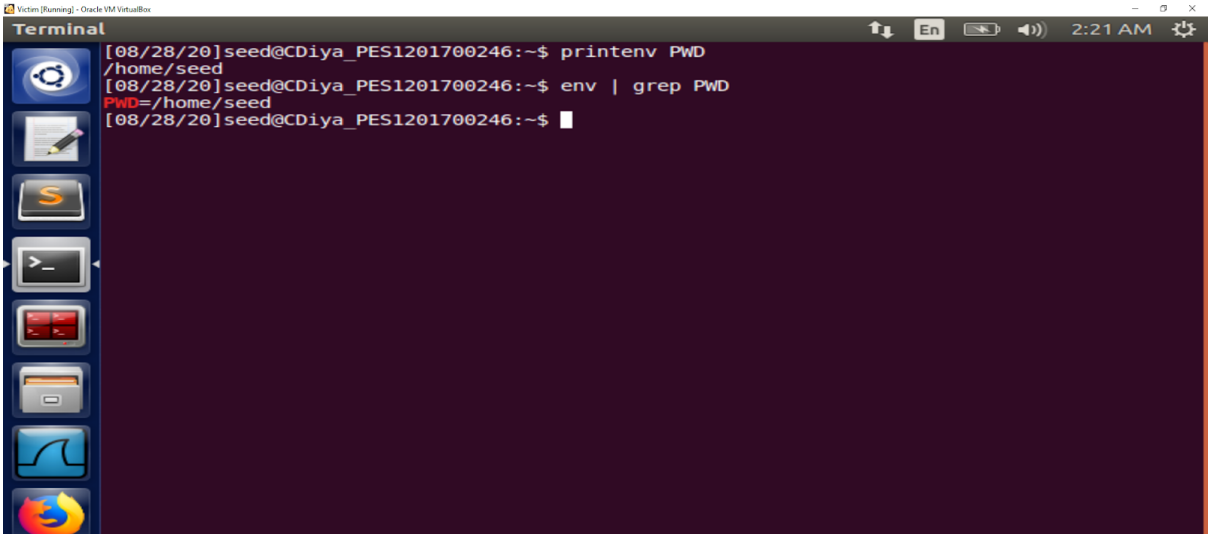
Information Security
LABORATORY 1

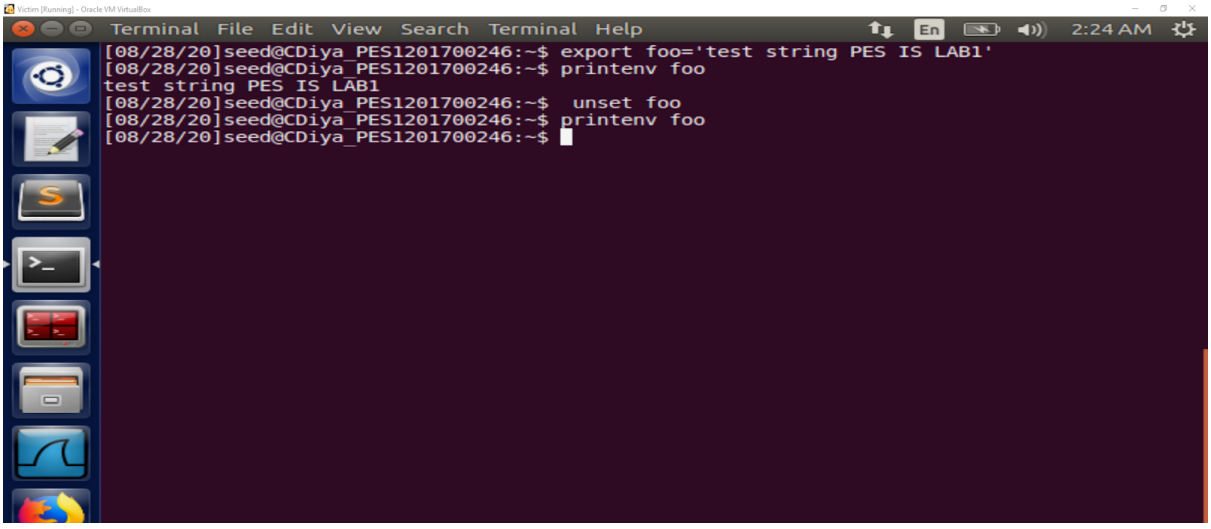
Environment Variable and Set-UID Program Lab

C Diya
PES1201700246

TASK 1: Manipulating environment variables

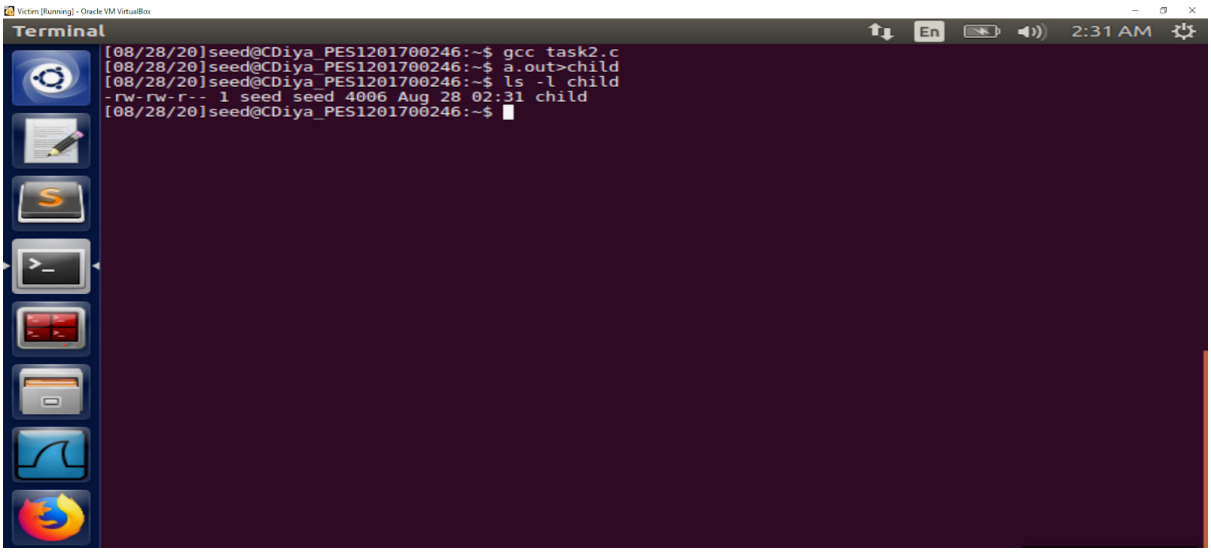


Observation: The printenv or the env command is used to print the environment variables. It can be clearly seen from the screenshot above that when the printenv PWD command is executed, the value of PWD is printed.PWD is an environment variable which stores the path of the current directory. The env command can be used in a similar manner by using grep to find the environment variable.

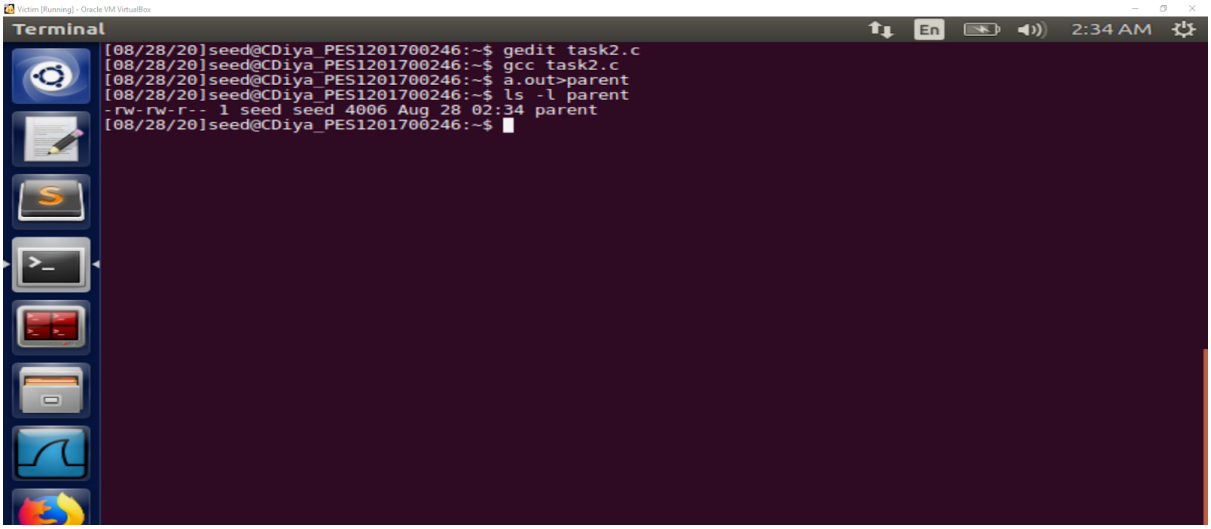


Observation: The screenshot below shows the setting and unsetting of environment variables.The export command can be used for this. A test environment variable foo is set with a variable and printenv verifies that it is now an environment variable. The unset command can be used to unset foo as an environment variable.

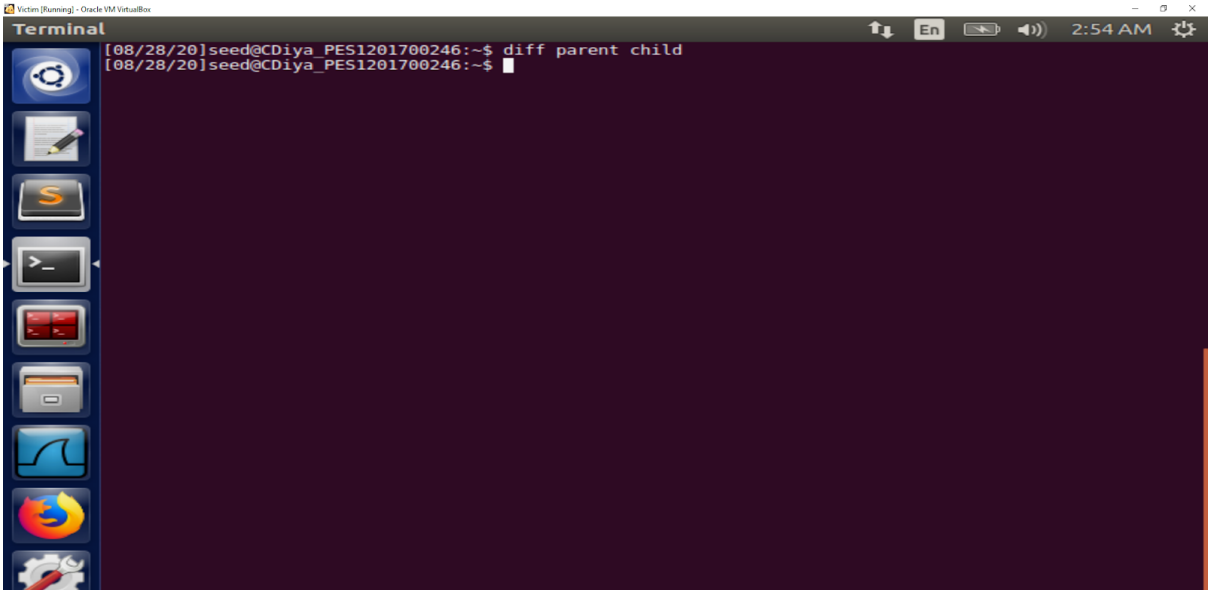
TASK 2: Inheriting environment variables from parents



Observation: the program compiled above uses the fork() unix call to create a child process from a parent process. The screenshot above prints the environment variables of the child process using the printenv command and the output is saved in a file



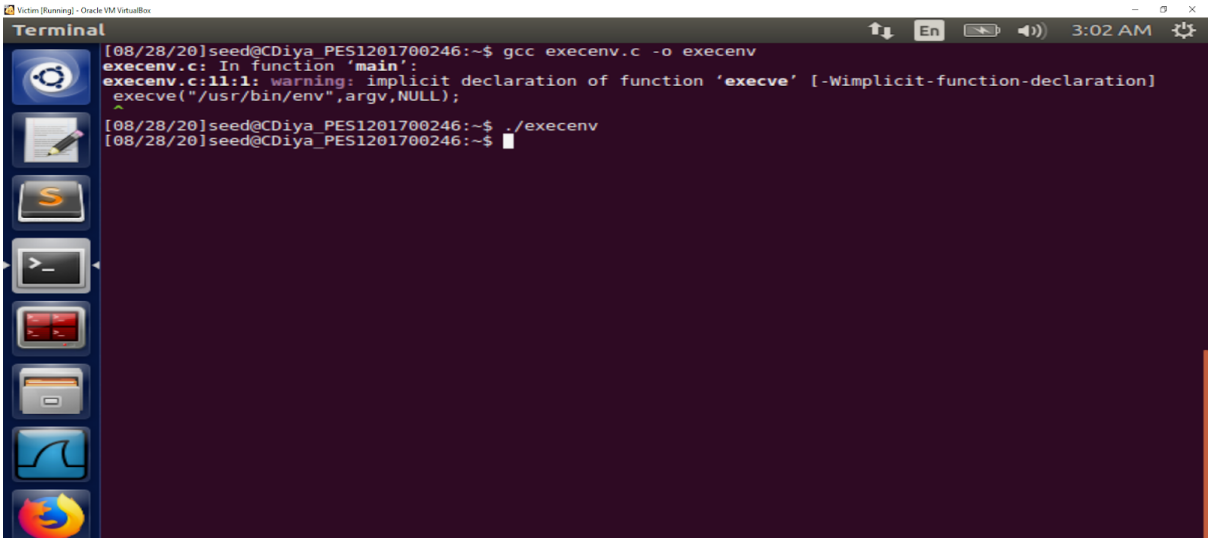
Observation: The screenshot above prints the environment variables of the parent process and the result is saved on an output file.



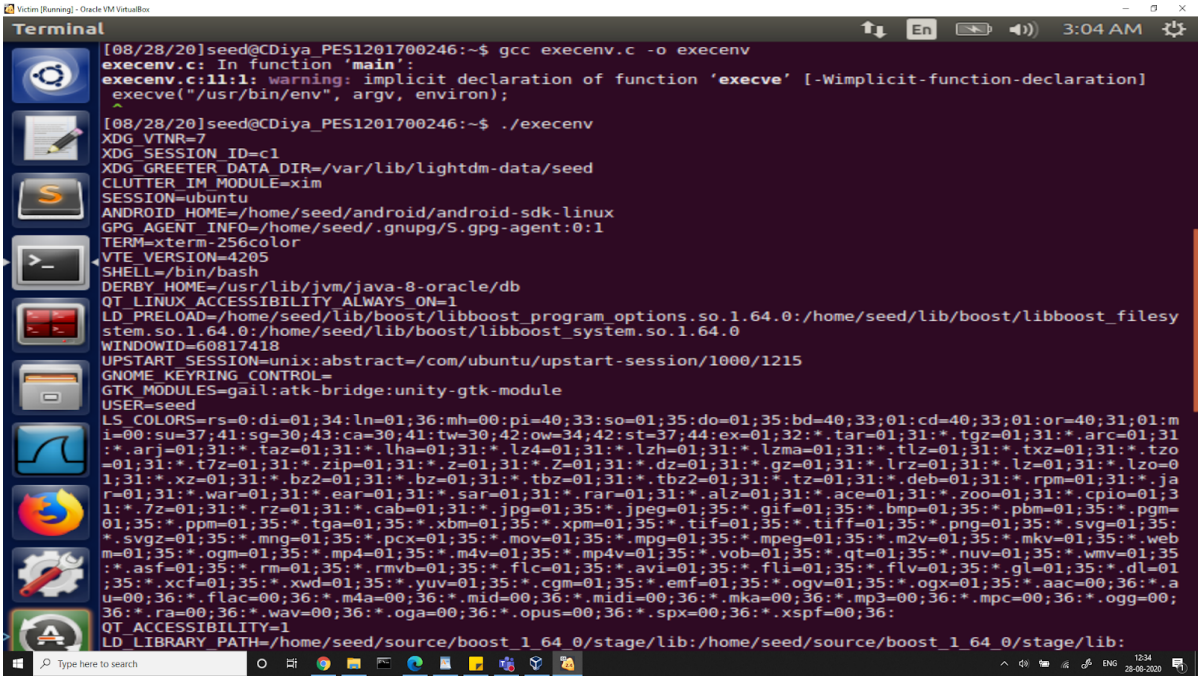
Final Observation:

The diff command is used to compare the output from the parent and child. Thus, it can be observed that nothing gets printed since there are no changes between parent and child. This is because fork() causes child processes to inherit parent's environment variables and no difference can be observed.

TASK 3: Environment variables and execve()

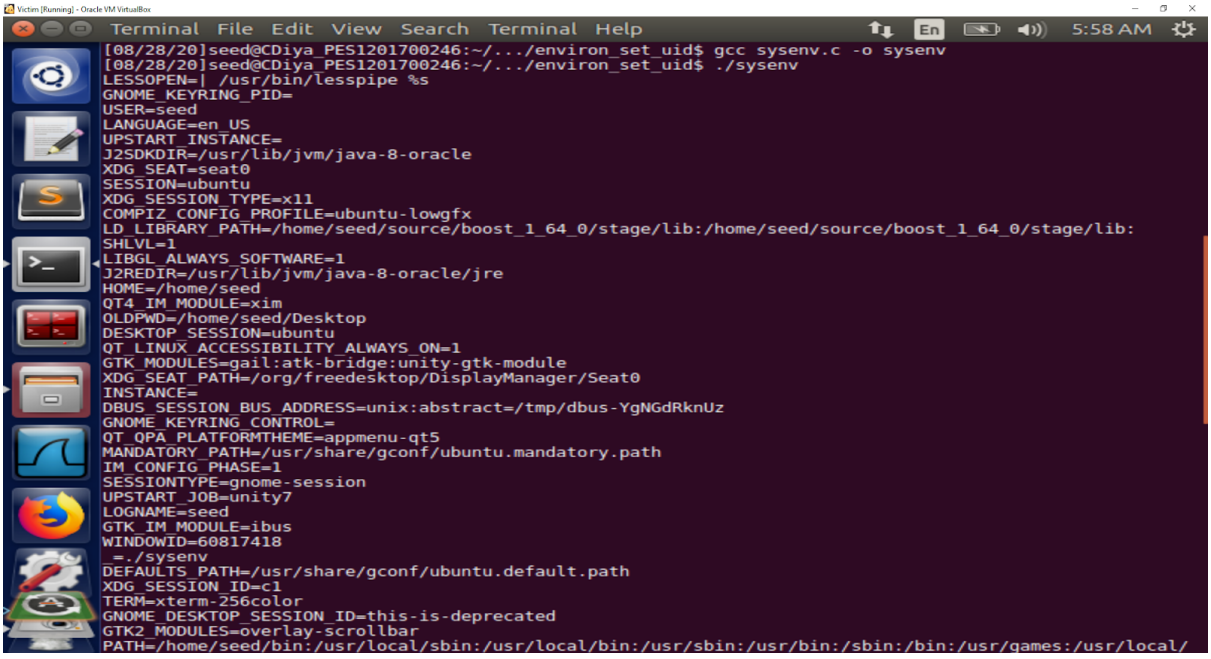


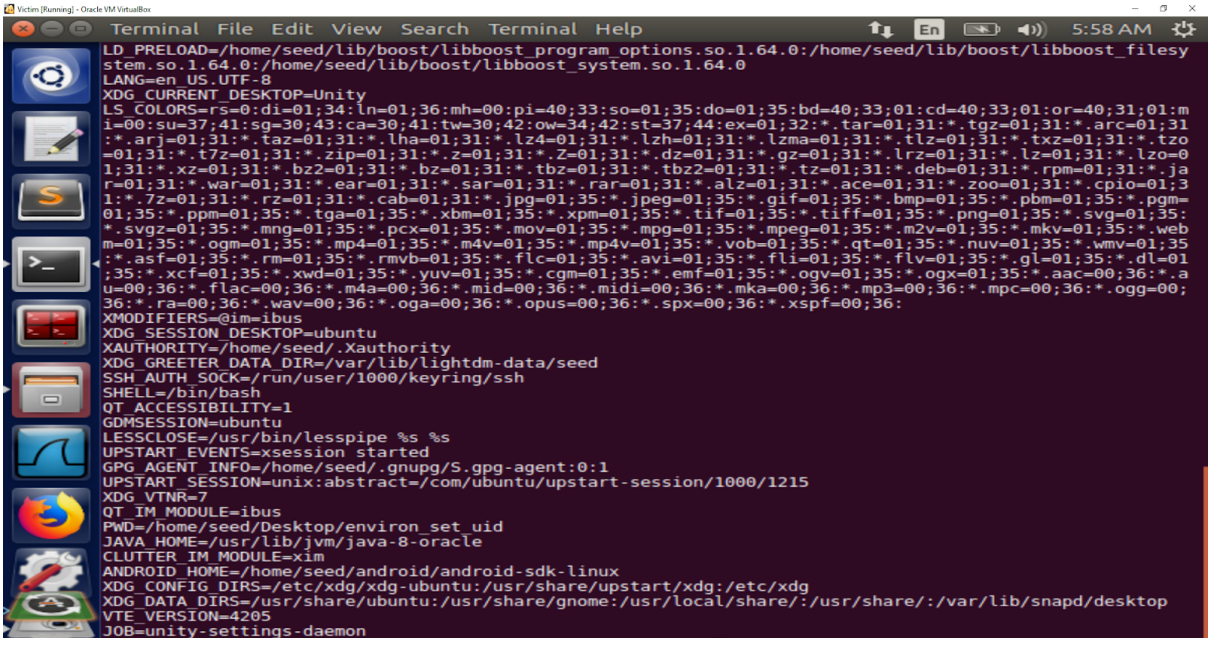
Observation: The screenshot above compiles the program that uses the execve() command. The execve() runs the new program inside the calling process. Since NULL is passed as a parameter to the execve() command, nothing gets printed as no output is generated. The environment variables of the process are not visible.



Observation: On changing the program above to pass environ as a parameter to the execve() command, the environment variables of the process are printed. Thus, execve() runs the new program inside the calling process and the environment variables of the program are generated as the output.

TASK 4: Environment variables and system()

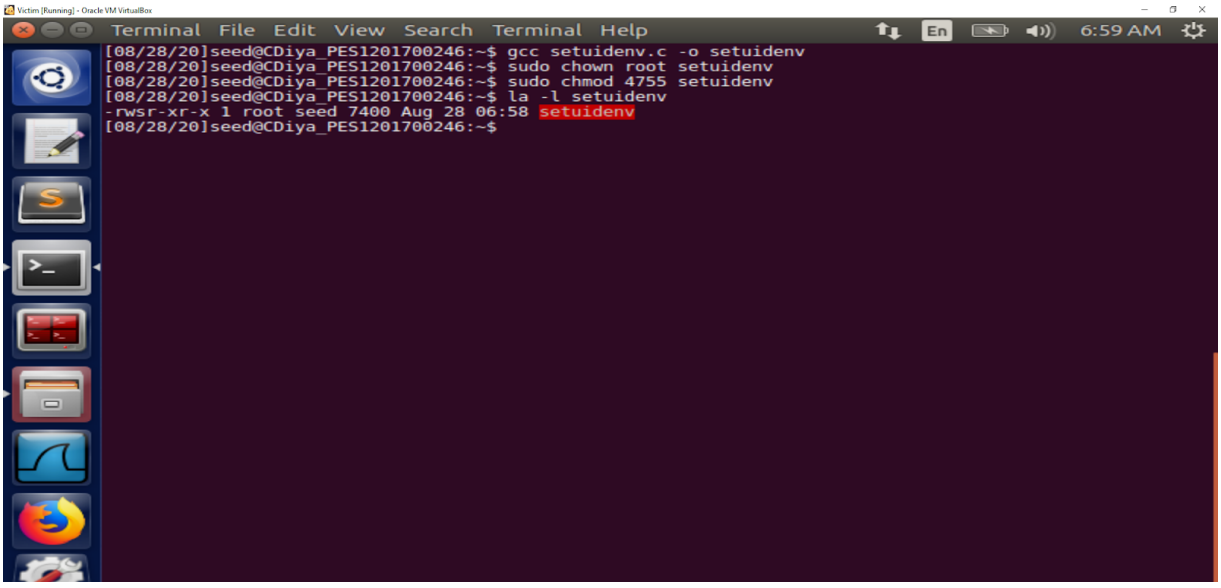




Observation:
The system() executes a child process that executes the shell command specified by calling /bin/sh -c command, and returns after the command has been completed. Thus, it can be seen from the screenshots above that the system() passes down the environment array to the new program which gets printed.

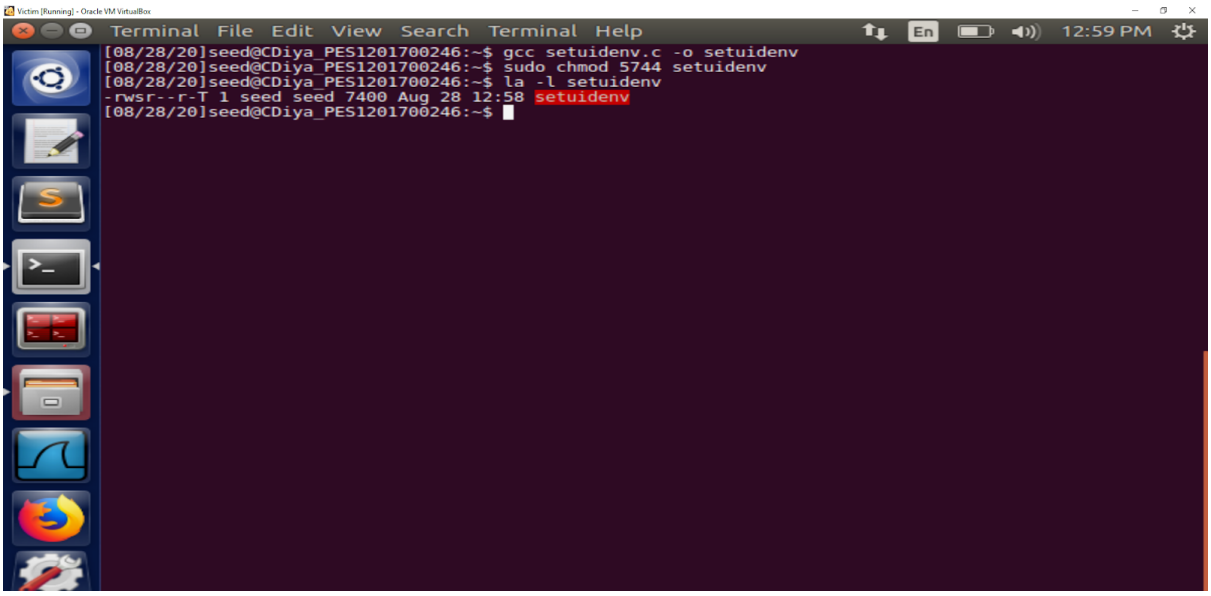
TASK 5: Environment variable and Set-UID Programs

STEP 1:



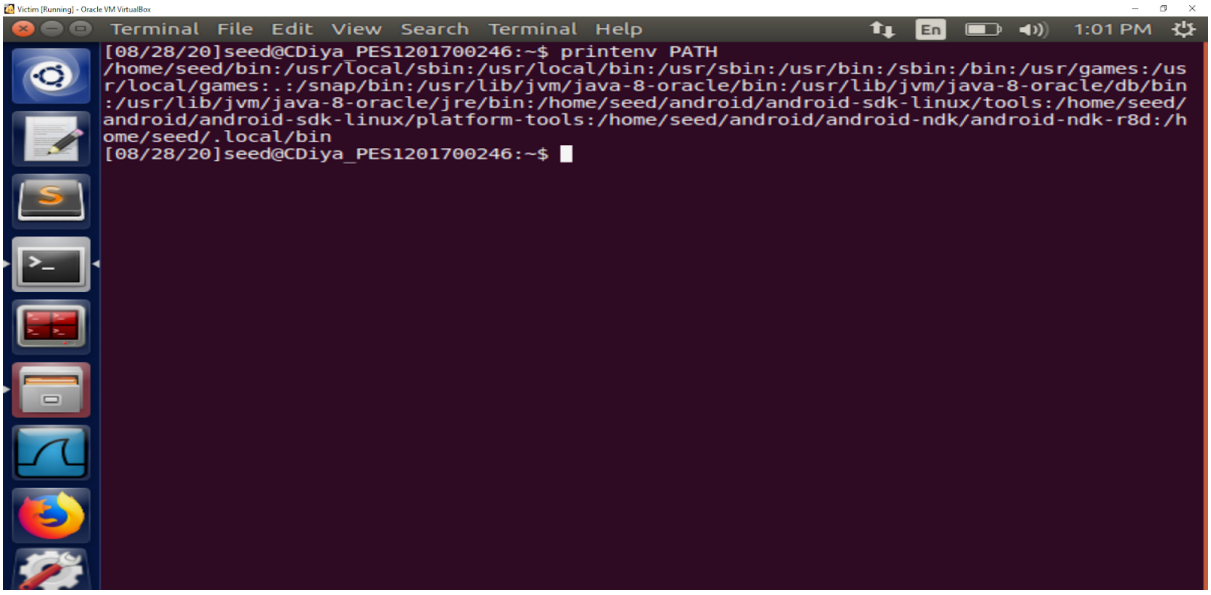
Observation: It can be inferred from the screenshot above that the setuid command is used to set the program owner to root. The following chmod 4755 command is used to sets read, write, and execute permissions for users, and sets read and execute permissions for Group, but not write permission.

STEP 2:

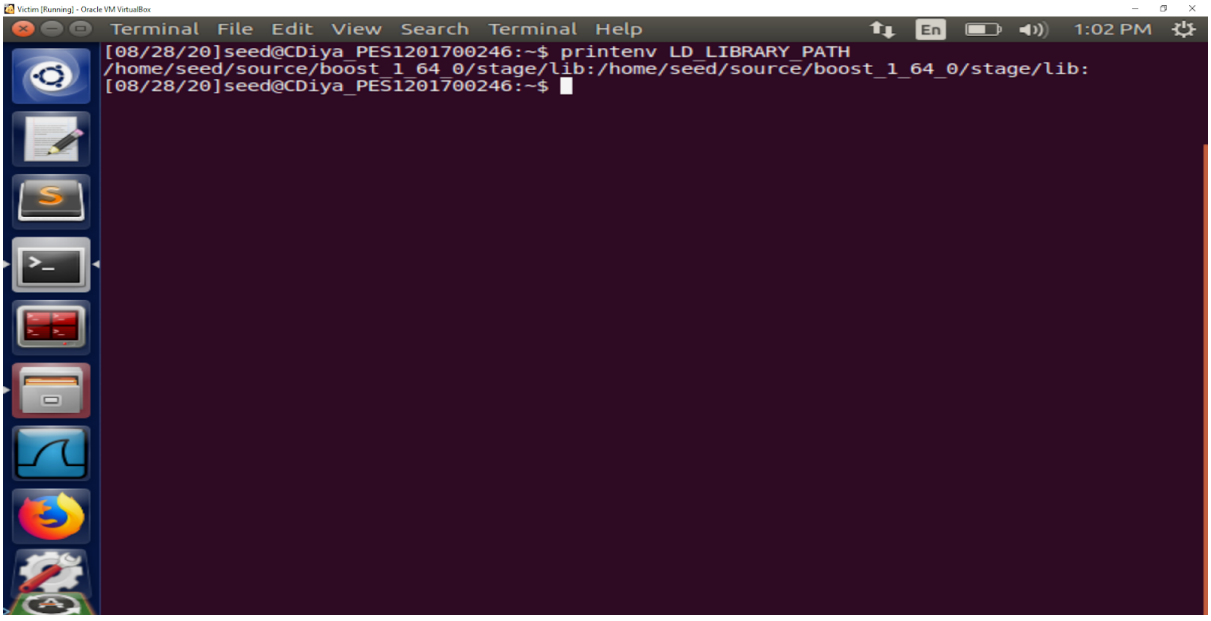


Observation: the chmod 5755 ensures that the owner of the program can read, write and execute but the group now has only read permissions.

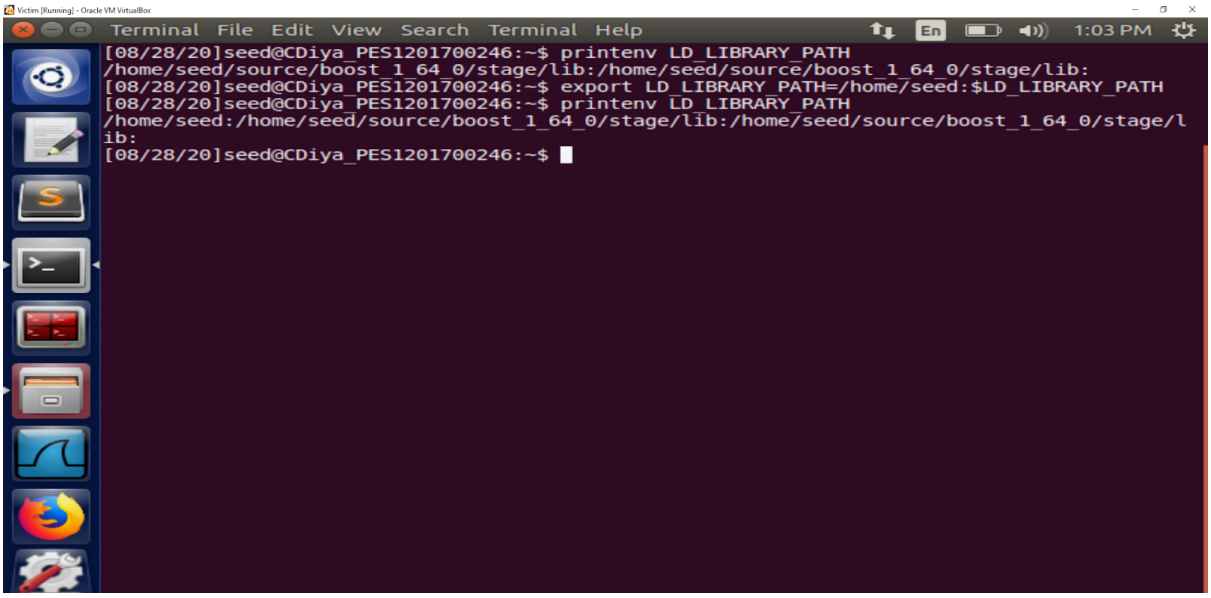
STEP 3



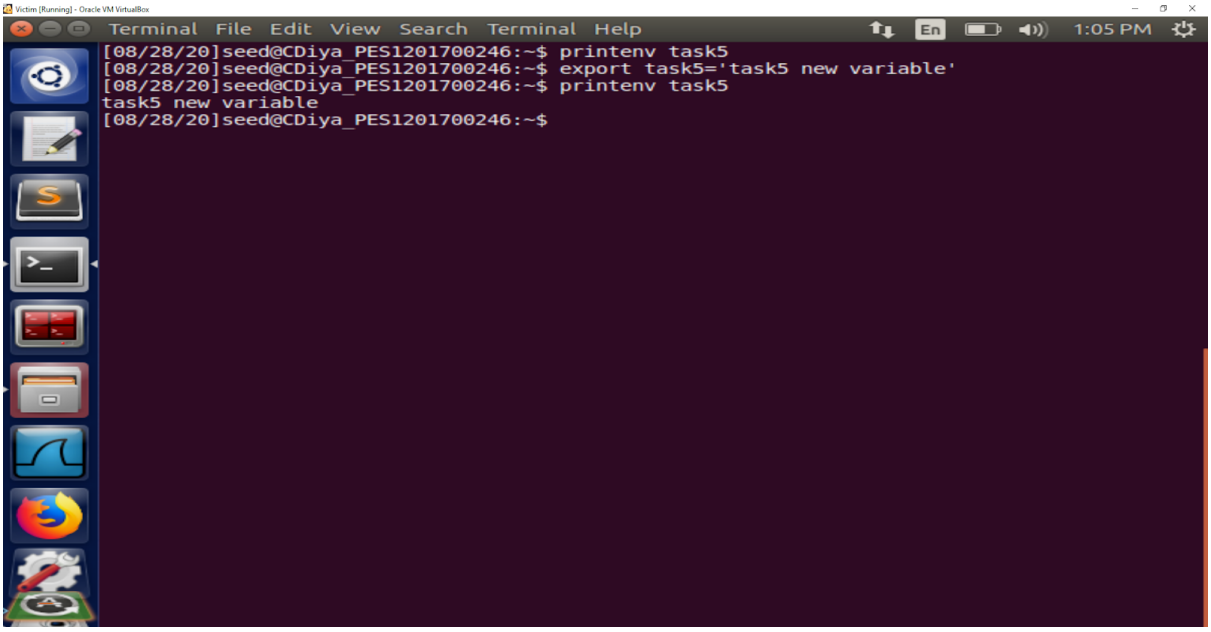
Observation: prints the value present in the PATH environment variable



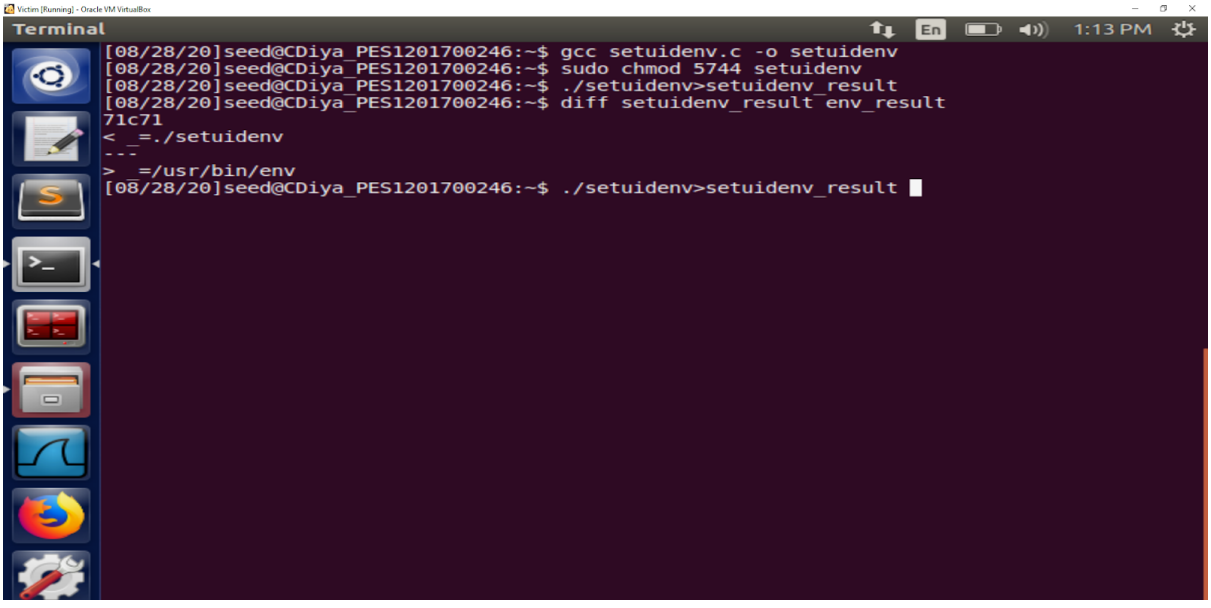
Observation: prints the value present in the LD_LIBRARY_PATH environment variable



Observation: The screenshot below shows the setting of the LD_LIBRARY_PATH environment variable using the export command.

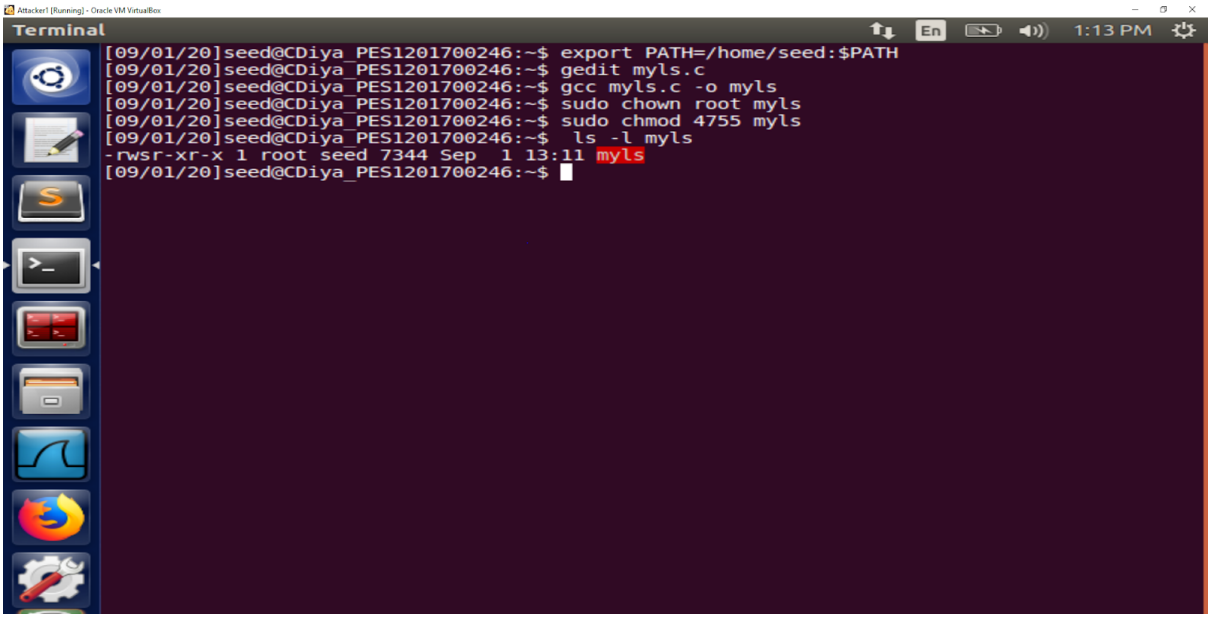


Observation: The screenshot below shows the setting of environment variables using the export command.



Observation: When the Set-UID program from Step 2 is run in the shell, the shell forks a child process, and uses the child process to run the program. There is an addition of a new environment that can be seen above and thus, all the environment variables that were set in the shell process (parent) get into the Set-UID child process.

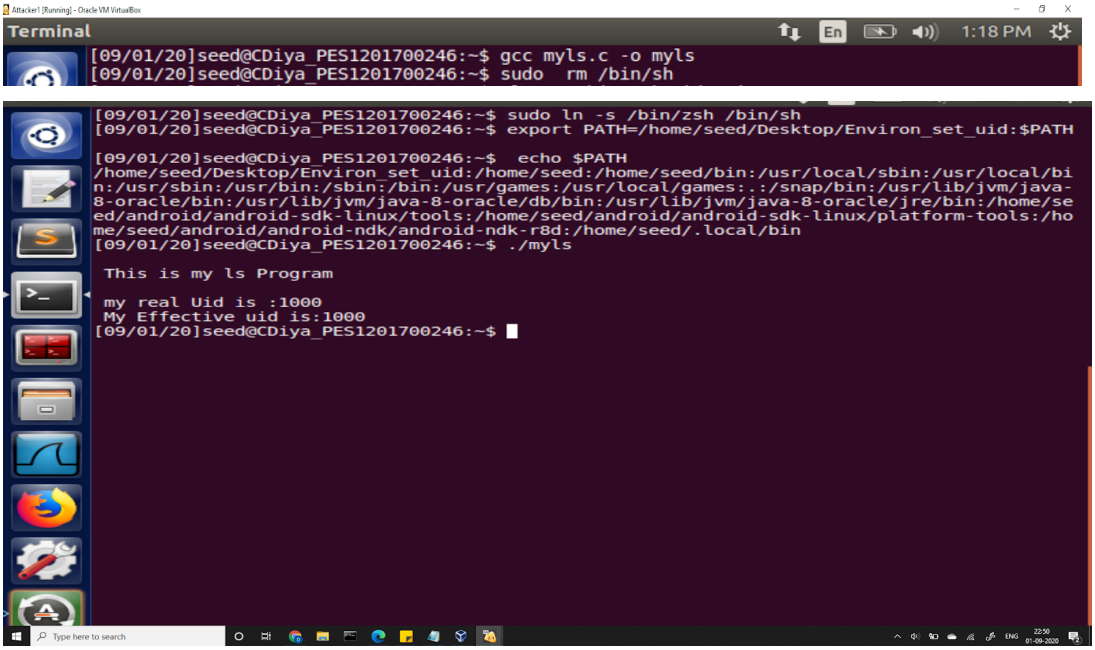
TASK 6: The PATH Environment variable and Set-UID Programs



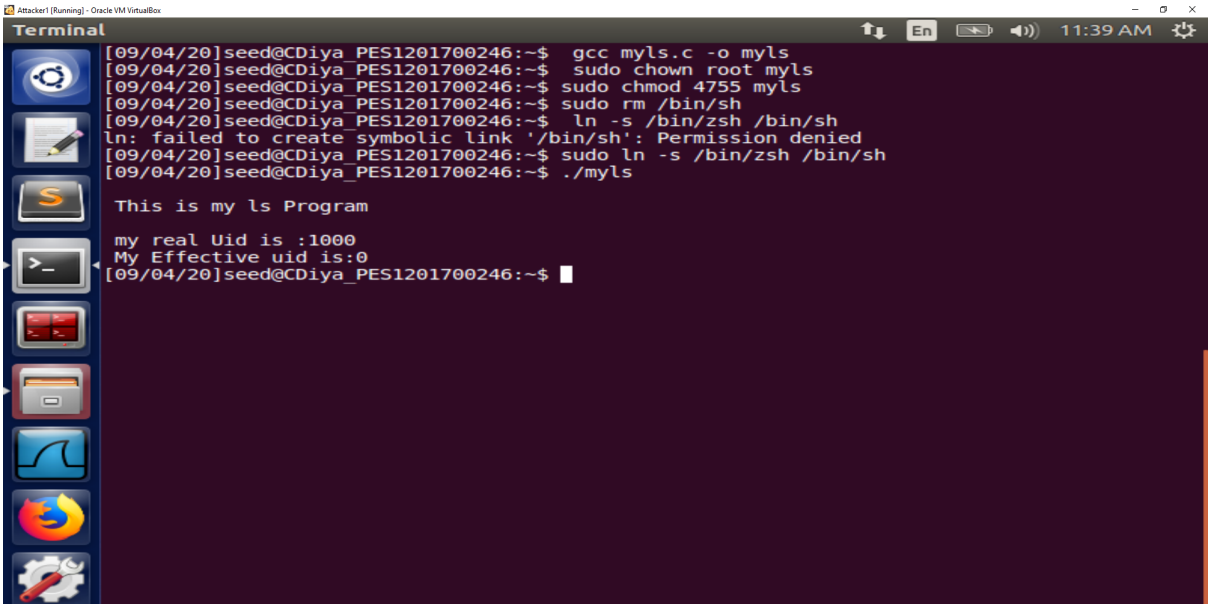
Observation:

Can you let this Set-UID program run your code instead of /bin/ls? If you can, is your code running with the root privilege?

Answer: The screenshot above shows that the SET-UID program can be used in place of /bin/ls. It also shows that the program runs with root privileges which can be inferred that the root is the owner of the program.



Observation:
Real UserID : It is an account of the owner of this process. It defines which files that this process has access to.
Effective UserID : It is normally the same as Real UserID, but sometimes it is changed to enable a non-privileged user to access files that can only be accessed by root. SET UID is used to set this.
When the program is run without changing any privileges, the real and effective ID is the same(ie 1000). Additionally, if root access was enabled: the following screenshot is obtained.

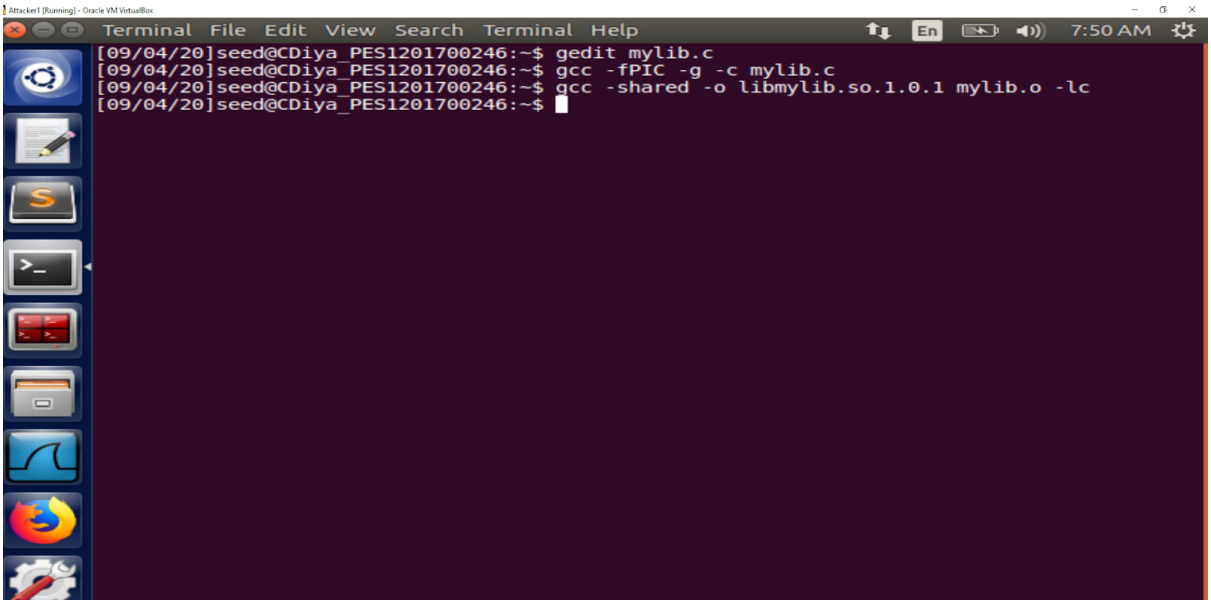


Observation:(real id=1000 and effective ID=0)
It can be observed above that if the owner is changed to root, the effective ID which was externally set becomes 0. 0 EFFECTIVE ID implies that the root is the owner and the program has root privileges

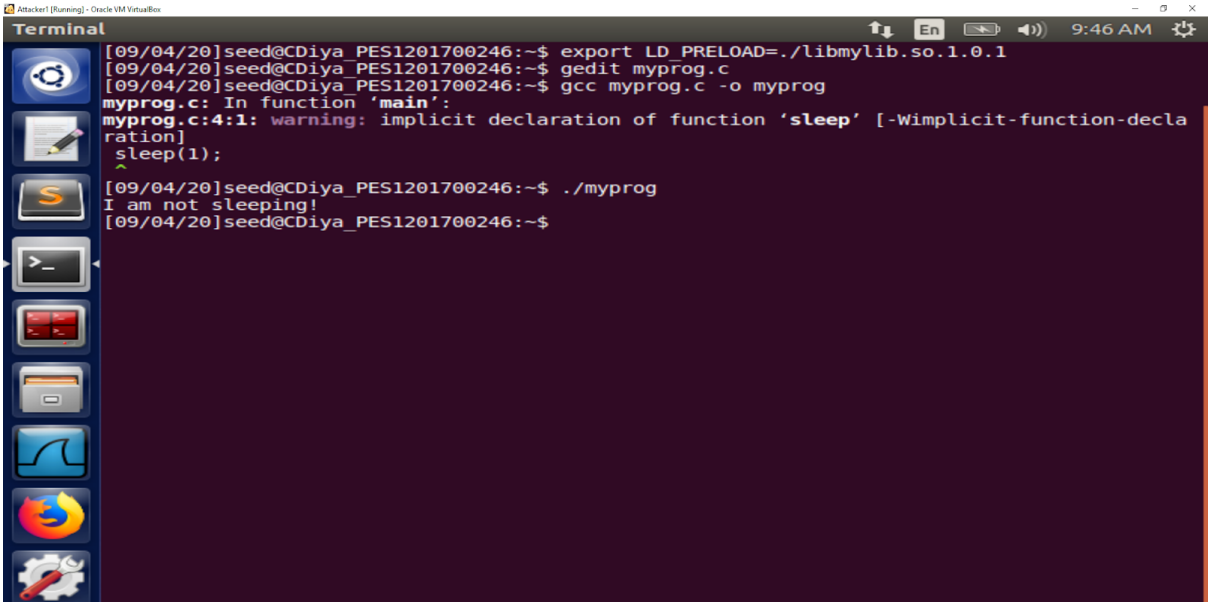
TASK 7:The LD PRELOAD environment variable and Set-UID Programs

Step 1:

1. The program is compiled using the steps given.

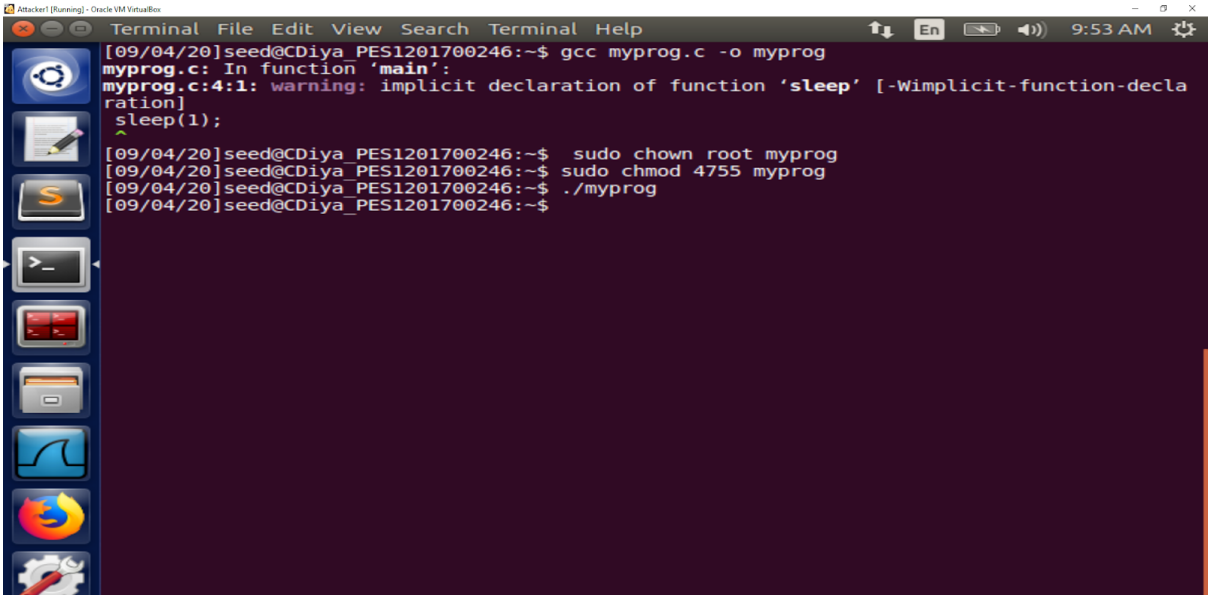


2. Make myprog a regular program, and run it as a normal user.



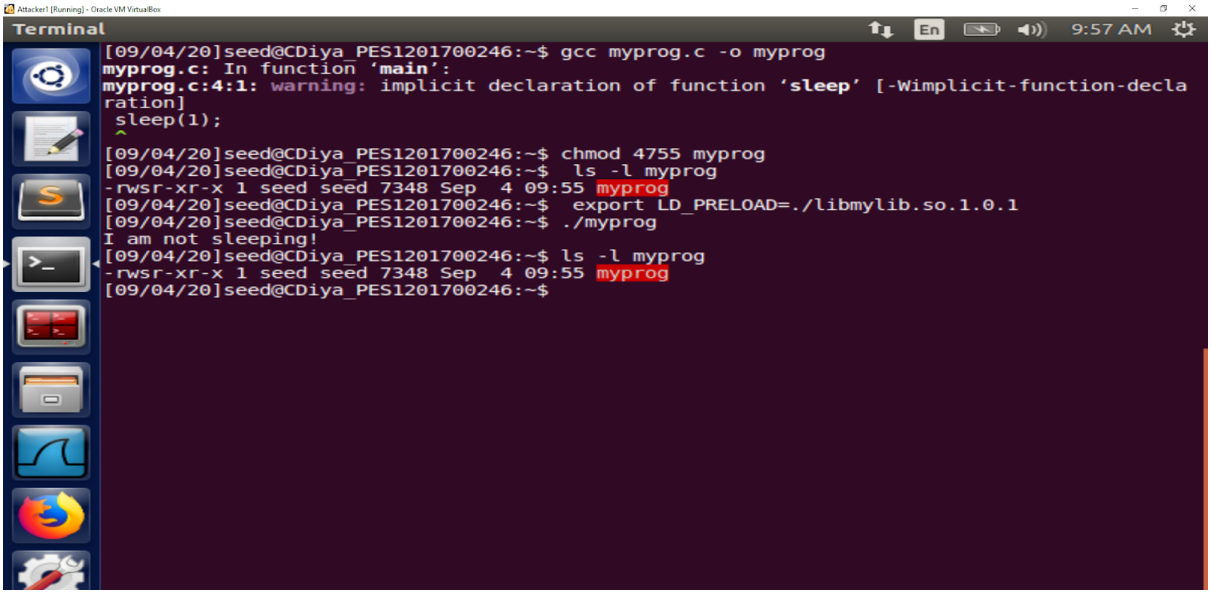
Observation: From the screenshot above, it can be inferred that when the given program is run, the output is printed “I am not sleeping”. This is because the root UID has not been set yet and it is run in a normal mode.

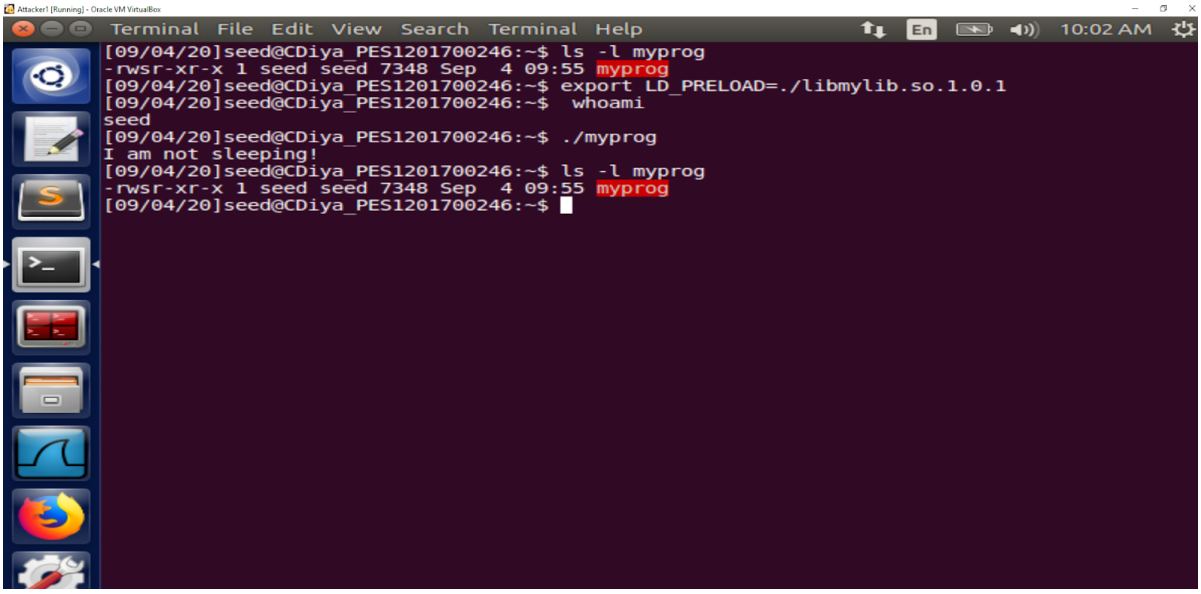
3. Make myprog a Set-UID root program, and run it as a normal user.



Observation: When the program runs after setting the UID root program it sleeps for 1 second.

4. Make myprog a Set-UID root program, export the LD PRELOAD environment variable again in the root account and run it.

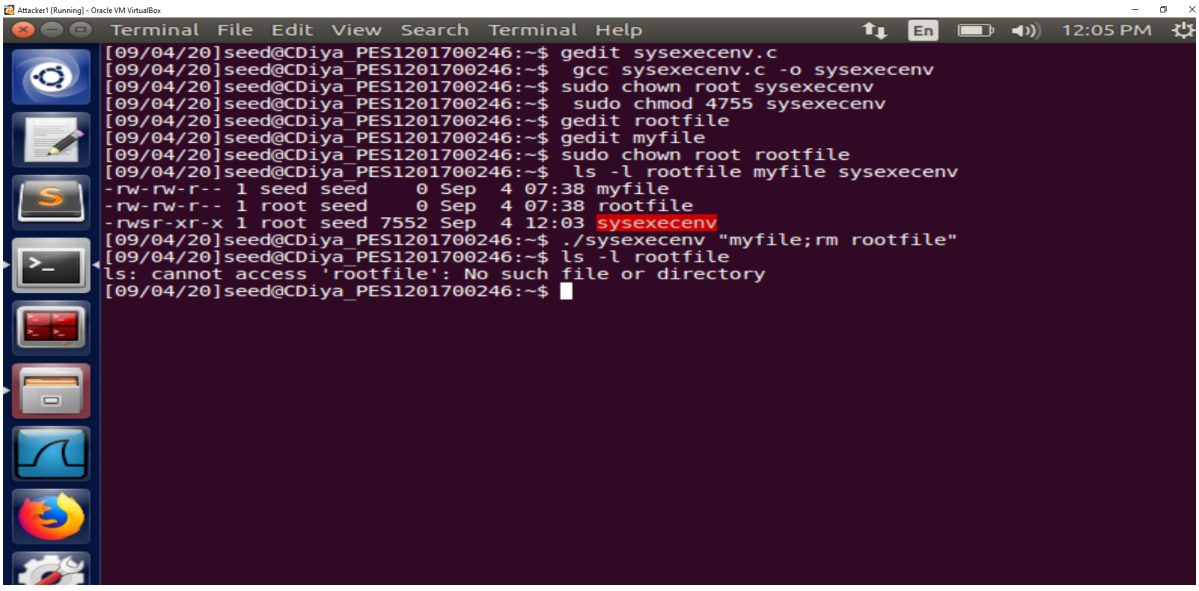




Observation:From the above screenshots, when the seed is the user, the seed user would have all the execute permissions.
While using the root user it can be inferred that the owner and group is of the root.Thus, access of the root library is possible

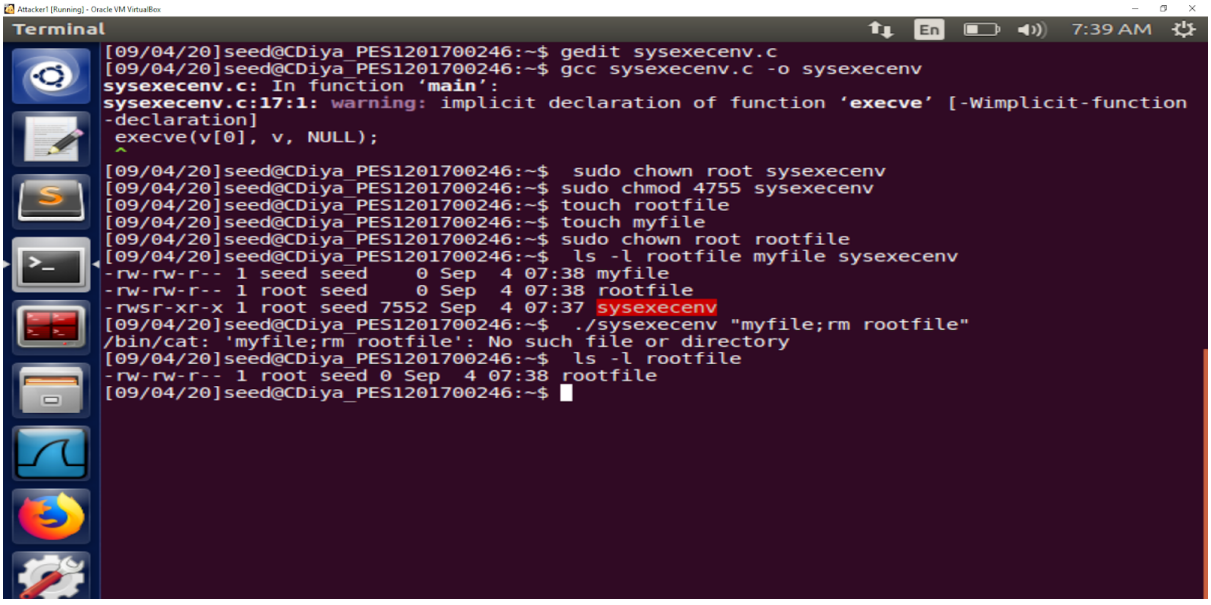
TASK 8: Invoking External Programs Using system() versus execve()

STEP 1:



Observation: Two files were created, two files “myfile”(owner is seed) and “rootfile”(owner is root). Bob can compromise the integrity of the system by deleting the rootfile using the program given. The program executable is used to enter the rm command and the rootfile(root owner) which is passed to the system() command. Thus, despite the rootfile having root ownership, Bob can delete the file using the system(). Thus, system() is not safe because the PATH environment variable affects the behavior of system(), because the variable affects how the shell works.

STEP 2:

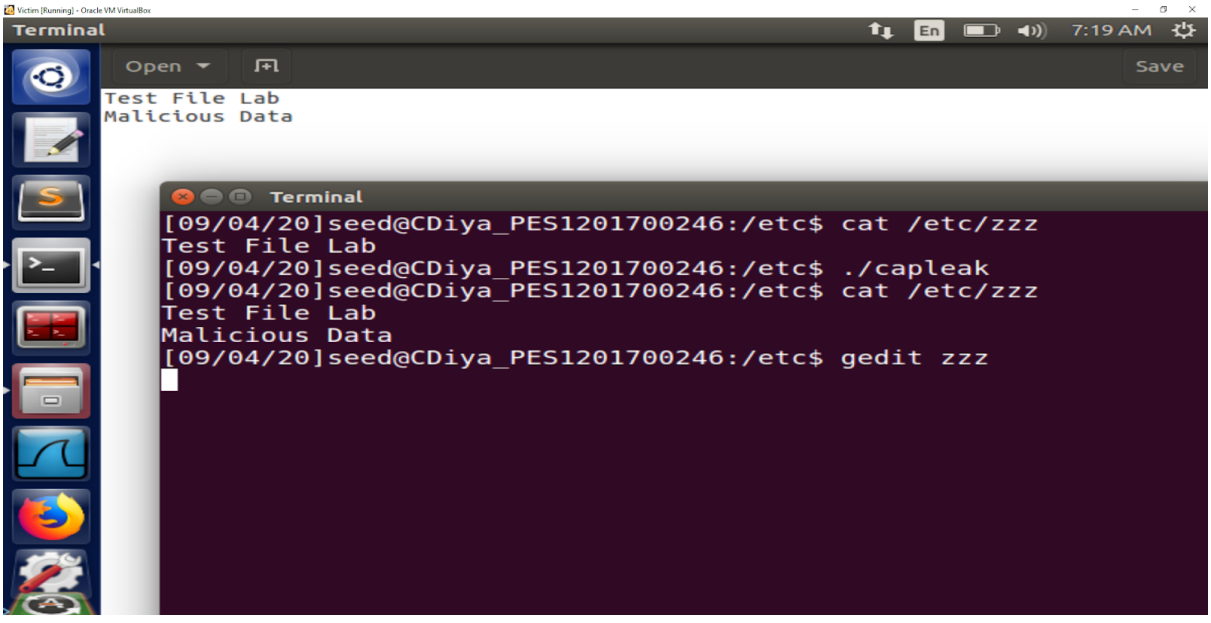


Observation: system() is replaced by the execve(). When the program above is compiled and the executable is used to remove the rootfile, it can be seen that the rootfile does not get deleted. This is because execve() does not invoke the shell because PATH environment variable does not affect the behavior of execve(). Thus, it does not access environment variables due to which the file does not get removed and the integrity is not compromised.

\

TASK 9: Capability Leaking

The initial content of the file can be seen above:



Observation: “Malicious Data” has been added to the /etc/zzz file. Due to root privileges that were retained, read , write and execute privileges allowed the file to be written into.

If the root privileges are no longer needed, it’s time to relinquish the root privileges permanently. When revoking this privilege, one of the common mistakes is capability leaking. The process may have gained some

privileged capabilities when it was still privileged; when the privilege is downgraded, if the program does not clean up those capabilities, they may still be accessible by the non-privileged process.