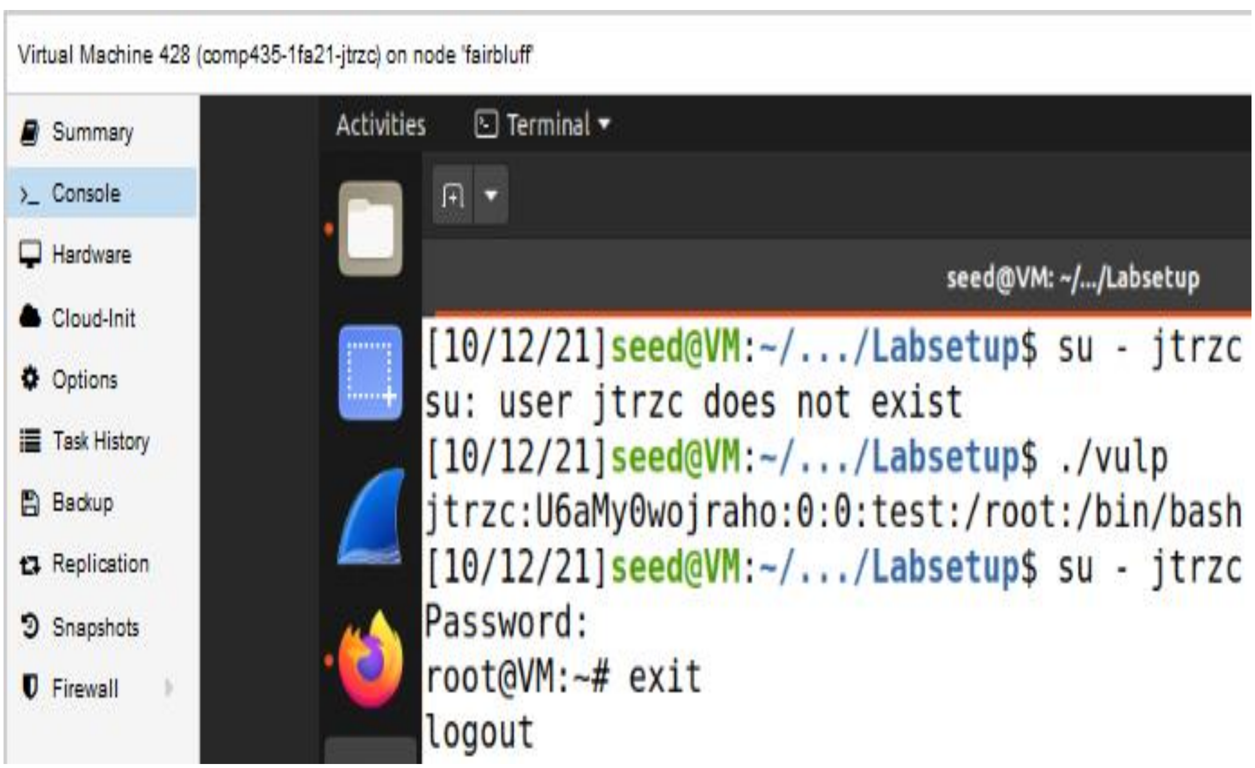
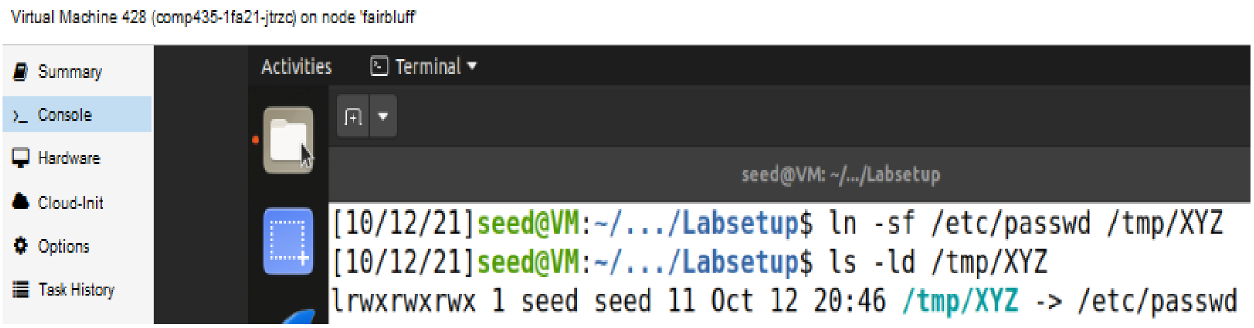


Image Four



Task 2B: The Real Attack

In this attack, instead of simulating the computer being really slow, and adding sleep(10), it will be more realistic, and sleep(10) will be removed from vulp. The premise for the attack remains the same, now the timing is much more difficult, and pretty much impossible for a human to achieve by manually unless they were extremely lucky. Instead of manually changing the symbolic link, a program was written to continually do this, and this program can be seen in Image Nine. All it is going to do is infinitely loop, linking and unlinking, with a very small delay in between. While that program is running, a shell script is also being run, which will keep on running vulp, and this can be seen in Image Seven. In the shell script, vulp is being run, taking input from text.txt, which is Image Eight, and all that is is the string to add to the passwd file that was used in the previous attacks. With the shell script and the attack program both being run, eventually once the timing aligns, jtrzc will be added as a user to /etc/passwd and will not need a password as well. After about 3 minutes, the shell script detected a change in /etc/passwd, seen in Image Five, and in Image Six, it can be seen that the attack was successful, and jtrzc was added with no passwd, and after switching to it, a root shell was opened.

Image Five

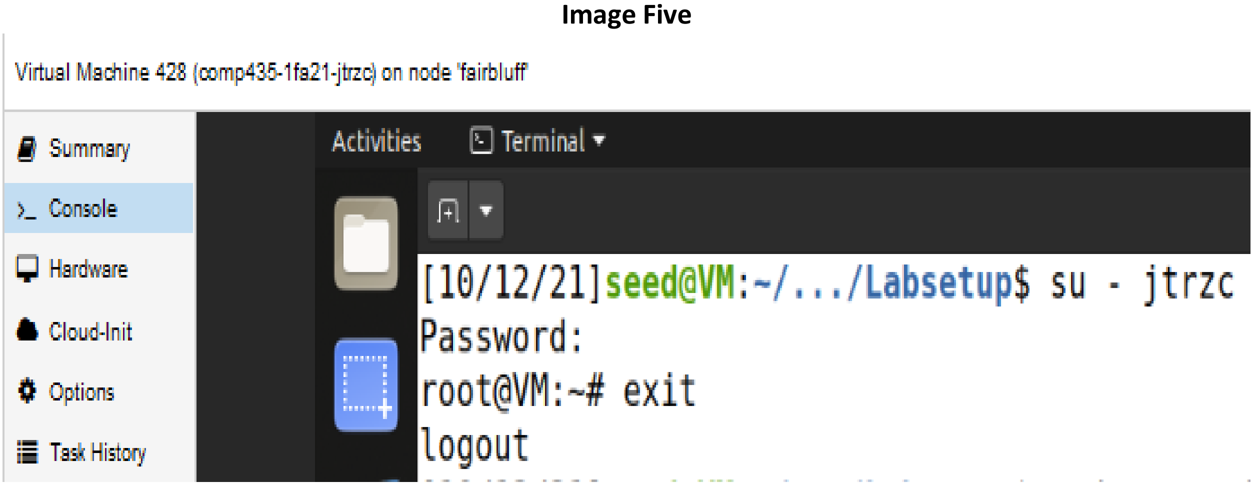


Image Six

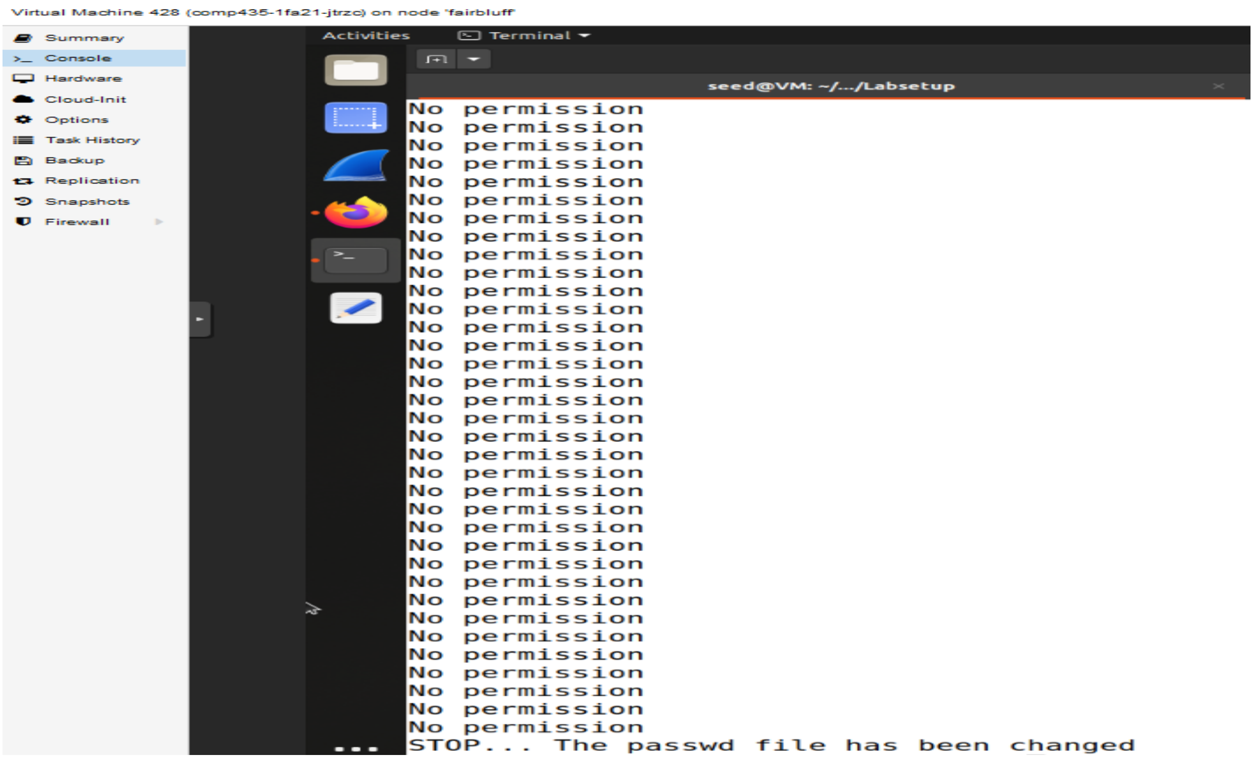


Image Seven

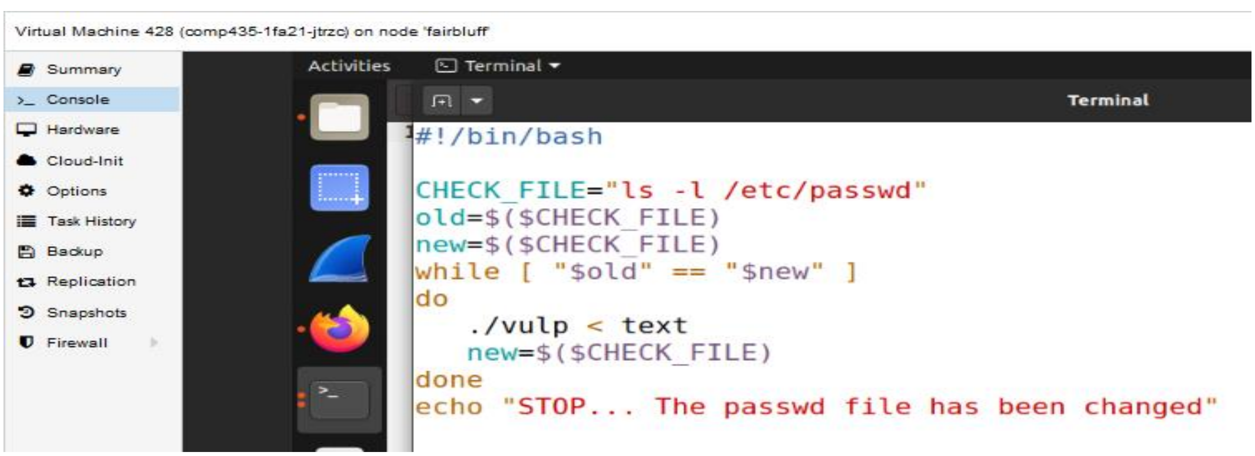


Image Eight

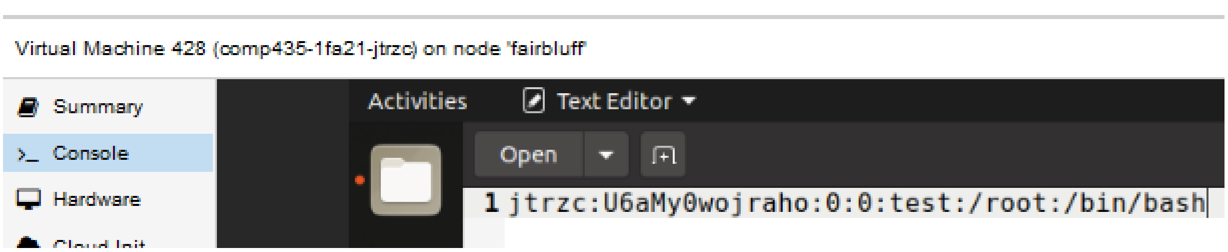


Image Nine

Task 2C: An Improved Attack Method

This did not happen when I was doing Task 2B, but it was possible that the owner of /tmp/XYZ could become root, which would essentially defeat any chances of the attack succeeding. In order to fix it, you would need root to delete the file, which defeats the purpose of an attack to gain root, if you need root to accomplish the attack. With this in mind, the code that links and unlinks is slightly changed in order to avoid this race condition that could make the owner of /tmp/XYZ root. Image Eleven contains the edited attack code, and instead of just doing link and unlink, renameat2 is used to atomically switch the two symbolic links. This allows us to make the changes without having the risk of the attack being defeated by a race condition itself. The shell code, Image Twelve, remains unchanged from the previous attack, and the successful attack can be seen in Image Ten.

Image Ten

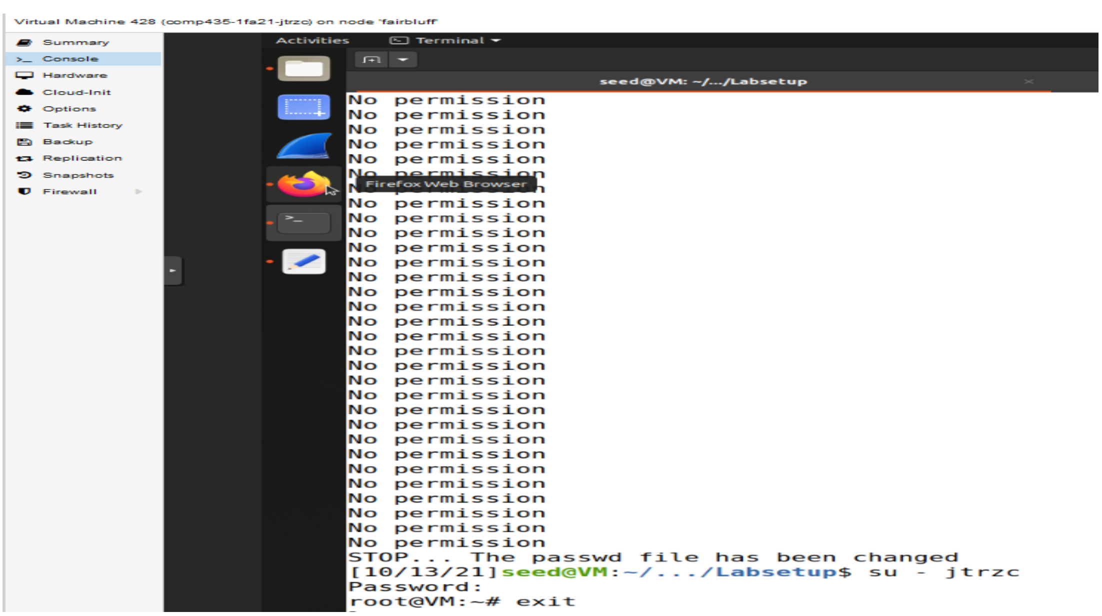


Image Eleven

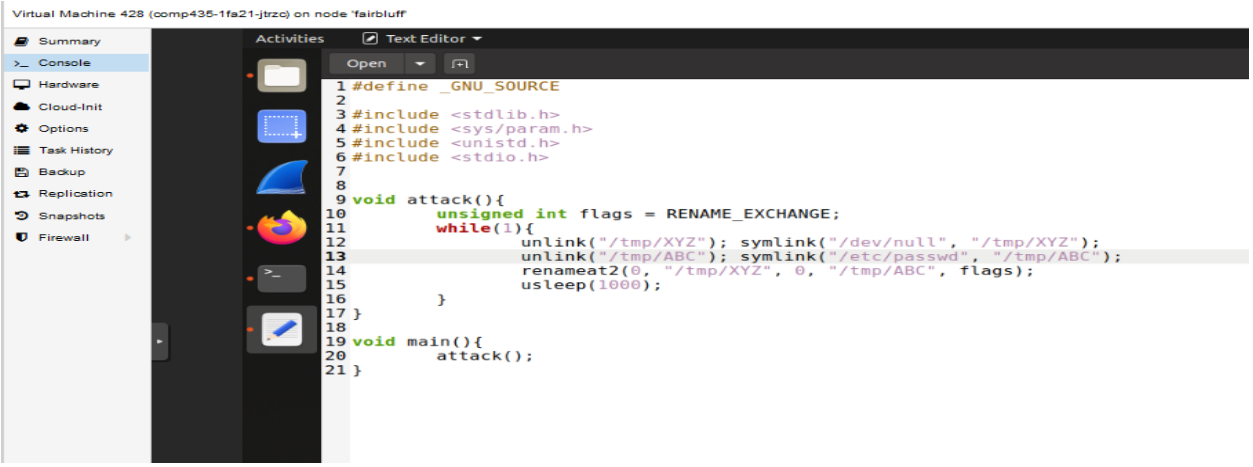
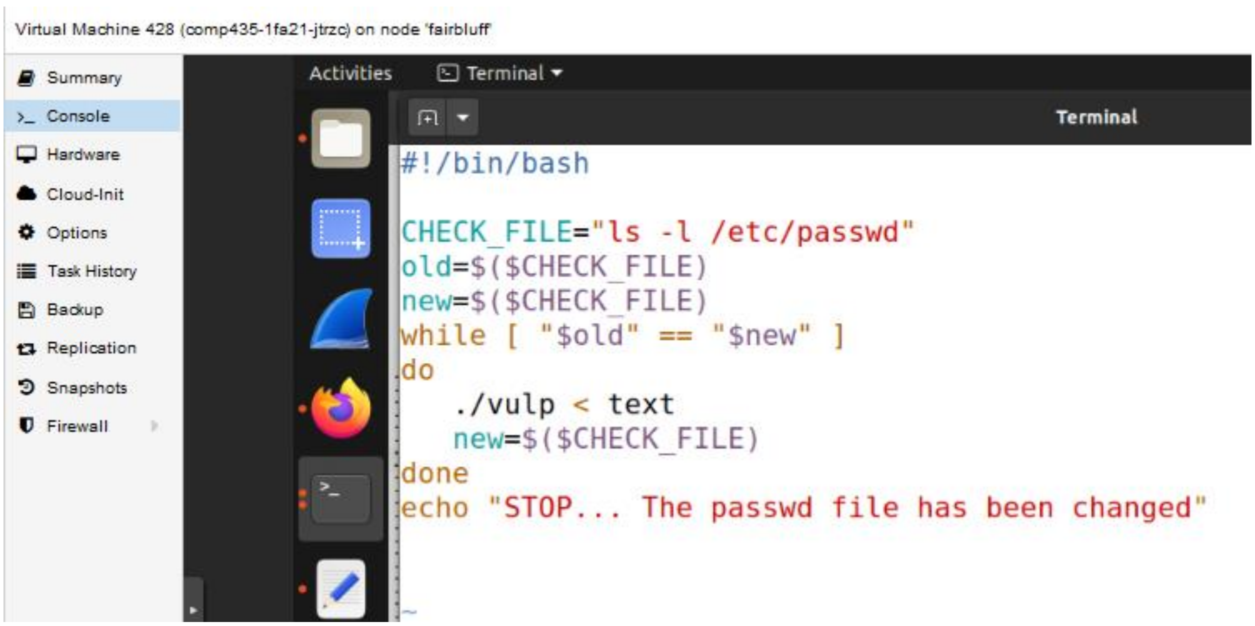


Image Twelve



Task 3A: Applying the Principle of Least Privilege

One of the reasons that the earlier attacks were able to be successful was due to the fact that vulp was not operating under the principle of least privilege. It did not need root level privilege to write to /tmp/XYZ, so this leaves potentially vulnerabilities for attackers, as seen in the above attacks. In this task, root level privilege will not be granted when it is opening the file and writing to it. The updated vulp.c file can be seen in Image Fourteen, and the changes that were made was that first the euid and the uid are taken, and it checks for access, the euid is set to the uid, dropping privileges. Once it is done, the euid is set back to what it was, but now, there is no window to try and exploit, and in Image Thirteen, it can be seen that now when the attack is run with the shell script, the attack is unsuccessful.

Image Thirteen

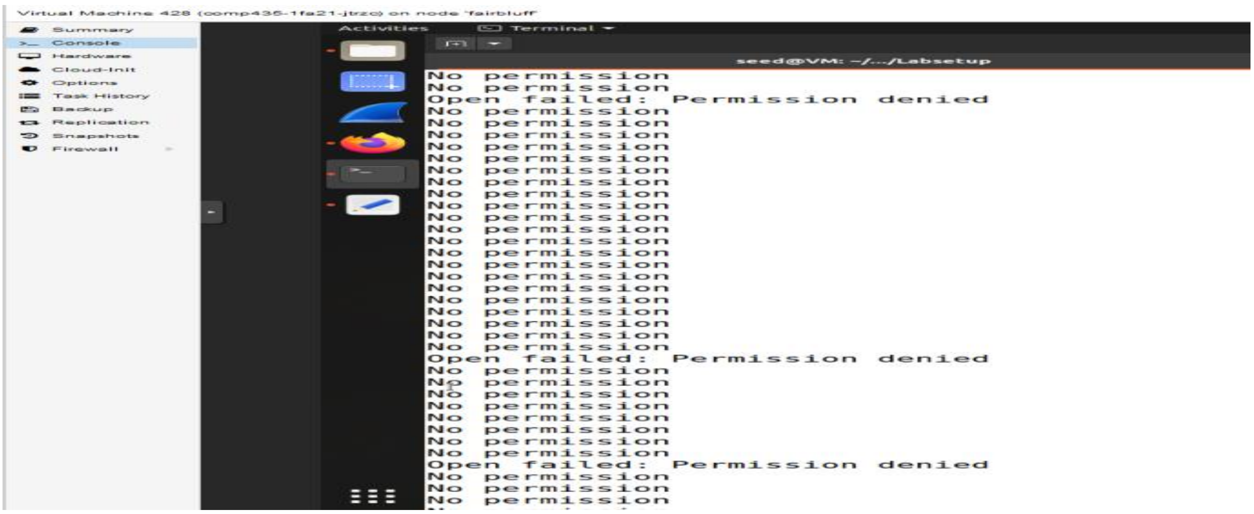
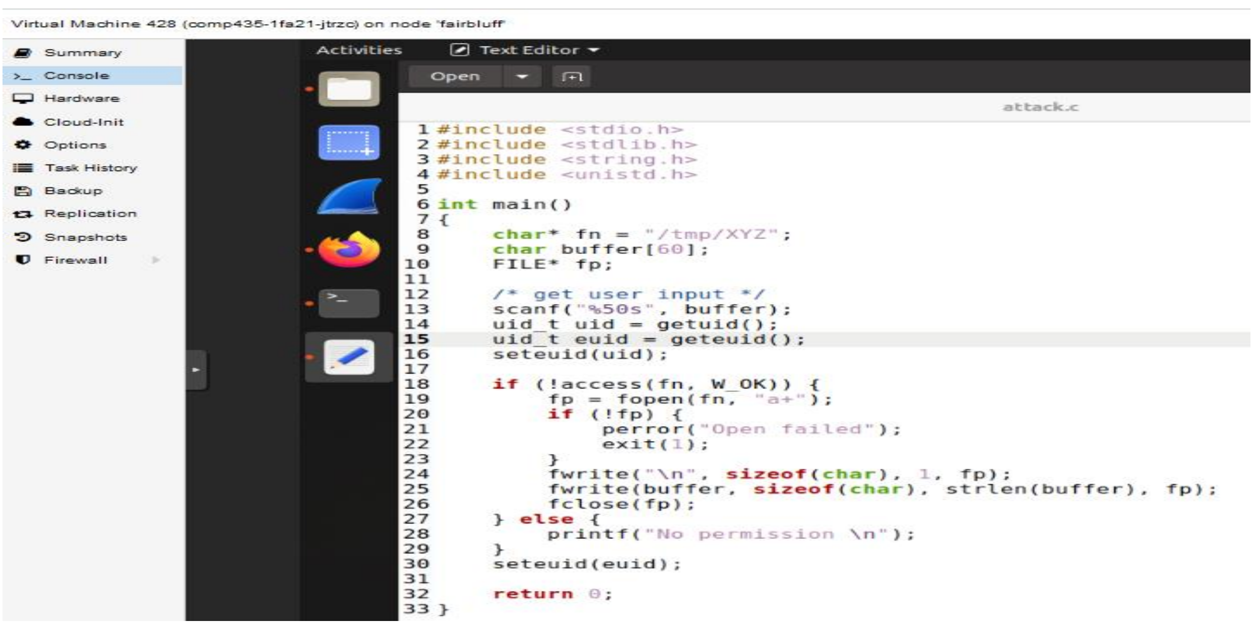


Image Fourteen



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

With the protections turned on, the attack once again is not successful. Both the shell and attack program would run continuously, and due to the protections being turned on, the way this attack is being run will not succeed, as seen in Image Fifteen. The protection scheme makes some changes to whether or not a symbolic link is actually followed. With it turned on, symlinks are only allowed when they are not used in some sort of sticky world-writeable directory, which /tmp is a world-writeable directory. If it is in some world-writeable directory, the symlink is only allowed when the uid of the symlink and the follower are the same, or if the owner of the directory is also the owner of the symlink. This defeats what is being done in the attack program, as vulp has root privilege and /tmp also is owned by root, so in order for a symlink to work, the symlink owner would have to be root level as well. This countermeasure is effective with an attack using a world-writeable directory, but there are some limitations. As stated above, these protections are only in world-writeable directories, so it is only a countermeasure in those type of directories, not all directories. This also does not automatically prevent race conditions, as all it is doing is preventing symlinks in certain scenarios.

Image Fifteen