Mohammad Fakhruddin Babar

ID: M564K763

**Task 1: Linux Access Control Warm-up**

**Q1.1: Search for Linux file permissions online.**

**What permission setting are you specifying with 0644?**

Ans: User will get read and write access, group and others will get only read access. And its not a set- UID program.

**Q1.2:Did the name of the file changed?**

**If so, why where you successful in doing so this time?**

Ans: Yeah, name of the file is changed.

**Q 1.3 Why do you see an operation not permitted error?**

Ans: As this file is owned by the user1, so wushock doesn’t have any access to change the permission.

**Q 1.4 Whose permissions did you change using the above chmod command?**

Ans: By using above command, both the owner lost its write privilege and others got write privilege, but it doesn’t have read privilege.

**Q 1.5 Why does running cat on the file produce a permission denied error, but the echo/append**

**command does not?**

Ans: We have removed wushock from the group and it became others. And others have only written access, not the read access. That’s why we couldn’t read the file. but able to write into the file.

**Q 1.6 Are you still able to change the name but not the mode (permissions) of the file?**

Ans: No, this time I couldn’t change the file name too.

**Q 1.7 What do you see displayed?**

Ans:

Play Angry

Go Shockers

**Q 1.8 Why do you see a permission denied error?**

Ans: Group has only the read access, so it couldn’t write into a file.

**Q1.9: What umask setting is returned and how do you calculate the default permissions of a new file?**

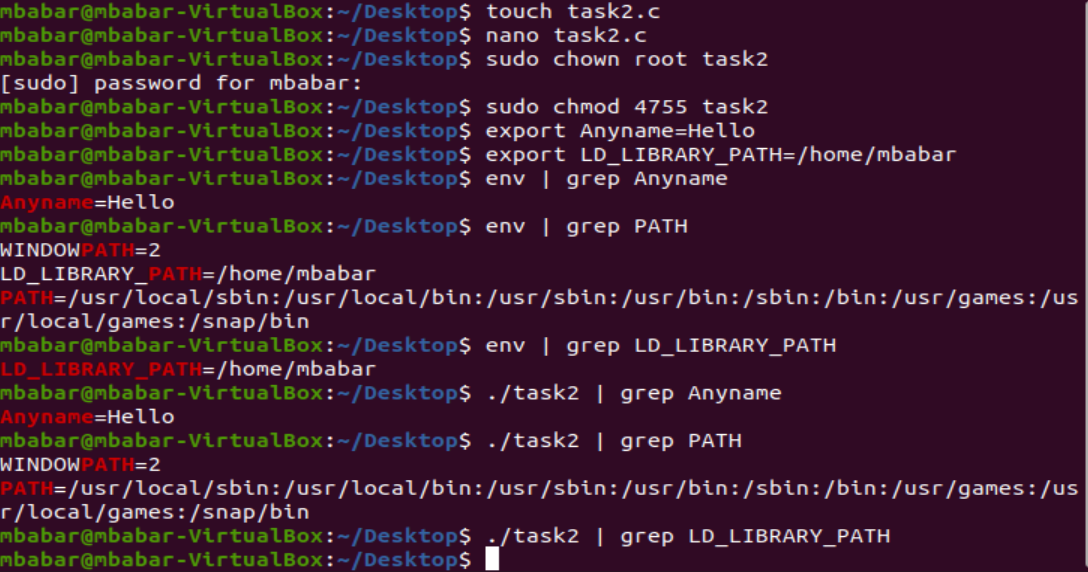
Ans: Unmask value is 002. By subtracting this from 777, we get 775. So, user has rwx, group has rwx and others have r-x permission

**Q 1.10 What do each of these umask settings mean for users attempting to access any new files or directories you create?**

Ans: Those different umask are resetting the file permission.

**Task 2: Environment Variable and Set-UID Programs**

Ans:2.1:

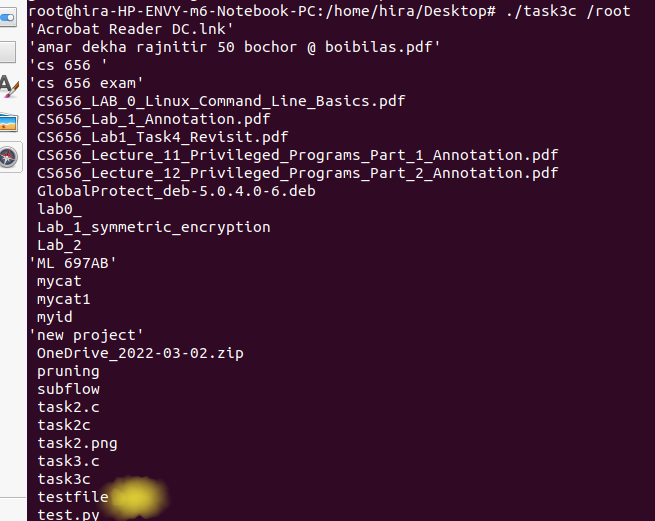


Ans:2.2:

All variables must pass from the parent to the child when it comes to passing environment variables (e.g., PATH, ANYNAME). However, we can see that the LD LIBRARY PATH was not passed to the child. This is because the dynamic linker on most operating systems removes variables that control dynamic linking from the environment of setuid executables for security reasons.

**Task 3: The PATH Environment Variable and Set-UID Programs**

Ans:3.1: “tesfile’’ has been created.

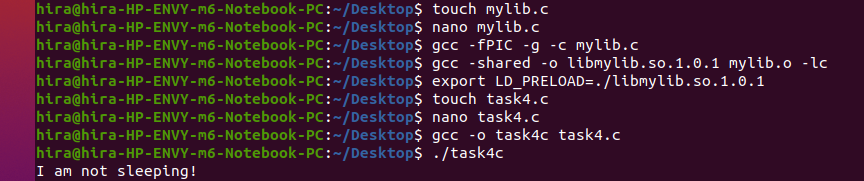


Ans:3.2:

Task3 will run with root privileges, the task3 is owned by the root and it’s a Set-UID program, so it will run with the root privilege

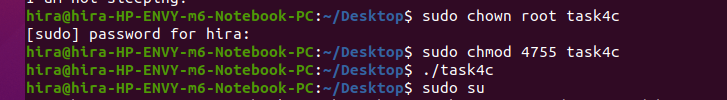
**Task 4: The LD PRELOAD Environment Variable and Set-UID Programs**

Ans:4.1:



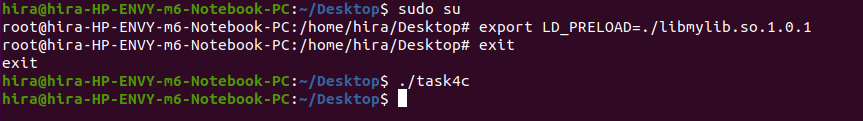
Task4 used the malicious library and printed "I'm not sleeping." As we have exported the environment variable LD\_PRELOAD, the sleep function has been linked to that function instead of the libc standard sleep function.

Ans:4.2:



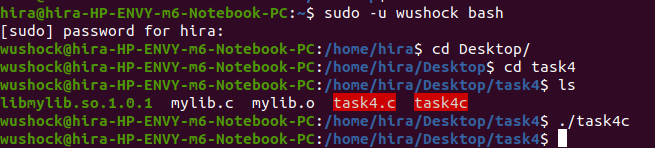
Because of security concerns, the program uses the standard libc library. If the program is Set-UID and the effective user ID differs from the id of the owner process, then the LD PRELOAD environment variable is ignored.

Ans:4.3:



This time task4 uses the standard libc library, the same as the previous answer. If the effective user ID is different than the process’s real ID, then the LD\_PRELOAD won’t be taken into consideration in the complication if the program is Set-UID. Here we can see that this operation has no relation to whether we export the environment variable as root or normal user.

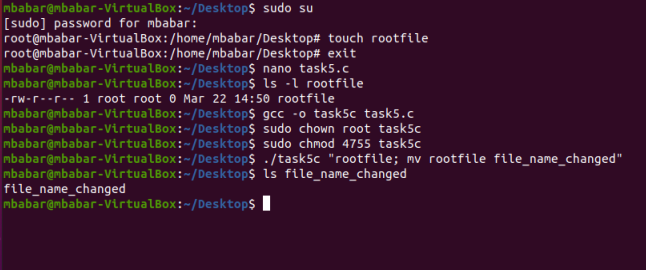
Ans:4.4:



We made Task4 a Set-UID program owned by a non-root user different from the regular user, say wushock. Export the LD PRELOAD environment variable again. However, this time in the wushock account. Execute task 4 as wushock. Task 4 employs the standard libc library, producing the same result as the previous task.

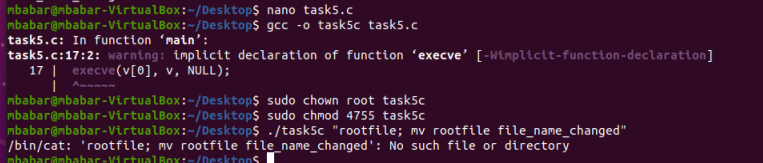
**Task 5: Invoking External Programs Using system() versus execve()**

**Ans:5.1:**



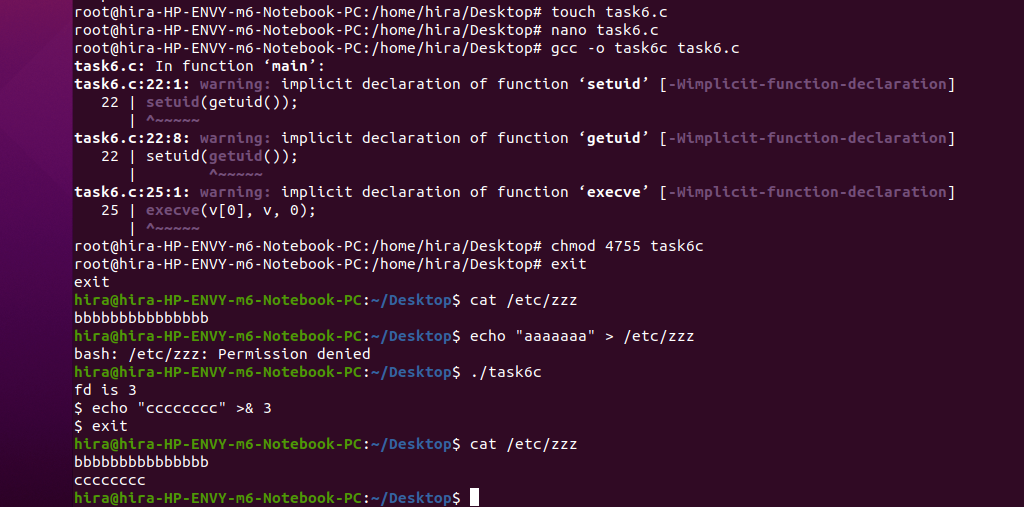
We create a file named “rootfile”, we made the root as the. We attempted to modify with normal user we couldn’t, but after exploiting the prog we could modify the file name with file\_name\_chnaged.

**Ans:5.2: commenting system() command and uncomment execv() command.**



We cannot modify the file. At first we used the system command, we could modify the file, because in fact the system command will open at first a shell (/bin/sh) after that it passes the command to it, and in the shell we can add pass and execute more than a command, we just need to seperate with a semicolon. In the other hand the execve command will directly execute the command and doesn’t invoke the shell.

**Task 6: Capability Leaking:**



The file /etc/zzz will be modified, the reason is that the file was opened with a root privilege, which has the capability of modifying the file, so even after downgrading the privileges, the process still remains with that capability and it passes to the child, consequently, the child can modify the file. To overcome this problem, we need to close the file before downgrading the privileges to clean up all the gained privileges.