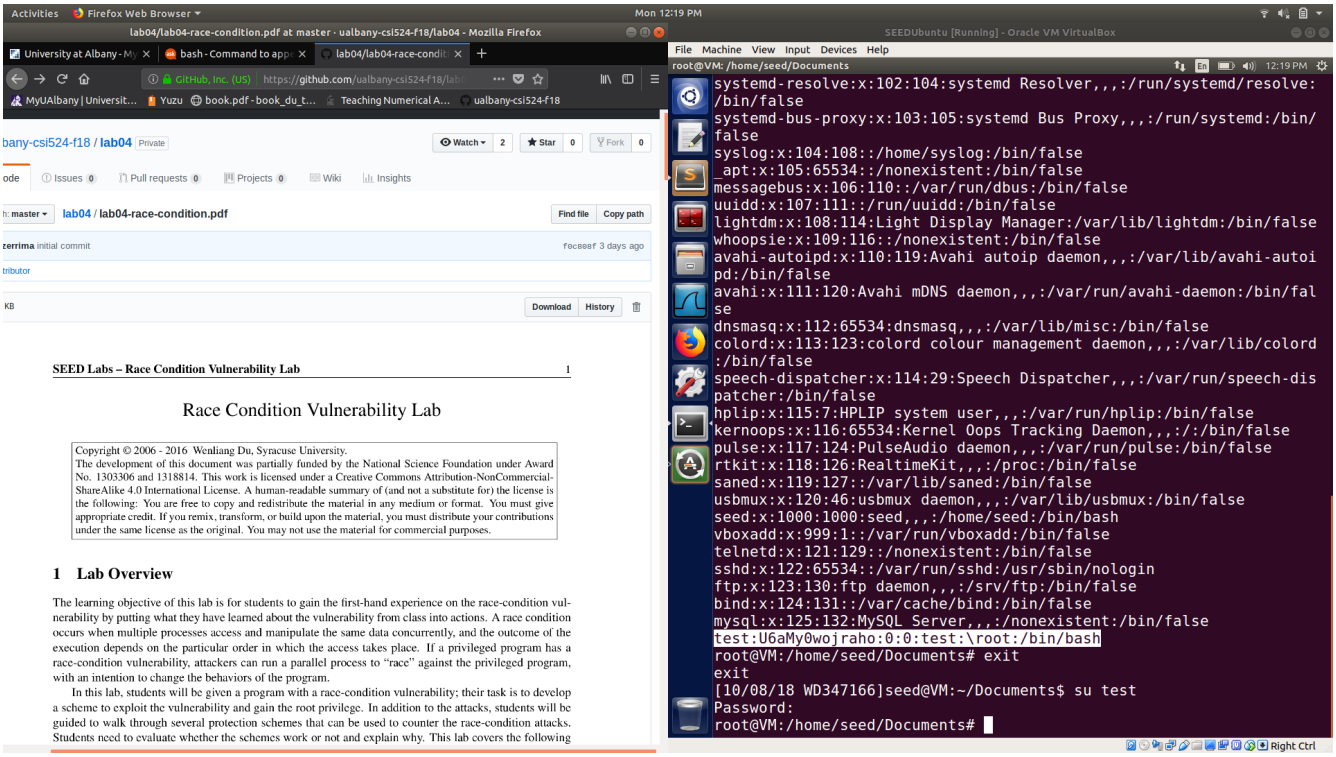
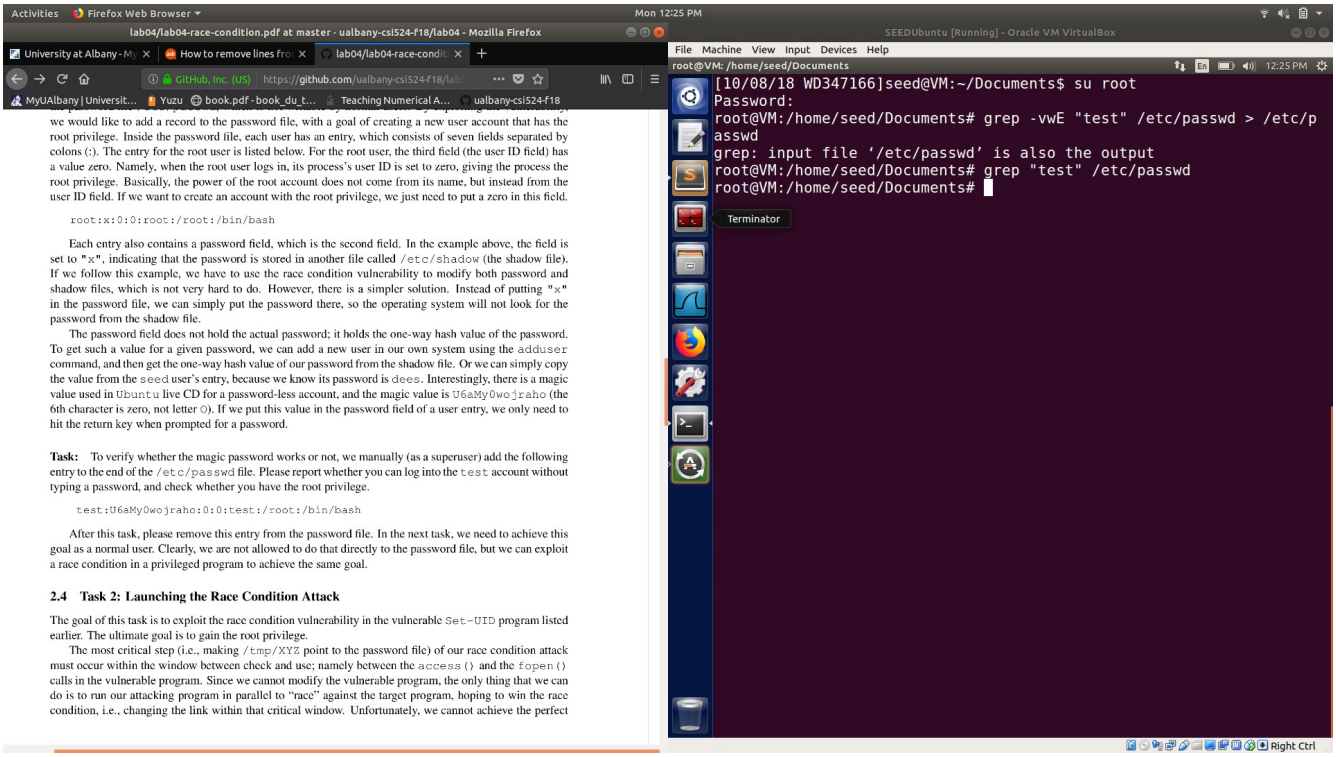
William Dahl
ICSI 424 Information Security
Lab 4
October 8th, 2018

Task 1: In this screen shot I show that the test user had been added to the etc/passwd/ file and the I was able to enter a root shell by switching to that user.



In this screen shot I remove the test user for the etc/passwd/ file using grep -v and then show that there is nothing in the etc/passwd/ file containing test



we would like to add a record to the password file, with a goal of creating a new user account that has the root privilege. Inside the password file, each user has an entry, which consists of seven fields separated by colons (:). The entry for the root user is listed below. For the root user, the third field (the user ID field) has a value zero. Namely, when the root user logs in, its process's user ID is set to zero, giving the process the root privilege. Basically, the power of the root account does not come from its name, but instead from the user ID field. If we want to create an account with the root privilege, we just need to put a zero in this field.

```
root:x:0:0:root:/root:/bin/bash
```

Each entry also contains a password field, which is the second field. In the example above, the field is set to "x", indicating that the password is stored in another file called /etc/shadow (the shadow file). If we follow this example, we have to use the race condition vulnerability to modify both password and shadow files, which is not very hard to do. However, there is a simpler solution. Instead of putting "x" in the password file, we can simply put the password there, so the operating system will not look for the password from the shadow file.

The password field does not hold the actual password; it holds the one-way hash value of the password. To get such a value for a given password, we can add a new user in our own system using the adduser command, and then get the one-way hash value of our password from the shadow file. Or we can simply copy the value from the seed user's entry, because we know its password is dees. Interestingly, there is a magic value used in Ubuntu live CD for a password-less account, and the magic value is U6aMy0wojraho (the 6th character is zero, not letter O). If we put this value in the password field of a user entry, we only need to hit the return key when prompted for a password.

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.

```
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
```

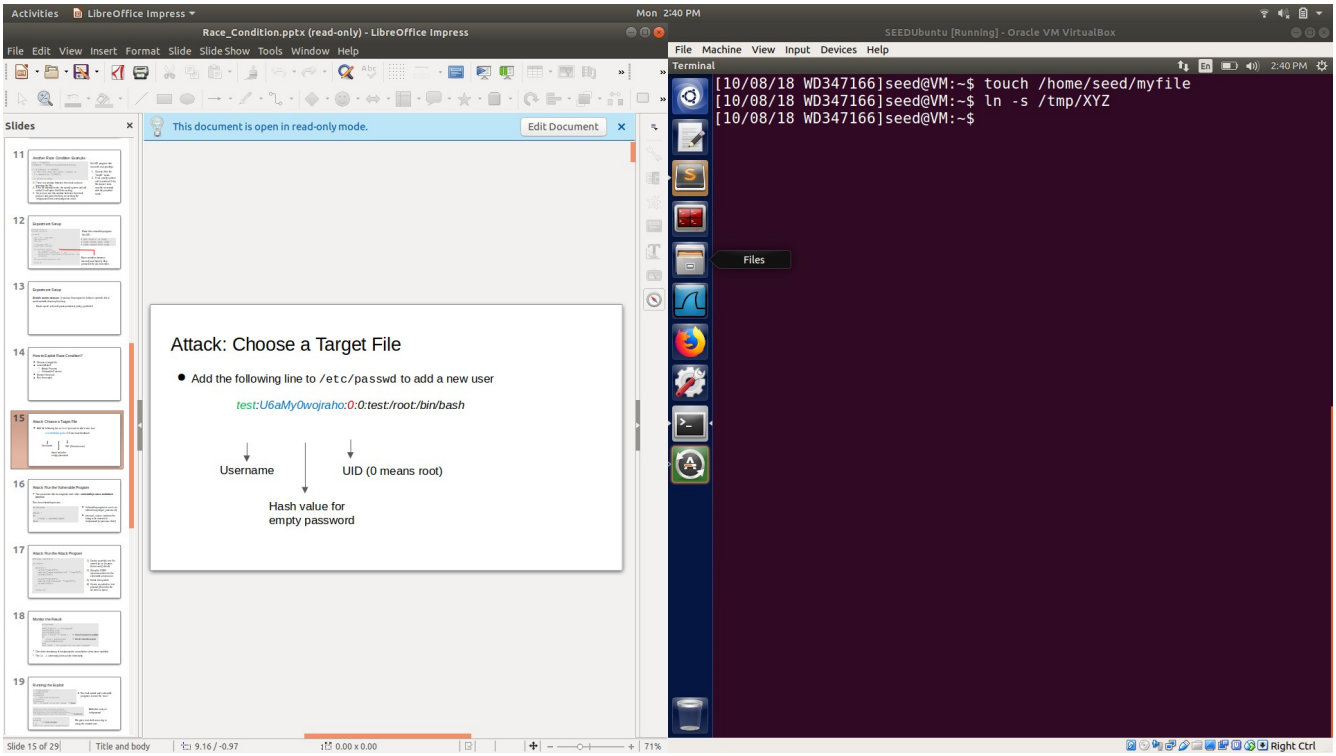
After this task, please remove this entry from the password file. In the next task, we need to achieve this goal as a normal user. Clearly, we are not allowed to do that directly to the password file, but we can exploit a race condition in a privileged program to achieve the same goal.

2.4 Task 2: Launching the Race Condition Attack

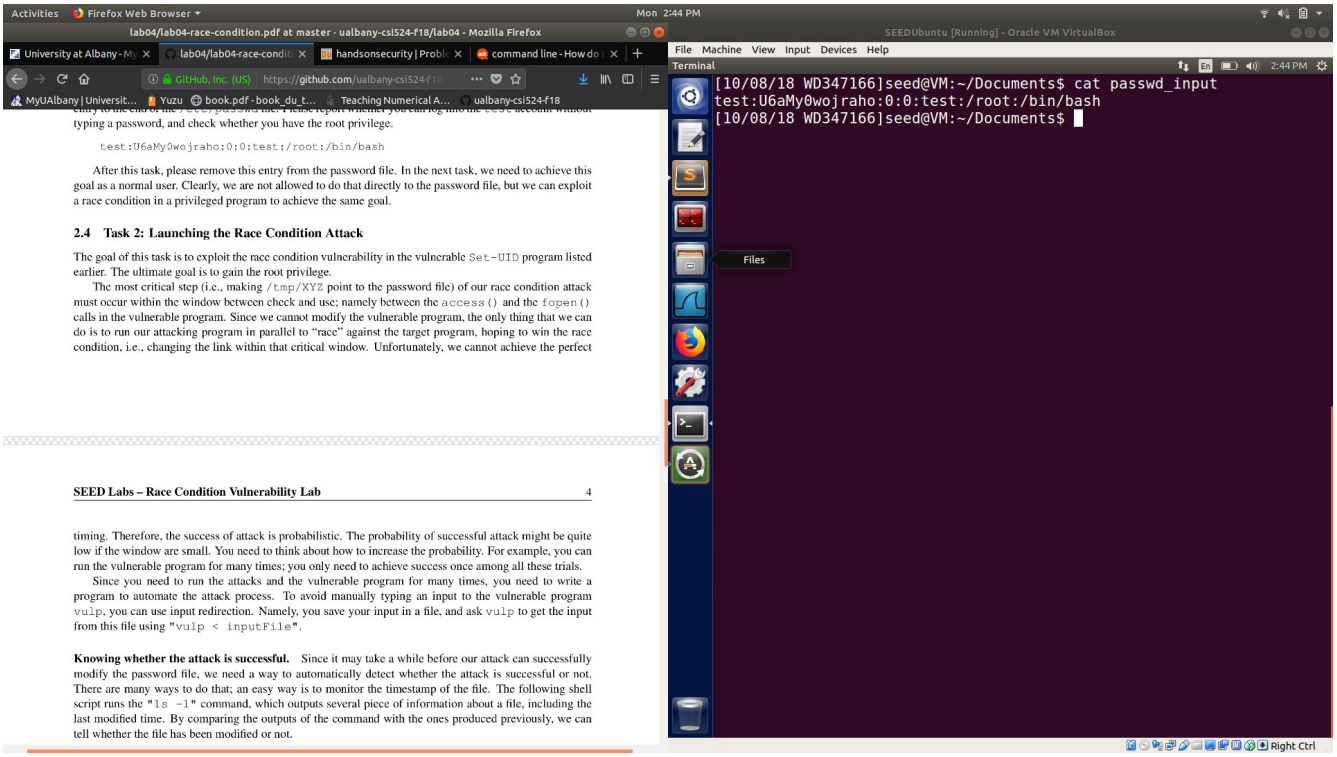
The goal of this task is to exploit the race condition vulnerability in the vulnerable Set-UID program listed earlier. The ultimate goal is to gain the root privilege.

The most critical step (i.e., making /tmp/XYZ point to the password file) of our race condition attack must occur within the window between check and use; namely between the access() and the fopen() calls in the vulnerable program. Since we cannot modify the vulnerable program, the only thing that we can do is to run our attacking program in parallel to "race" against the target program, hoping to win the race condition, i.e., changing the link within that critical window. Unfortunately, we cannot achieve the perfect

Task 2: In this screen shot I created a file called home/seed/myfile and then created a symbolic link called tmp/XYZ



This screen shot shows the contents of the file passwd_input



typing a password, and check whether you have the root privilege.

```
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
```

After this task, please remove this entry from the password file. In the next task, we need to achieve this goal as a normal user. Clearly, we are not allowed to do that directly to the password file, but we can exploit a race condition in a privileged program to achieve the same goal.

2.4 Task 2: Launching the Race Condition Attack

The goal of this task is to exploit the race condition vulnerability in the vulnerable `Set-UID` program listed earlier. The ultimate goal is to gain the root privilege.

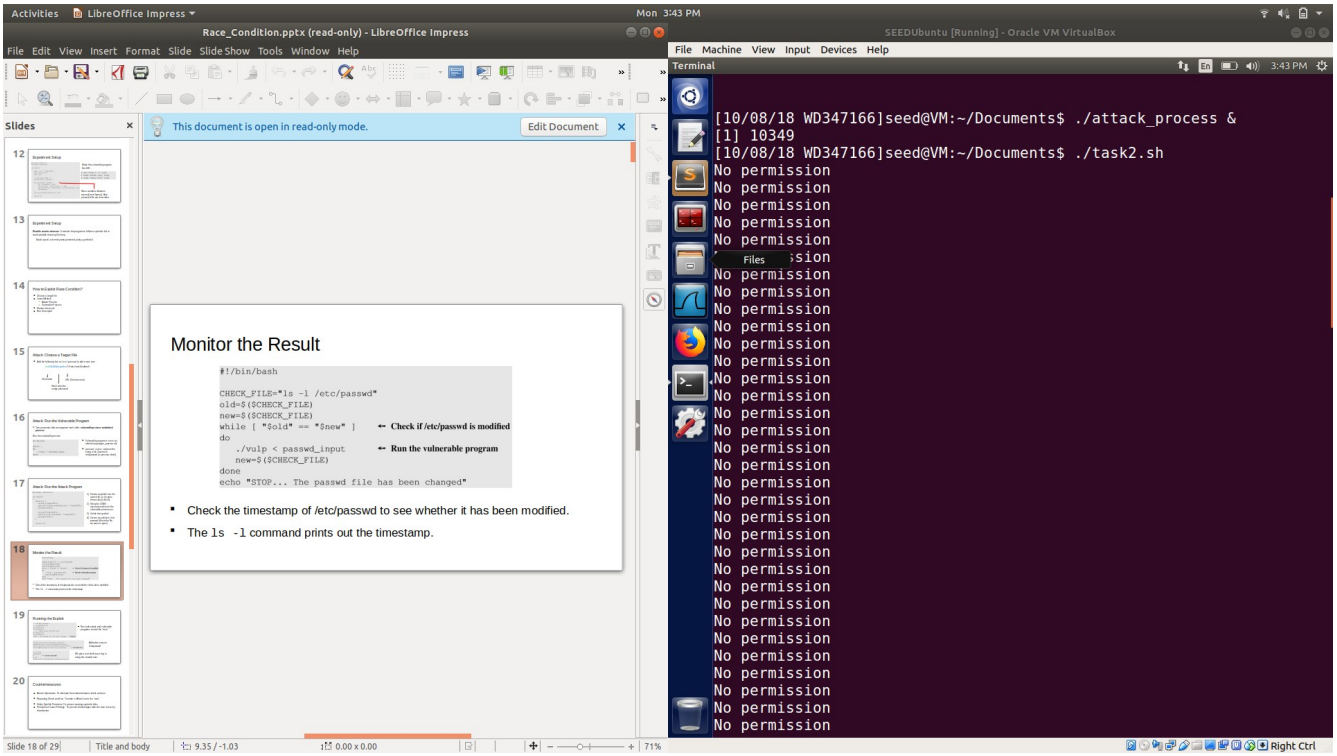
The most critical step (i.e., making `/tmp/XYZ` point to the password file) of our race condition attack must occur within the window between check and use; namely between the `access()` and the `fopen()` calls in the vulnerable program. Since we cannot modify the vulnerable program, the only thing that we can do is to run our attacking program in parallel to "race" against the target program, hoping to win the race condition, i.e., changing the link within that critical window. Unfortunately, we cannot achieve the perfect

timing. Therefore, the success of attack is probabilistic. The probability of successful attack might be quite low if the window are small. You need to think about how to increase the probability. For example, you can run the vulnerable program for many times; you only need to achieve success once among all these trials.

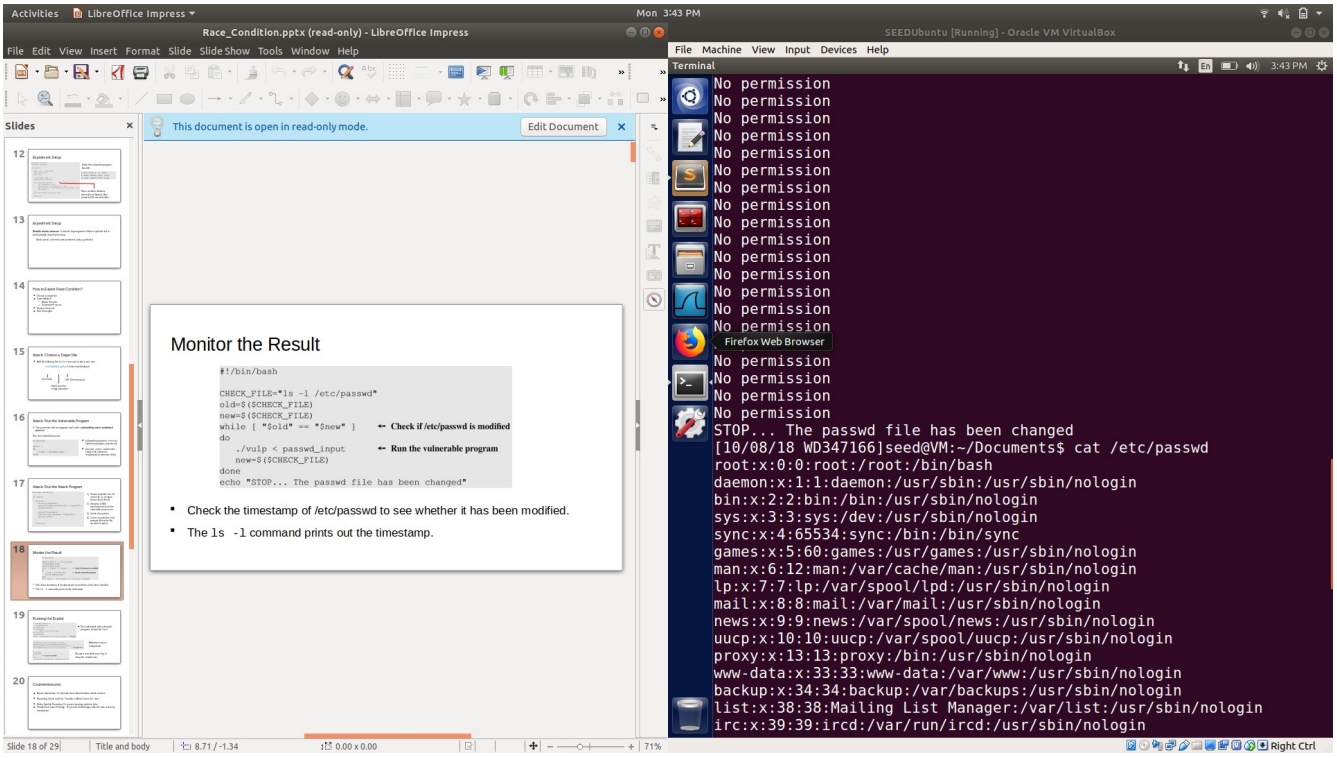
Since you need to run the attacks and the vulnerable program for many times, you need to write a program to automate the attack process. To avoid manually typing an input to the vulnerable program `vulp`, you can use input redirection. Namely, you save your input in a file, and ask `vulp` to get the input from this file using "`vulp < inputFile`".

Knowing whether the attack is successful. Since it may take a while before our attack can successfully modify the password file, we need a way to automatically detect whether the attack is successful or not. There are many ways to do that; an easy way is to monitor the timestamp of the file. The following shell script runs the "`ls -l`" command, which outputs several piece of information about a file, including the last modified time. By comparing the outputs of the command with the ones produced previously, we can tell whether the file has been modified or not.

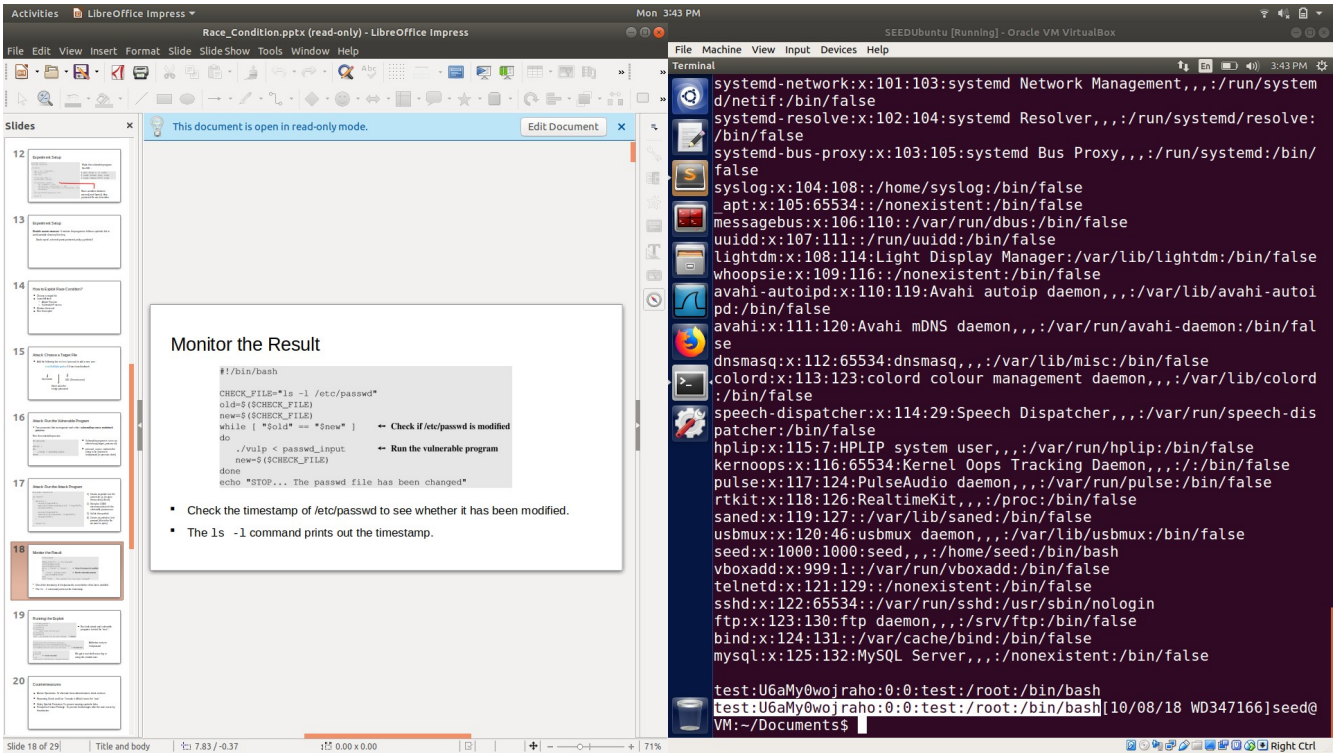
In this screen shot I run the attack process and the vulnerable process in a loop



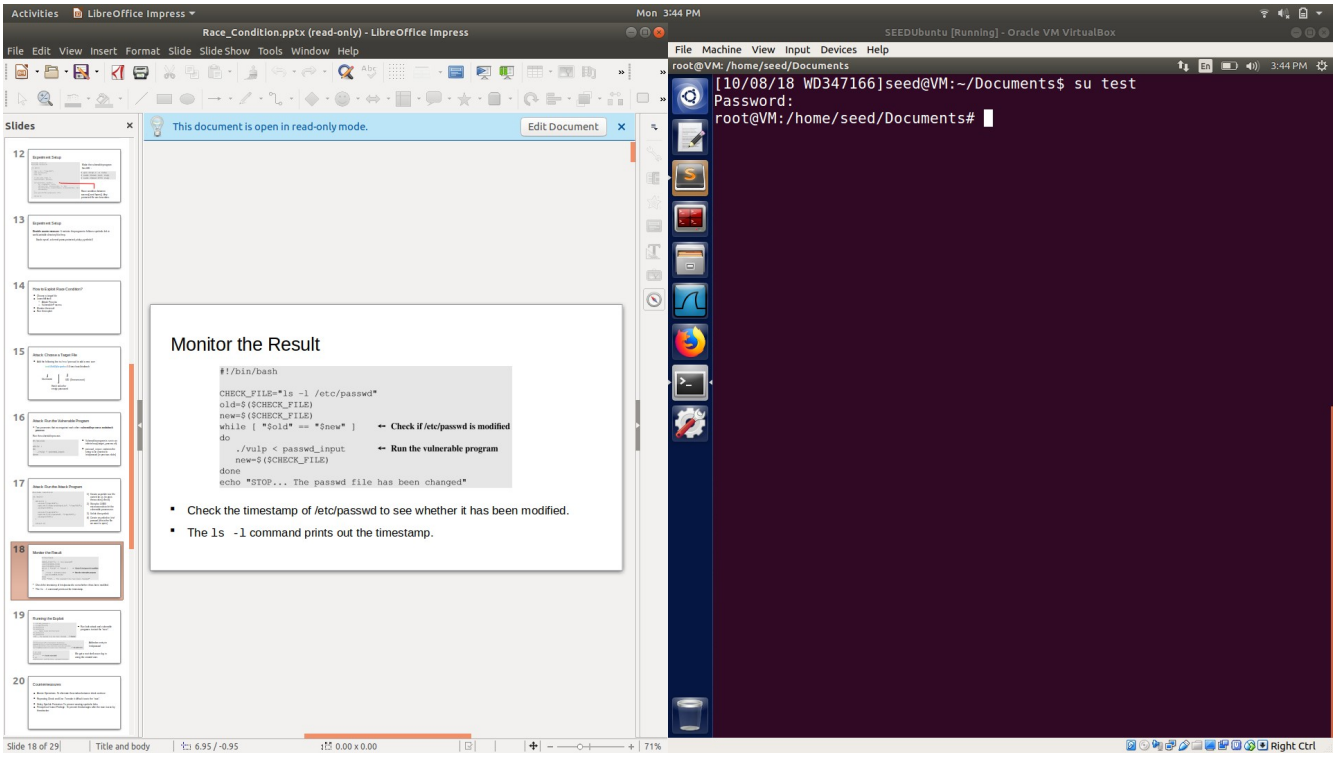
In this screen shot I get the message that the passwd file was changed



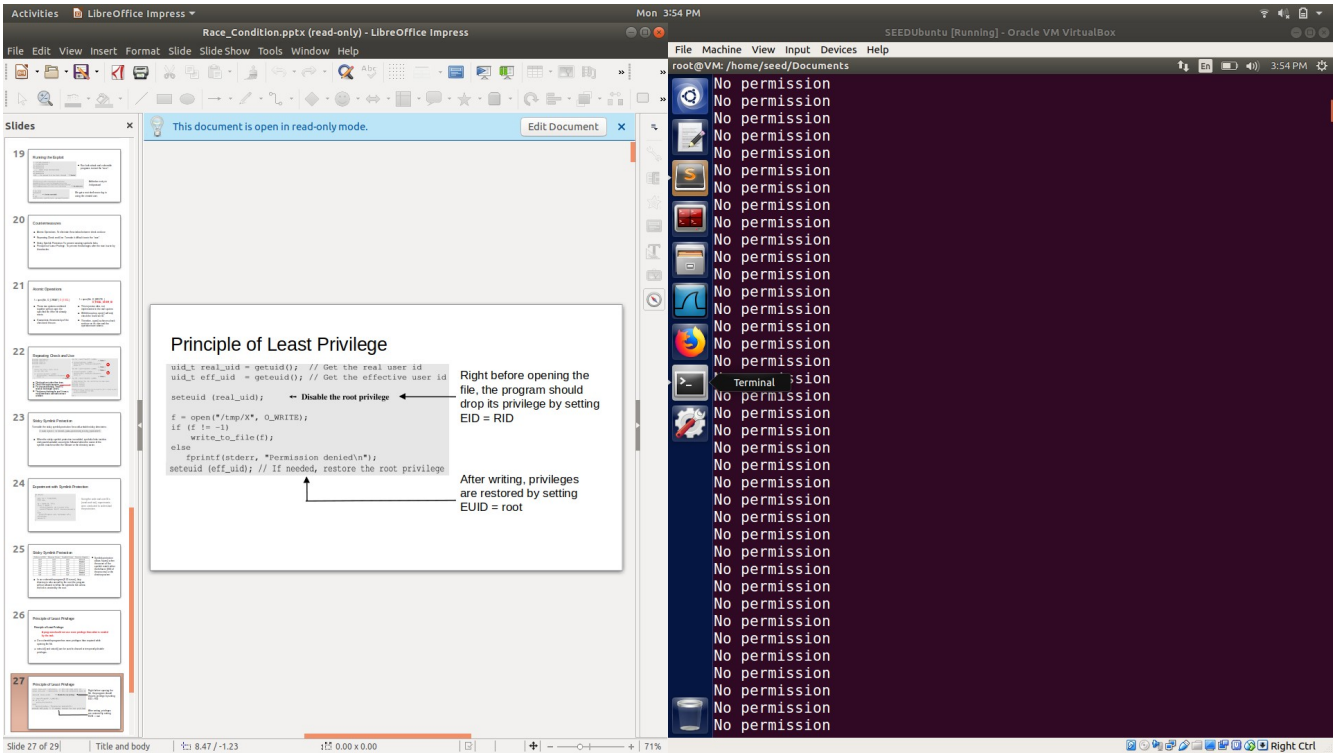
In this screen shot I show that test is entered in the passwd file



In this screen shot I show that I can get a root shell by switching users to test

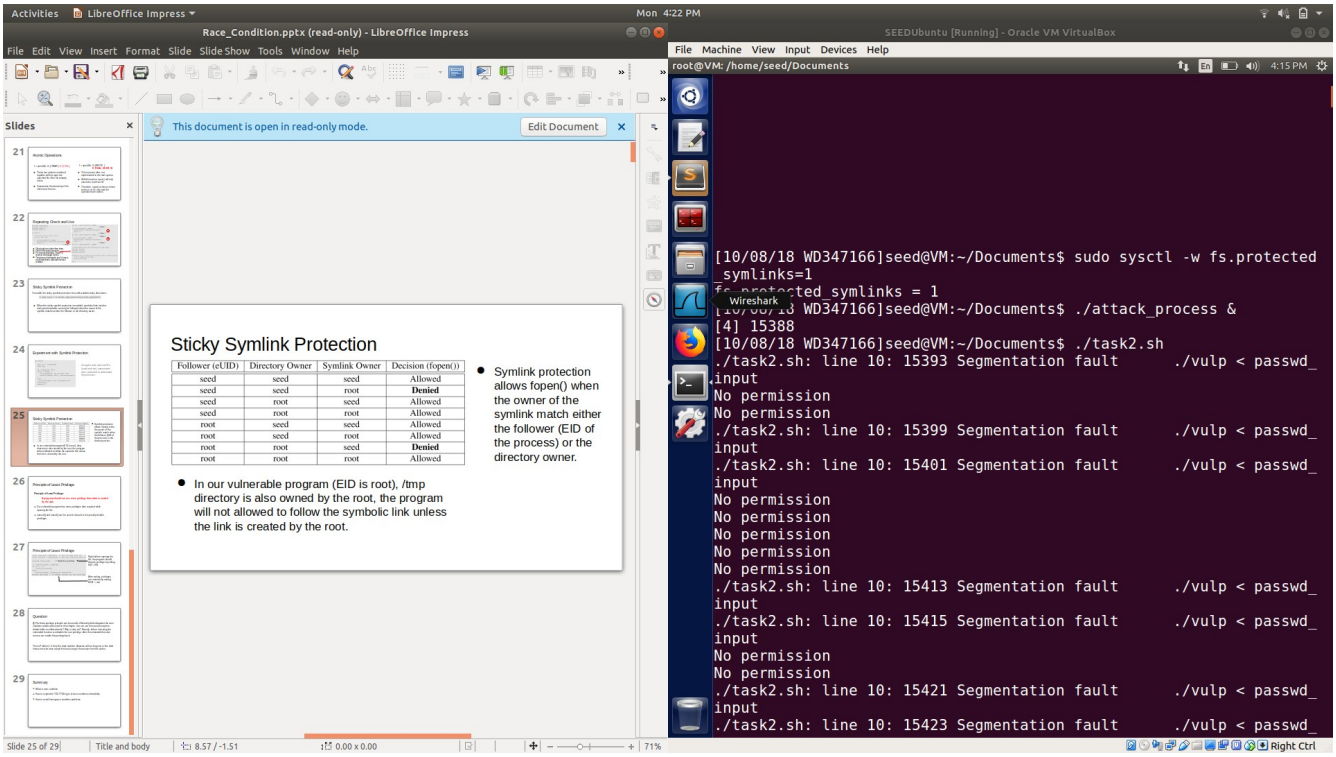


Task 3: In this screen shot I ran the attack again but this time implementing the principle of least privilege in the vulnerable program and the attack did not succeed



The attack was not able to succeed because right before the access was checked the effective uid was changed to the real uid which was not the root thus the passwd file could not be written too because the person who was running the program did not have write permissions

Task 4: In this screen shot I turned on the protected symlinks and ran the attack again



The attack did not work again, this is because when the symlink protection is on it will deny access when it uses a symlink whose owner does not match the EID or the owner of the directory. In this case neither the EID or the owner of the etc/ directory which was both root.