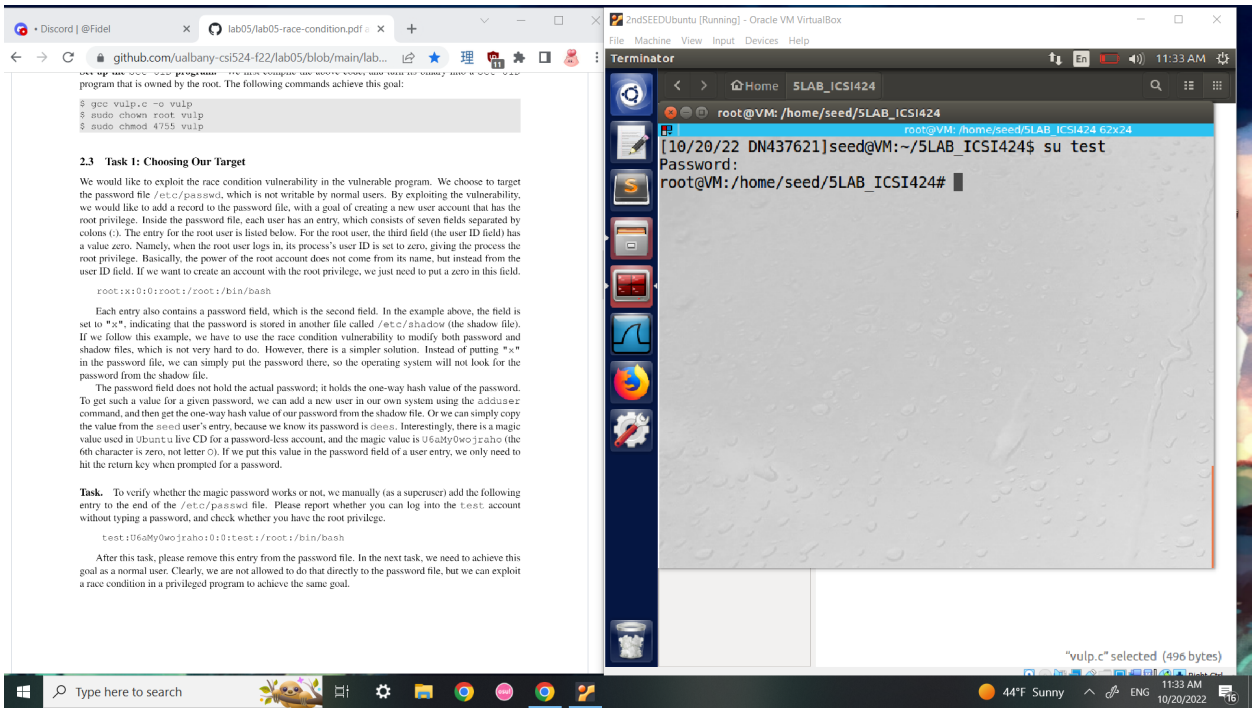
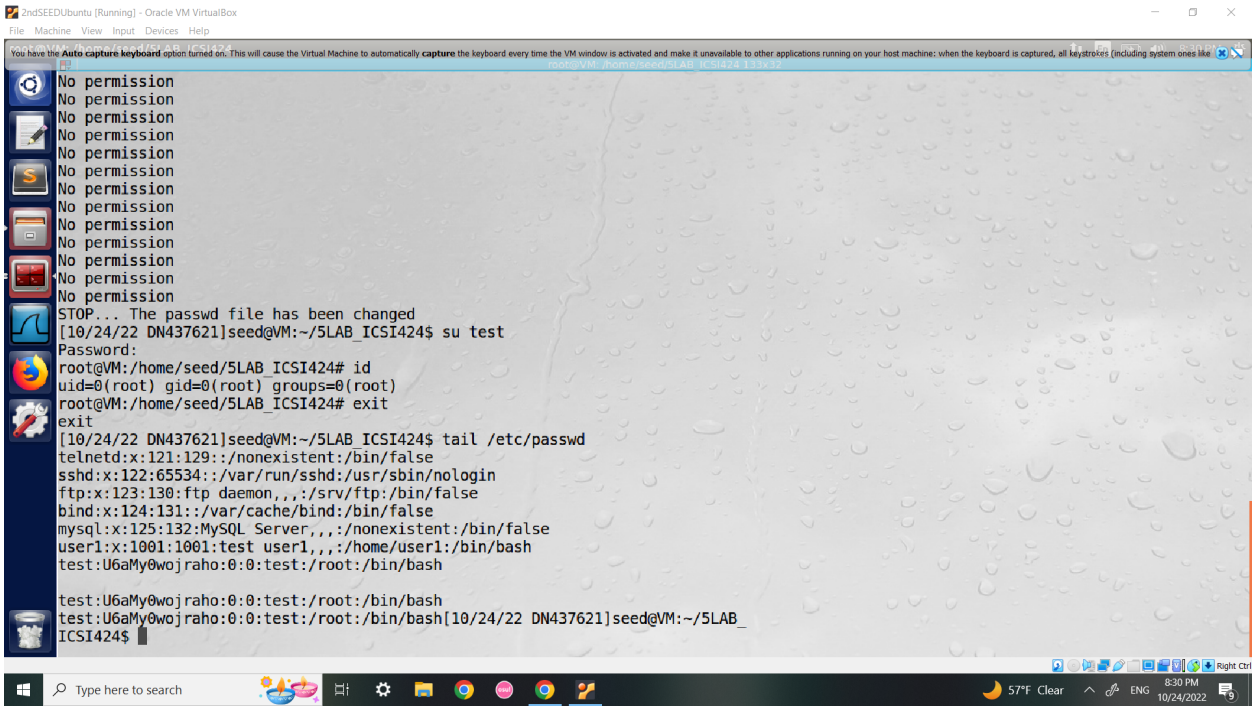
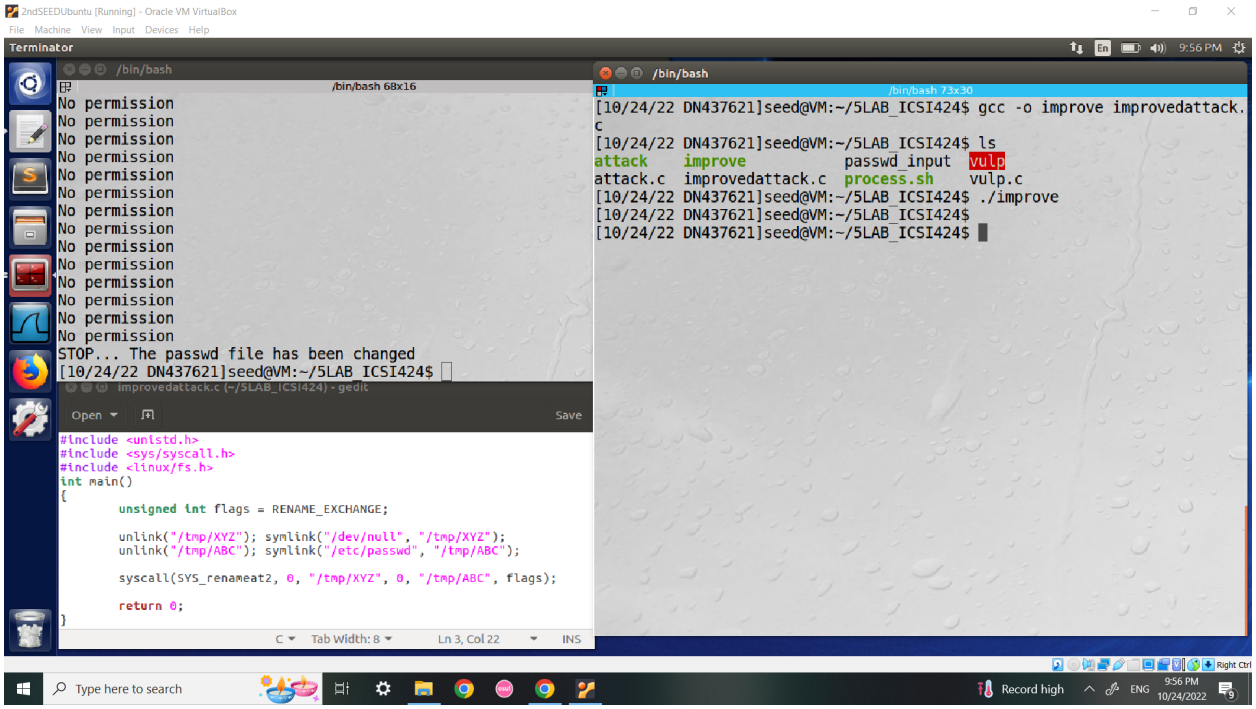
Task 1: Before doing Task 2a, we must verify if we are able to see if the magic password value can gain root access with no password use. By doing su test, then pressing enter at the password prompt, we can see that we have entered root. Indicated by the far left, we will then remove this entry, so that we can have this same result but with vulp.c instead.



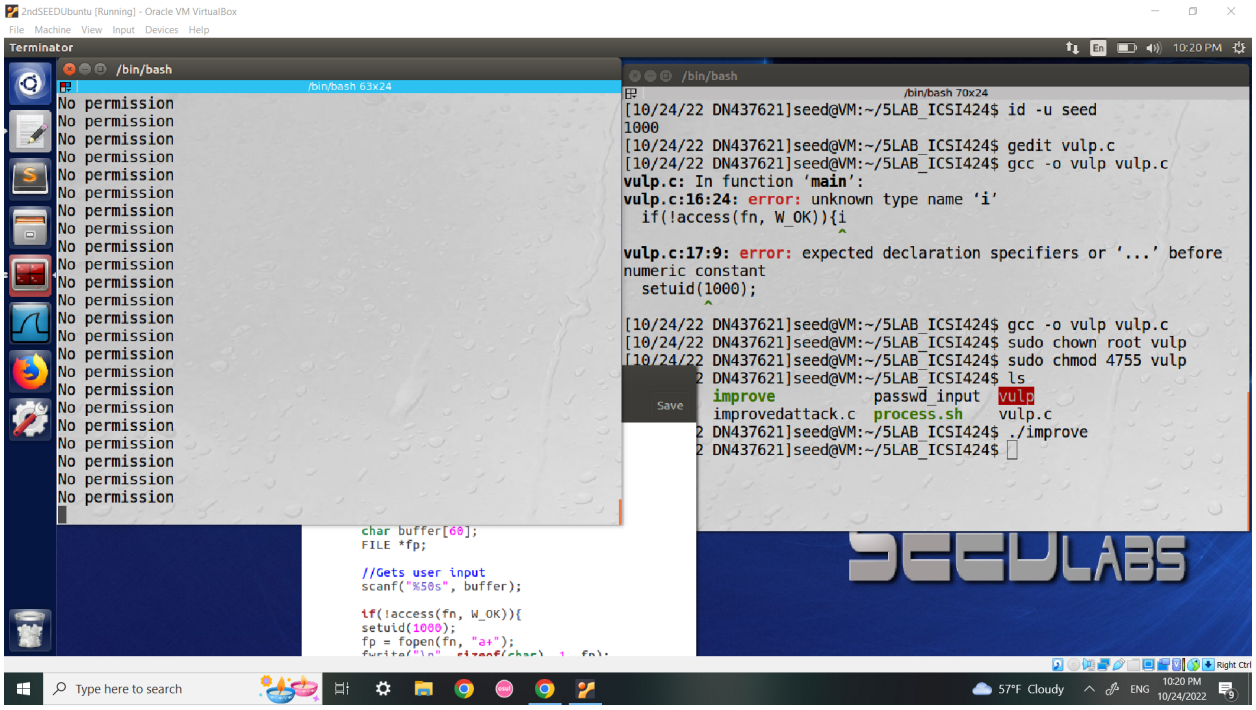
Task 2a: The attack process is run in the background, then the process.sh is run, which will keep running the root process vulp.c over and over again. In vulp.c the symbolic link is changed over and over again in between /dev/null and /etc/passwd. As the pointer changes over and over, no permission is printed until access is given. It will repeat until specific conditions are met, and access is given. We can determine if the attack is successful, by monitoring the time stamp, which is done in process.sh. When the time stamp changes the program will stop, since it signifies that passwd was changed. When doing su test, you can see that no password is required, yet we can gain root access, checked through the id command. When doing tail /etc/passwd we can see that a new entry was created at the end of the file.



Task 2b: By using unlink() and symlink() approach in task 2a, we have a race condition in the program within our actual attacking code. While trying to exploit the race condition in the target program, the target program can make a mistake and exploit the attacking program instead. We are going to use a different system call rename.This system call is introduced to make symlinking atomic. With this improved attack we can make the attack work when the attack is called right away. Having the same passwd file change result, but more consistently.



Task 3: By giving vulp.c a limited amount of privileges with setuid(1000), the attack never works. By having the setuid set to seed, even if the system is tricked into opening the password file, setuid(1000) states that the user has only seed access.



Task 4:
With protected_symlinks set to being on (this is why it is set equal to 1), the symlinks that are being exploited in the attacks are now safe. Because they are safe, similar to the last task, the attack does not work. Even with the improved more direct attack. Race condition takes advantage when a program insecurely creates files, so an attack system can create a symbolic link to that vulnerable file. protected_symlinks is Ubuntu's sticky link protection. When set to "1" the protected_symlinks, makes it so that symlinks are only permitted to be followed when the uid of the symlink and the directory owner matches the symlink's owner. The limitation of this fix, is that it doesn't solve the fundamental problems, the race condition problem. Allows for the user to make unsafe code, and to stay safe from what they would have been vulnerable to.