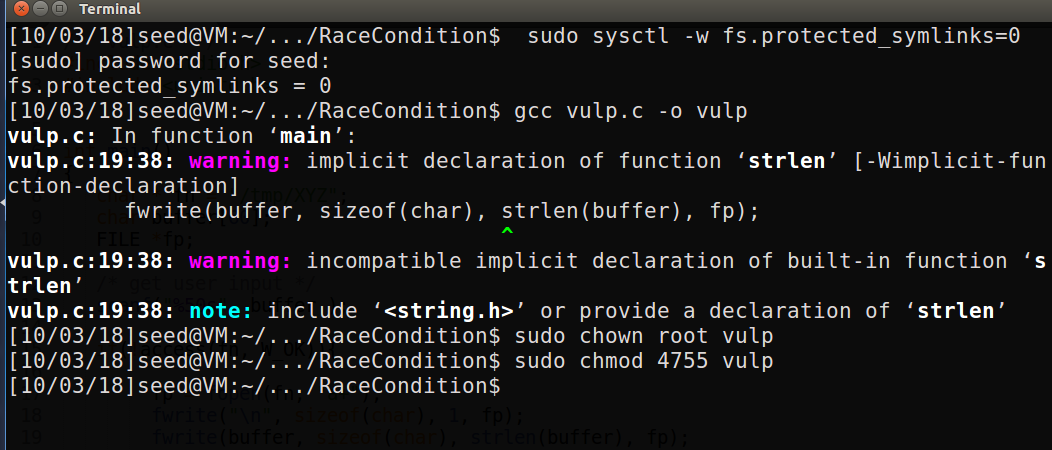
**Race Condition vulnerability lab**

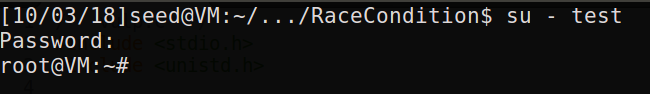
**Karan Amrutesh**



**Task 1: Choosing Our Target**

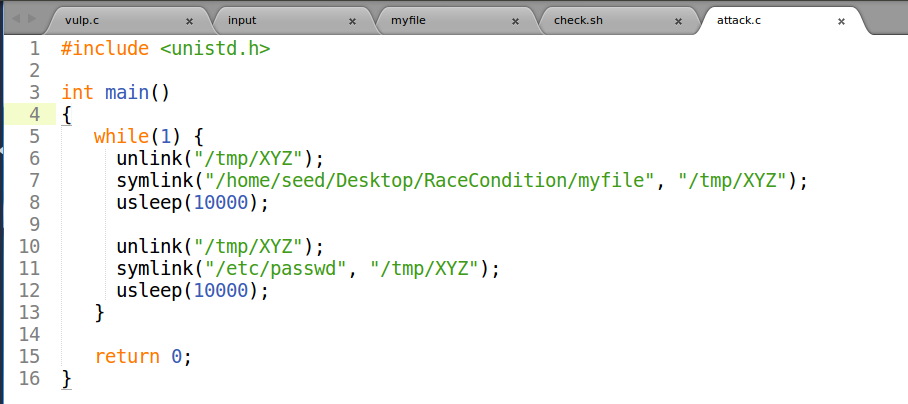
* After adding the following line in the /etc/passwd file we can login into the test user by just pressing enter:

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



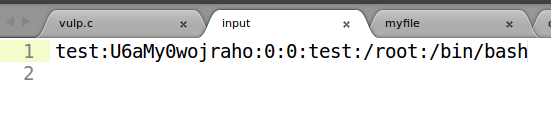
**Task 2: Launching the Race Condition Attack**

* We have the attack.c file as shown.
* We first make /tmp/XYZ point to one of our own files (myfile) so that we can pass through the *access()* check.
* Then we make the /tmp/XYZ to point to our target file /etc/passwd. We do these two steps repeatedly so to race against the target process.



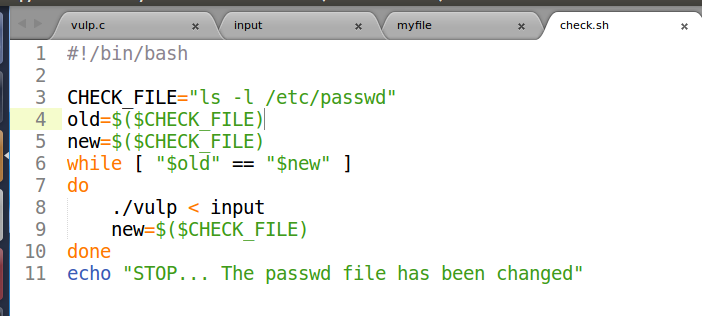
***attack.c***

* In the *input* file, we put the entry for the test user:



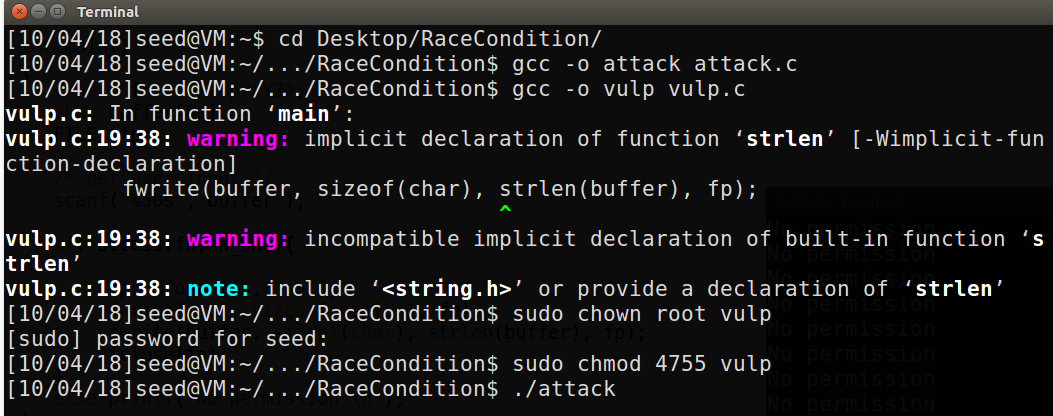
***input file***

* To know if the attack succeeds, we need to check the timestamp of the password file to see whether it has changed or not. We do this continuously by the following shell script which also runs the vulnerable program with the input file containing the entry for the test user.

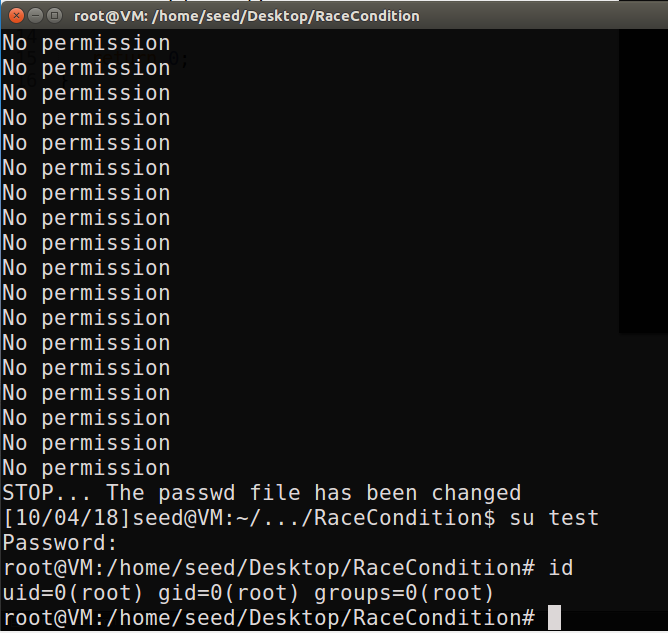


***target.sh***

* We first run the attack program in the background and then we run the shell script

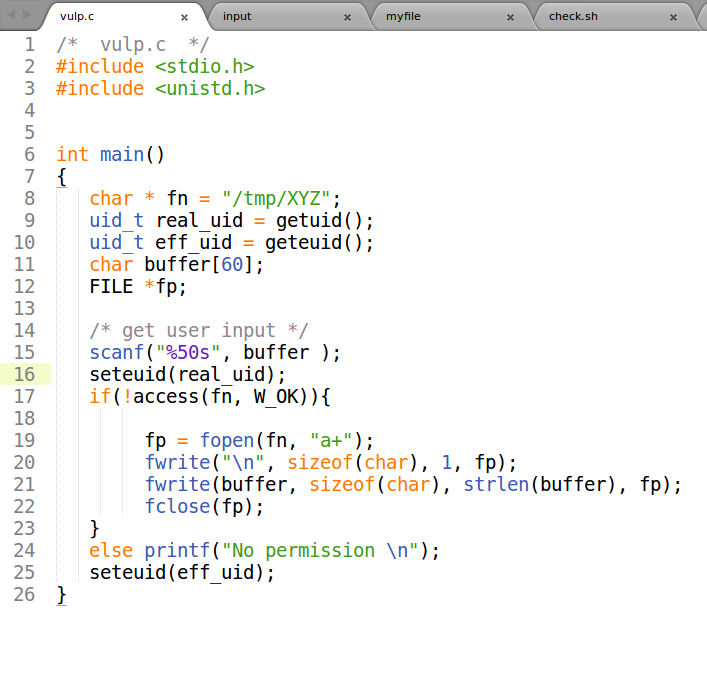


* Initially the program keeps printing “No permission” because of the failure of *access()* check.
* If we succeed in the race and have successfully modified /etc/passwd file, the target program will terminate.
* So we can log into the test account without typing any password. The output of the id command confirms that we have gained the root privilege.



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

* The principle of least privilege states that a program should not use more privilege than what is needed.

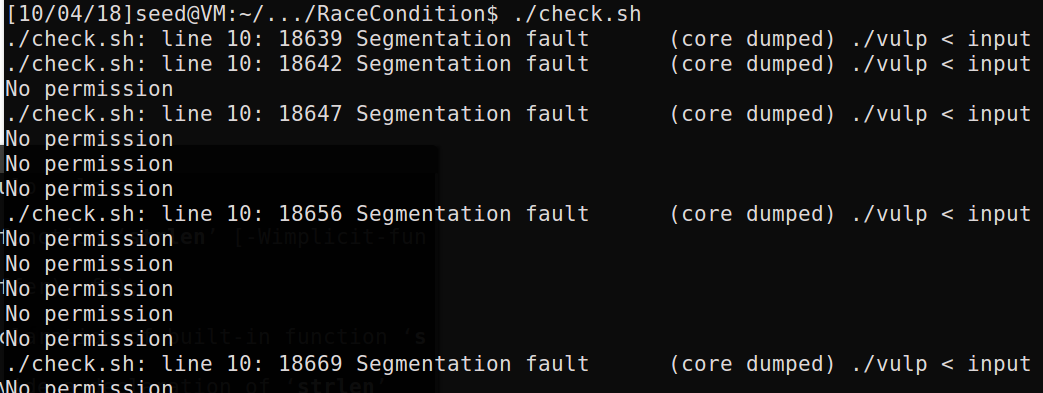


* The above code temporarily sets the effective user ID to the real user ID using the *seteuid(),* thus disabaling the root privelege.
* Due to this, the access rights of the real user, and not root, will be checked.
* So, the program cannot open any other file other than those accessible to the user. Because of this, we get the following output indicating the user does not have permission to write into the file.
* Once the task is completed, we set the effective user ID to its original value using *seteuid().*



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

* We do not succeed in the attack after turning on the protection scheme:



* The protection scheme does not let a program to follow any symbolic link that is not created by the owner of the program i.e. the symbolic links inside a directory can only be followed when the owner of symlink matches either the follower or the directory owner
* In our experiment, since the vulnerable program runs with the root privilege(euid is root) and the file in /tmp directory is also root, the program wont be allowed to follow any symbolic link that is not created by the root.
* Hence the program will crash when it tries to follow the symlink created by the attacker.

The limitation is that this protection is applicable to world – writable sticky directories like /tmp. So the directories like the /home/seed is not sticky!