**Environment Variable and SET-UID Lab - Seed Lab Solution with Explanation**

By Salameh Fawwaz Alawabdeh

Table of Contents

[Task 1: Manipulating Environment Variables 2](#_Toc85929625)

[Task 2: Passing Environment Variables from Parent Process to Child Process 4](#_Toc85929626)

[Task 3: Environment Variables and execve() 5](#_Toc85929627)

[Task 4: Environment Variables and system () 7](#_Toc85929628)

[Task 5: Environment Variable and Set-UID Programs 8](#_Toc85929629)

[Task 6: The PATH Environment Variable and Set-UID Programs 14](#_Toc85929630)

[Task 7: The LD\_PRELOAD Environment Variable and Set-UID Programs 15](#_Toc85929631)

[Task 8: Invoking External Programs Using system() versus execve(): 18](#_Toc85929632)

[Task 9: Capability Leaking 20](#_Toc85929633)

[Task 10: Lab Clean-up 20](#_Toc85929634)

## Task 1: Manipulating Environment Variables

* the shell /sbin/nologin, which is a dummy shell that not only stops the user from logging it, but also logs the login attempt to the syslog daemon (Turnbull, Lieverdink & Matotek, 2009).
* This method is usually used on Red Hat hosts to show that this user cannot log on. On Ubuntu hosts the shell /bin/false is used. Most users will have a shell entry that references the binary that launches their shell, for instance, /bin/bash (Turnbull, Lieverdink & Matotek, 2009).

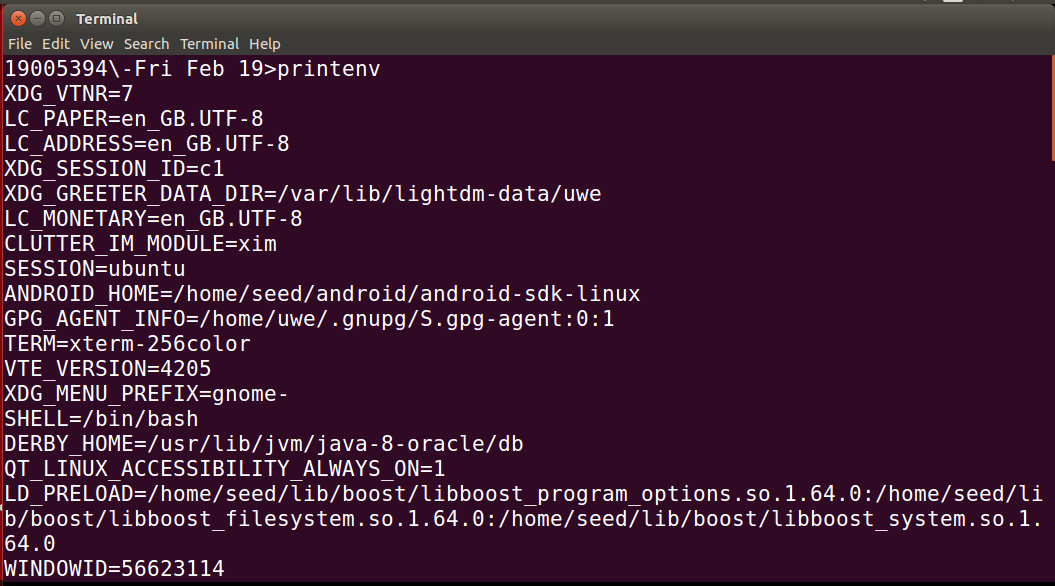


Figure 1

After export

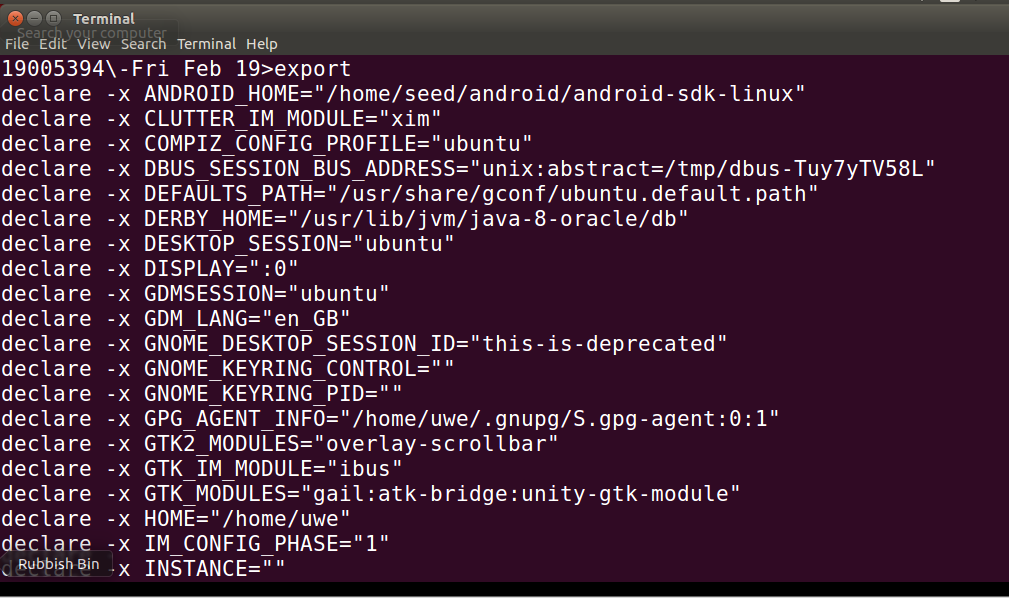


Figure 2

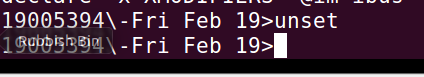


Figure 3

## Task 2: Passing Environment Variables from Parent Process to Child Process

the child inherits the environment variable from the parent without being change.

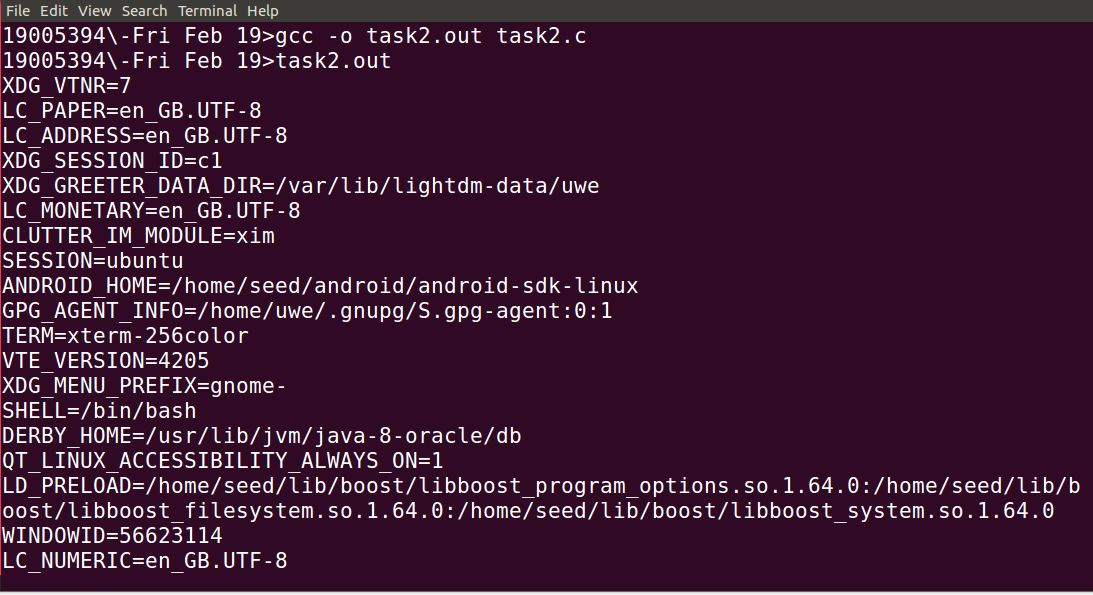


Figure 4

I have used the command task2.out > child, to inherit the exact environment variable from the parent to the child.

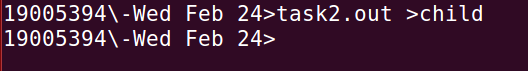


Figure 5

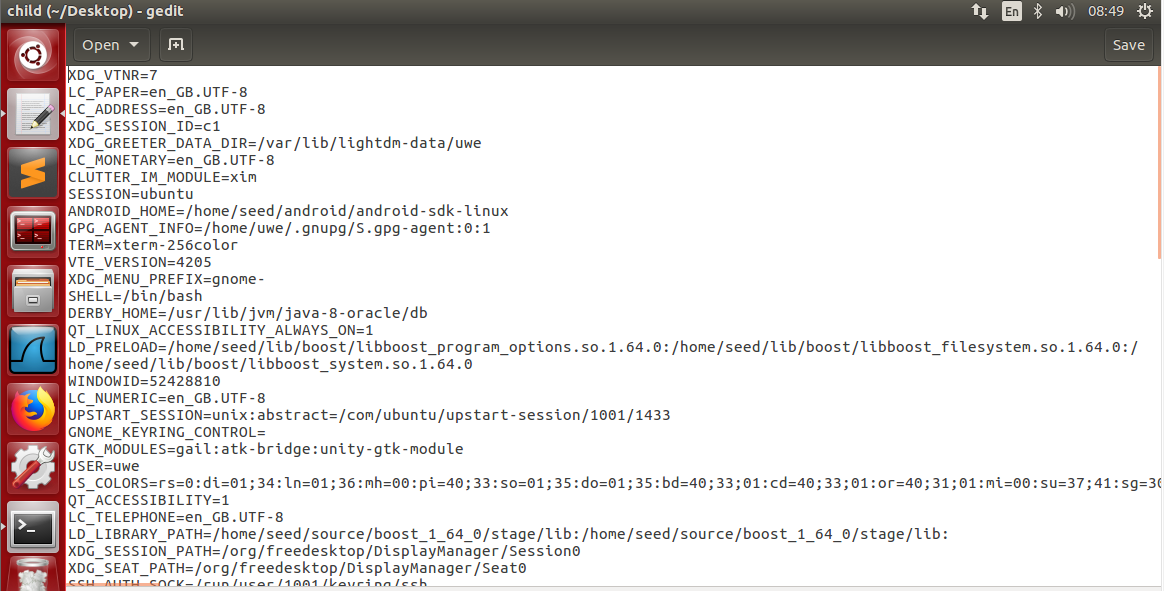
And the file automatically created after I finished that command and the result is stored in it.

Figure 6

## Task 3: Environment Variables and execve()

**Step 1:**

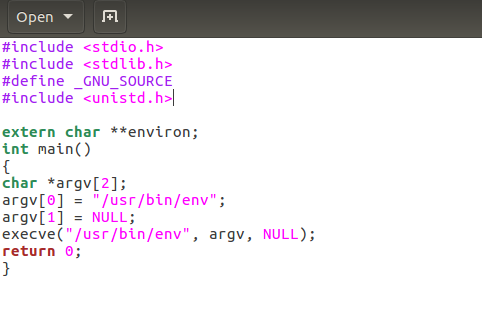


Figure 7

I have added GNU extension because the terminal gave me this message "implicit declaration of function", which I found out that \_GNU\_SOURCE should defined before including unistd.h as it is a GNU extension.

And after I have compiled and ran the program it showed that it is empty because in the code it is NULL so it will not pass any environment variables.

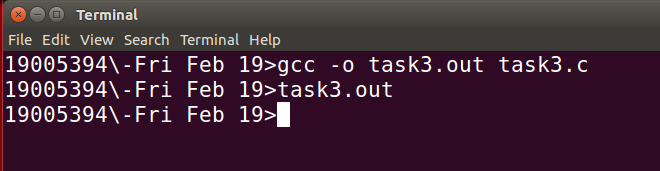


Figure 8

**Step 2:**

After I did change the invocation of execve() in Line ➀ to execve("/usr/bin/env", argv, environ); , I did compile and run my program task3 again.

what exactly happened to the environment variables of the recent process were automatically inherited.

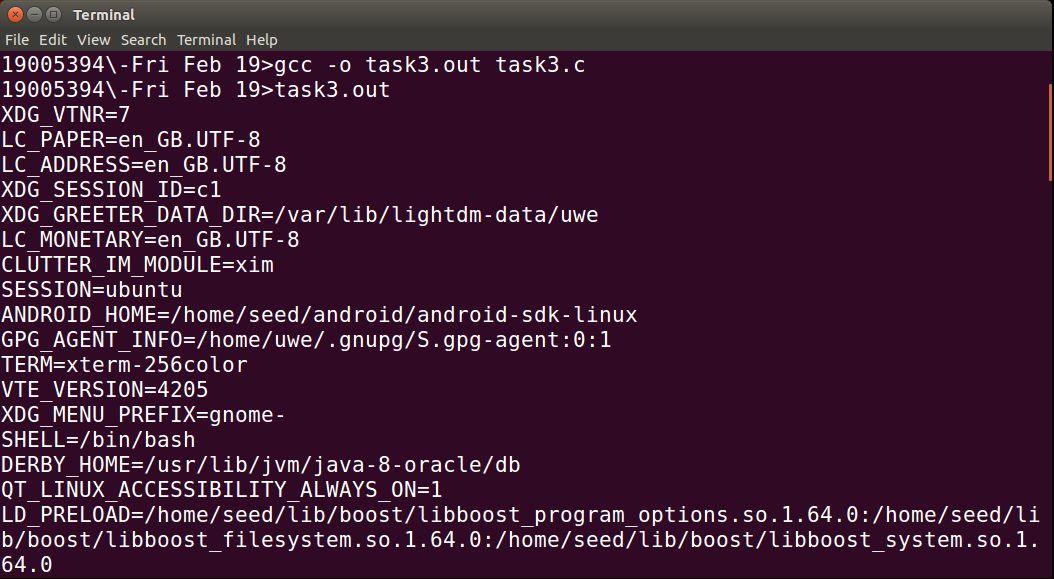


Figure 9

**Step 3: Question:** Please draw your conclusion regarding how the new program gets its environment variables.

The new program obtains its environment variables by taking what passed to it by the old program.

## Task 4: Environment Variables and system ()

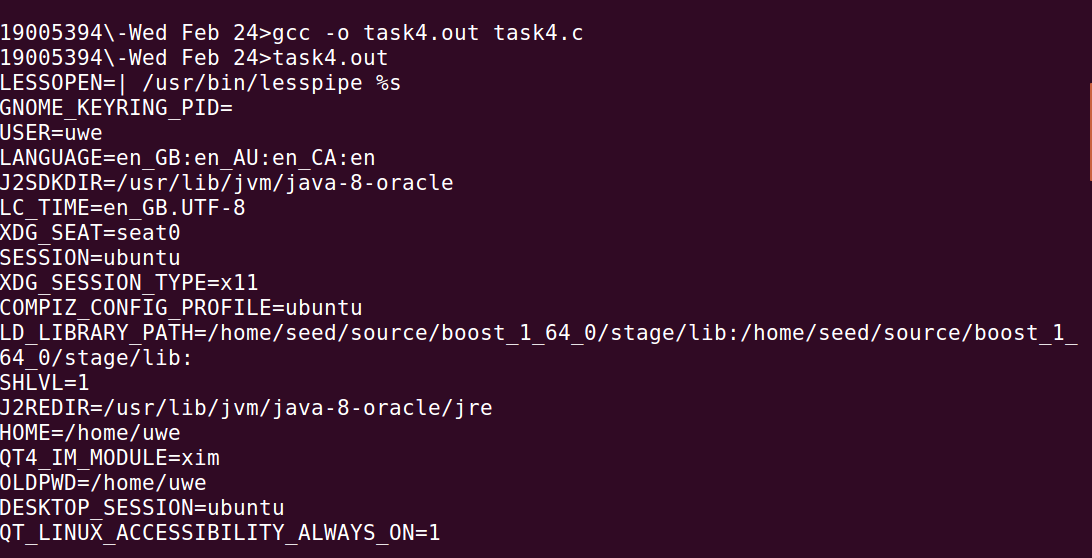


Figure 10

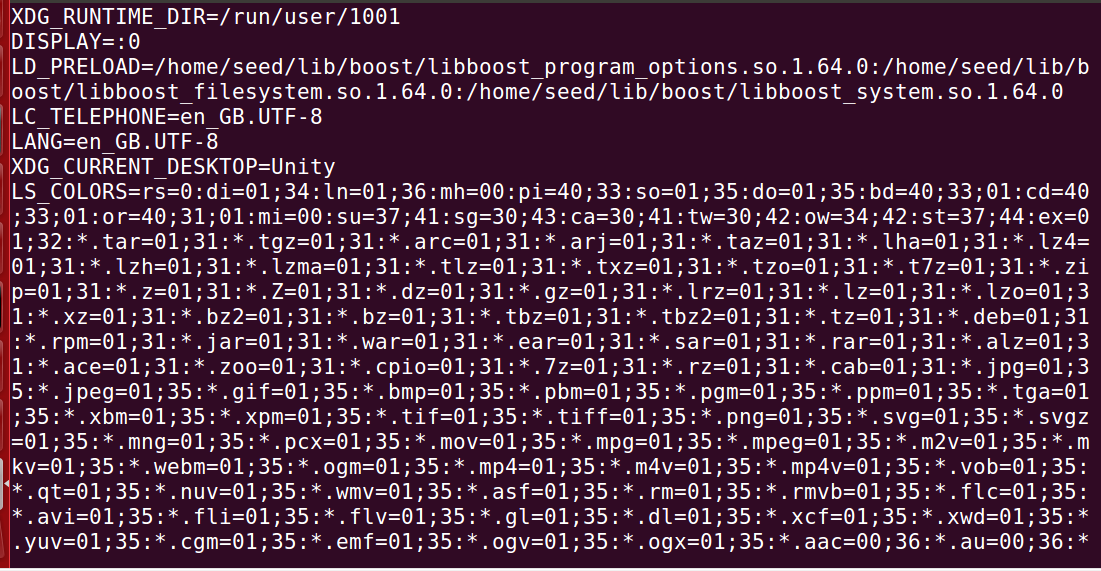


Figure 11

## Task 5: Environment Variable and Set-UID Programs

**Step 1:**

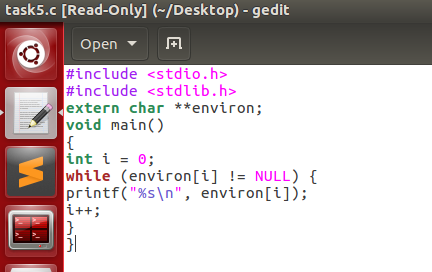


Figure 12

**Step 2:**

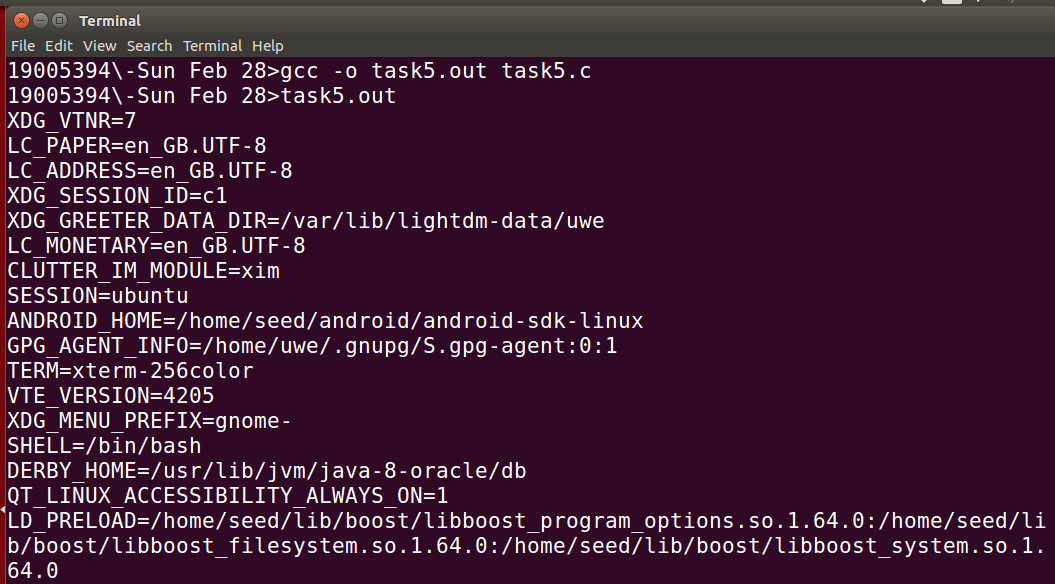


Figure 13

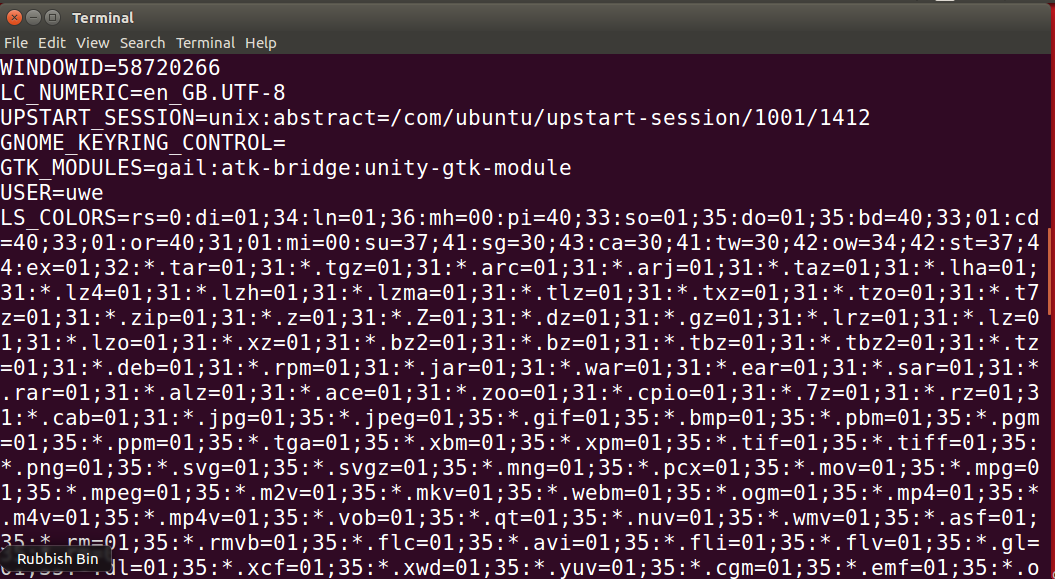


Figure 14

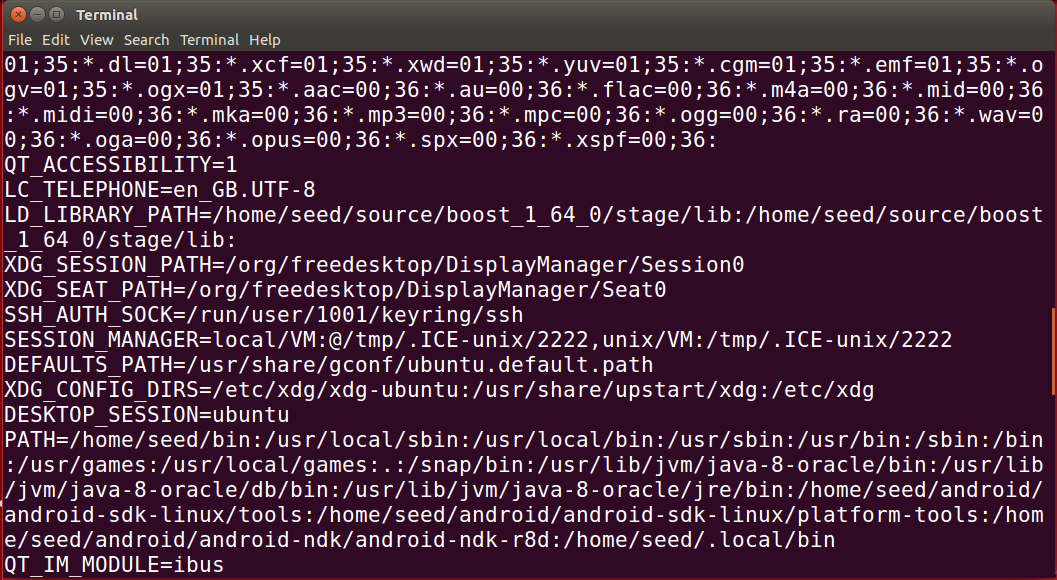


Figure 15

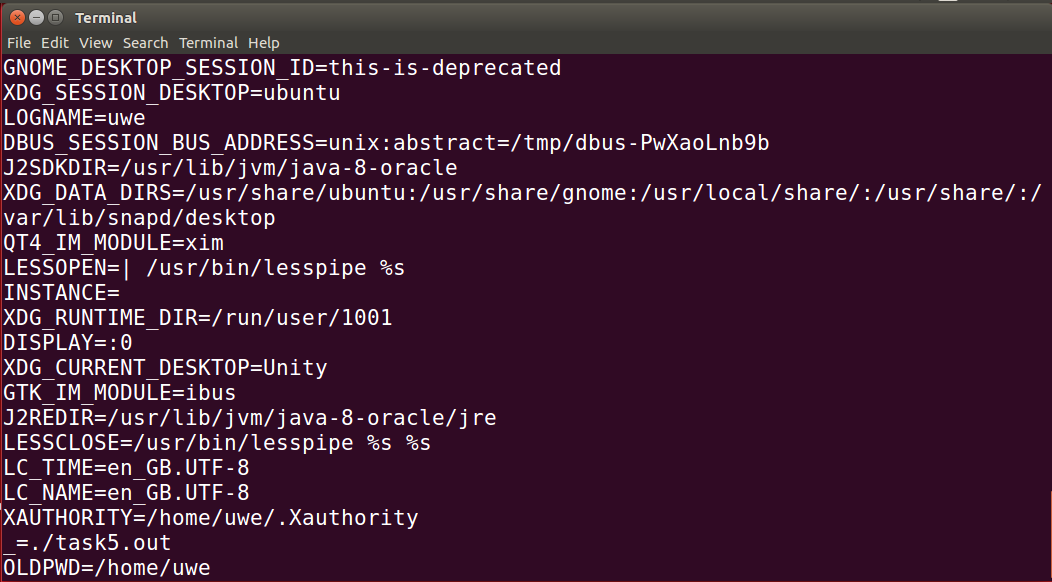


Figure 16

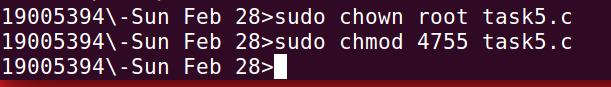


Figure 17

After I did change its ownership to root, and made it a Set-UID program a lock has shown on the program file and the file now is a Set\_Uid program and that happened with chown root command which it changes the ownership of the file to root, for the selected file& chmod 4755 which change the permissions of the folder to special user.

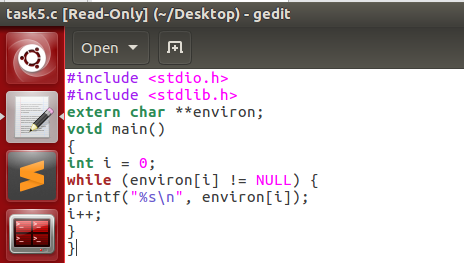


Figure 18

And in the next screen shot if we look at the file, we can see it is owned by the root, and that’s mean is when the program runs it runs as the user owner of the program (not the standard user)

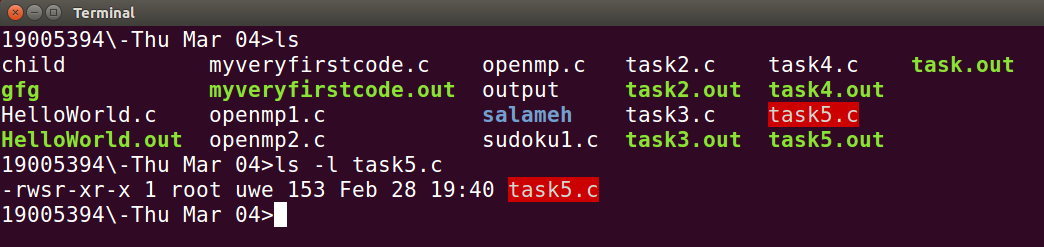


Figure 19

**Step 3:**

In the next screenshot I have exported the PATH & LD LIBRARY PATH environment variables:

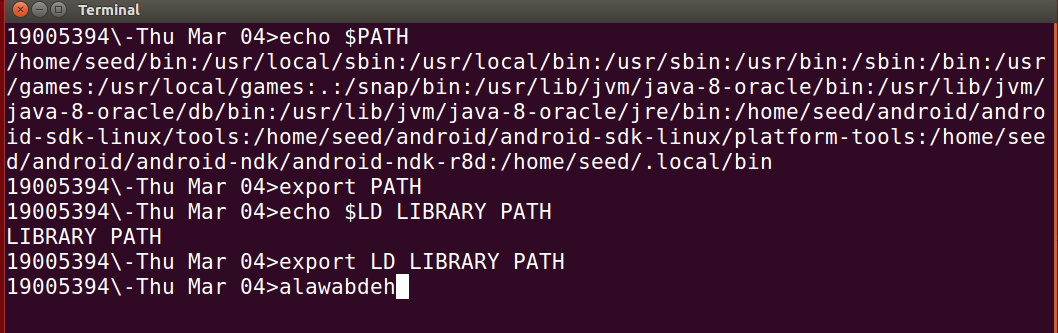
****

Figure 20

In the following screen shot I have created an environment variable, I called it ALAWABDEH and I added Hello to it, after that I exported it:

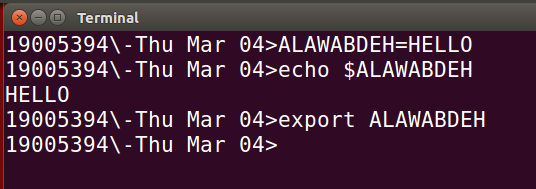
****

Figure 21

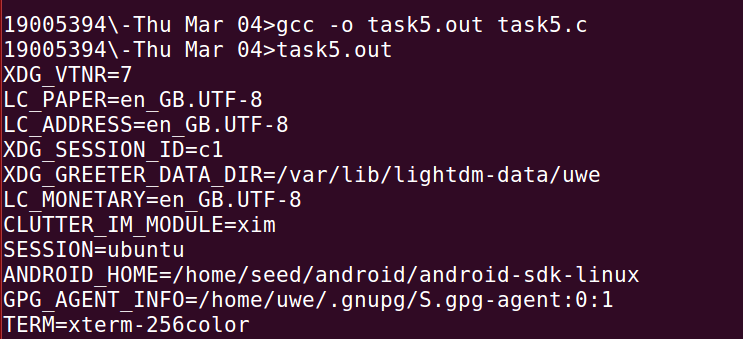
****

Figure 22

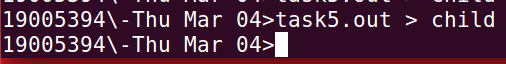
****

Figure 23

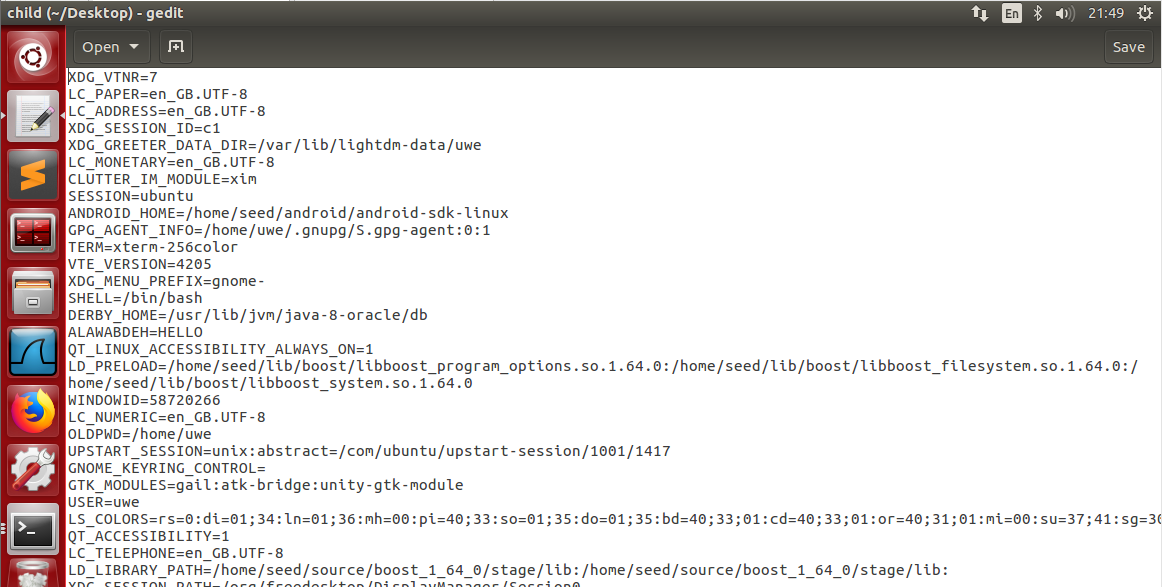
****

Figure 24

Of course all the environment variables which were in the program inherit to the Set\_Uid child process, as shown in the following screenshot, because I did use Export to the environment variables, which export is a built-in shell command for Bash that is used to export the environment variable to give the approval for new child processes to inherit it.

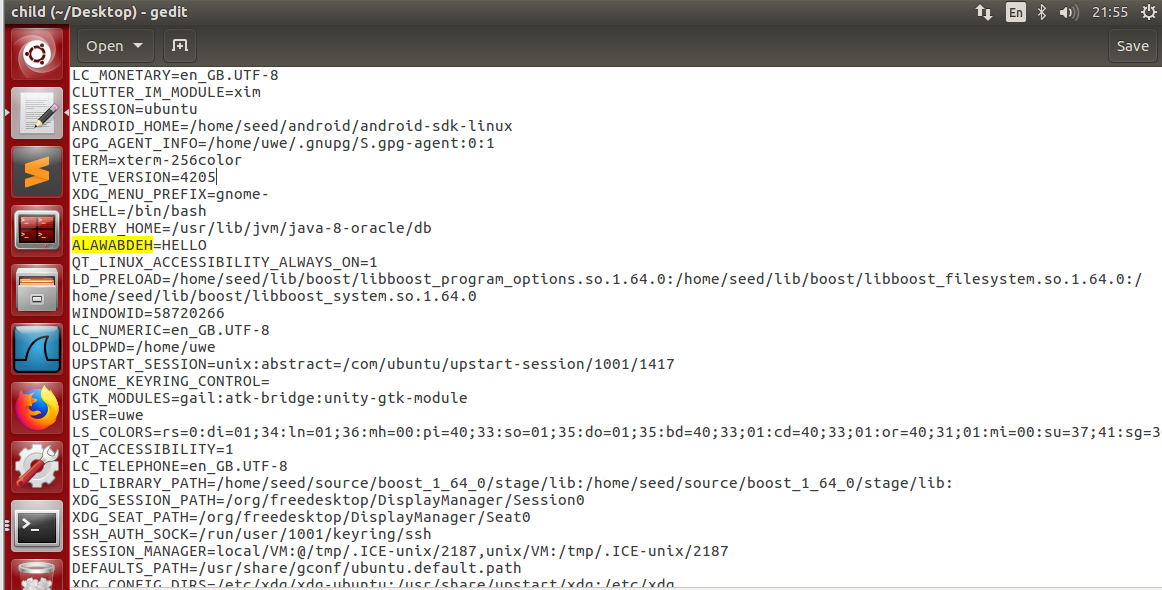
****

Figure 25

**Question:** Describe your observation. If there are surprises to you, describe them.

Not all of them are included in the Set-UID child procces. PATH has been modified, and my surname has been added, however LD LIBRARY PATH cannot be located.

## Task 6: The PATH Environment Variable and Set-UID Programs

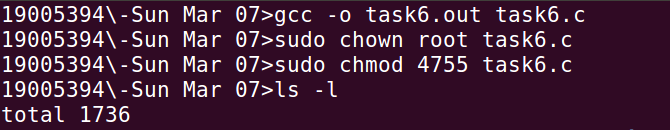
****

Figure 26

****

Figure 27

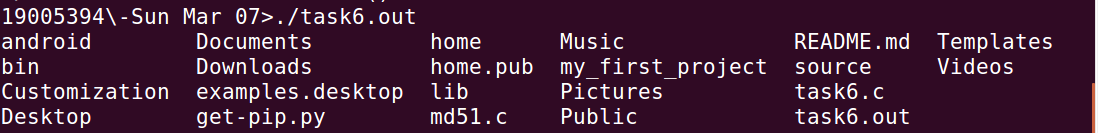
****

Figure 28

I have removed the currently /bin/sh by rm /bin/sh command first and after that I pointed it to /bin/zsh target using sudo ln -s /bin/zsh /bin/sh:

****

Figure 29

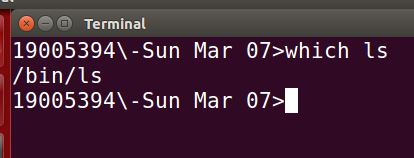
****

Figure 30

the arrow in the next screenshot means that the bin/sh is now linked to the shell program which called zsh.

****

Figure 31

it showed the contents which are inside our current folders just like the ls command without highlight the set\_uid programs on the contrary of the ls command that will highlight the set\_uid programs.

## Task 7: The LD\_PRELOAD Environment Variable and Set-UID Programs

**Step1:**

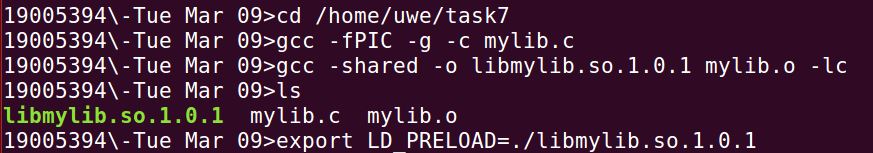


Figure 32

after the export command, it's now exported to the environment variables.

**Step 2:**

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

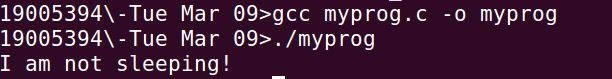


Figure 33

it print out the line "" in one second because I used sleep(1) which sleep delays the program for a specific amount of time ([https://www.freebsd.org/cgi/man.cgi?query=sleep&sektion=1)\***refrence**](https://www.freebsd.org/cgi/man.cgi?query=sleep&sektion=1)*refrence)

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

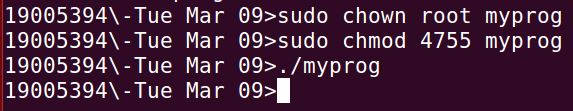


Figure 34

The program is became owned by the root so the normal user cannot run it, and when I did run it as a normal user it delayed for one second without the results because the LD PRELOAD environment variable is disable, so it neglect the LD PRELOAD environment variable and use the system's default sleep() function in this case.

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

I have used sudo su – to switch user to go to the root account.

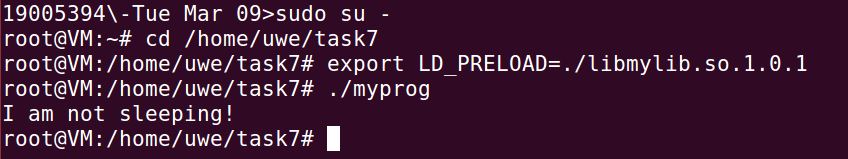


Figure 35

when I run it as a root its became valid again, Whereas, when I ran it as a normal user in program 2 it was invalid and did not show the line because the the LD PRELOAD environment variable was disabled. so It overrode the sleep() function and used the LD PRELOAD environment variable.

4) Make myprog a Set-UID user1 program (i.e., the owner is user1, which is another user account), export the LD PRELOAD environment variable again in a different user’s account (not-root user) and run it.

I have created a new user “user1”.

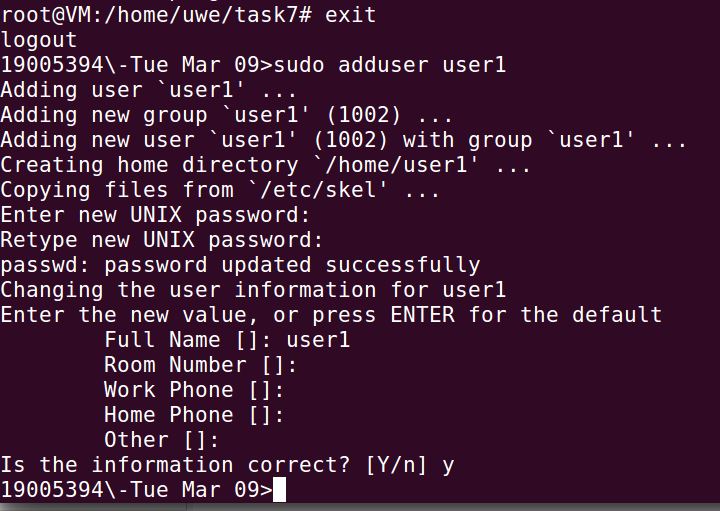


Figure 36

I did copied myprog to userprog , and it automatically was owned by seed so I did change it to user1, and after that I did run the userprog but the LD PRELOAD environment variable is disabled .

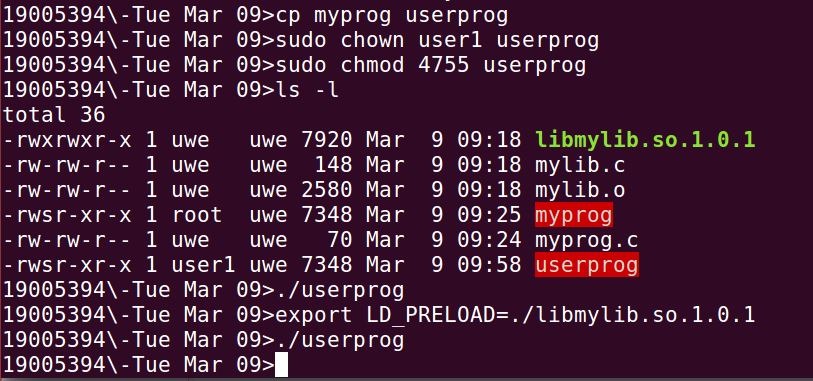


Figure 37

 it did not overrode sleep() function

Step 3:

These behaviours might be caused by different applications of the environment variable LD PRELOAD. Only when the program's owner and the exporting place of LD PRELOAD are all the same may a user execute the program using the overrides function.

## Task 8: Invoking External Programs Using system() versus execve():

**Step1:**

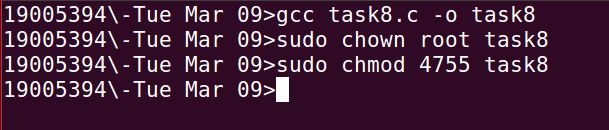


Figure 38

In the next screen shot I invoked shadow file

A shadow password file is a system file that stores encryption user passwords and hides them from unauthorize people).

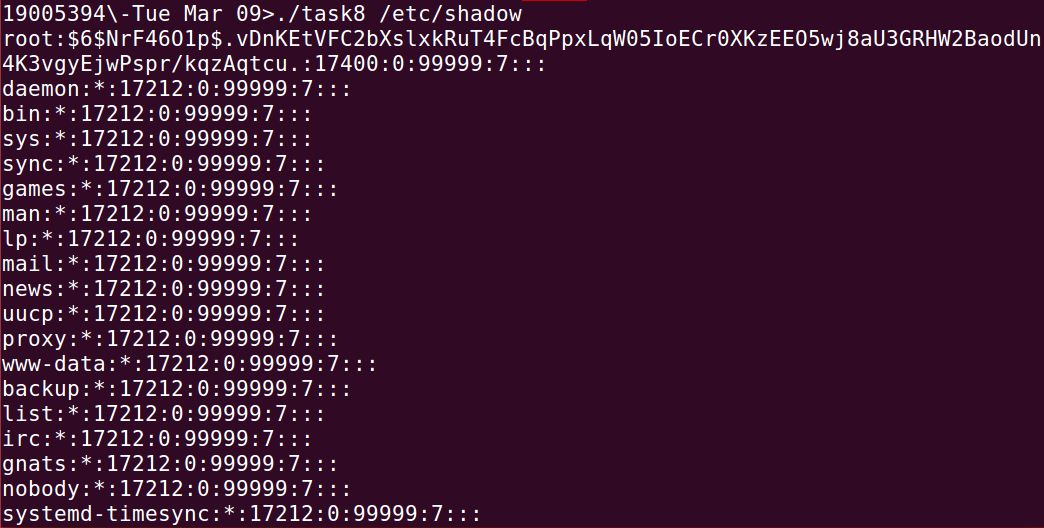


Figure 39

And finally, I got a new shell using this input

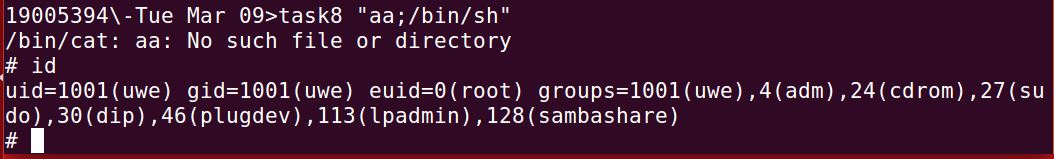


Figure 40

because of the system() in the program, it will call the other dash instead of the zsh that is why I cannot view it so I used “sudo ln -sf /bin/zsh bin/sh” from the task 6, to point it to insecure shell, after that I can invoke the shadow file and read its secret content of it and we can take advantage of that to make the system() disabled in a reliable way.

Finally, I got the root shell (euid=0(root)) which give me the privileges to modify, remove files or even destroy the system.

**Step2:**

it gave me an alert "implicit declaration of function 'exceve'" so i did added exceve header(#include <unistd.h>) in the program.

I did change the program to set\_uid program.

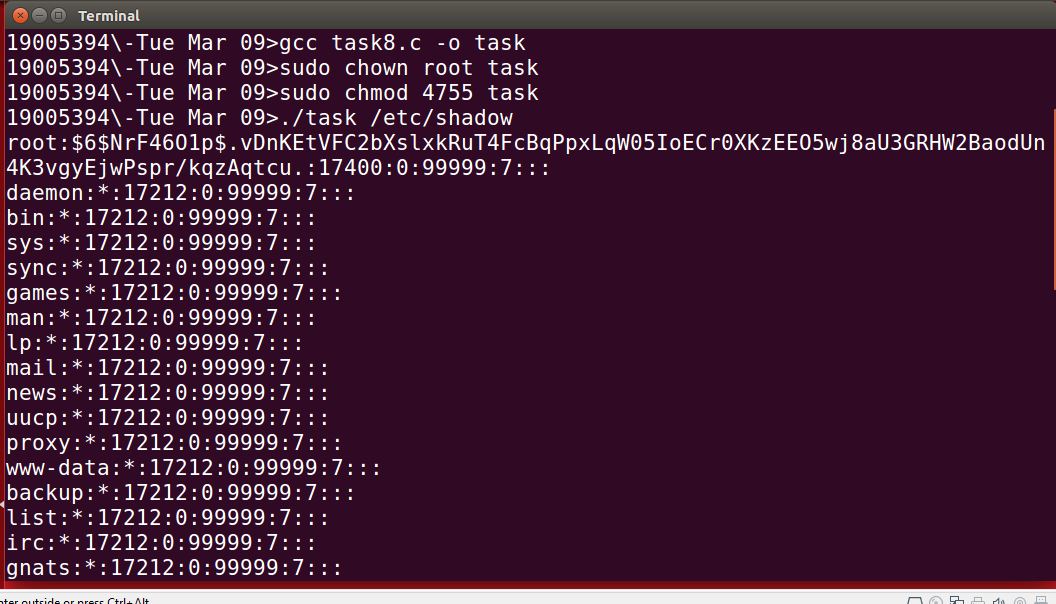


Figure 41

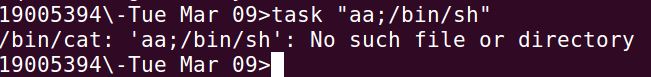


Figure 42

**Question:**

Exceve is a safe way so I could not get a root shell by it to attack the system. Because the function execve() does not look for any other commands. It just treats the input as a string of file names and attempts to locate it, rather than attempting to locate the command and then executing it with root system privileges like system() would do.

## Task 9: Capability Leaking

I had to add some important headers to the program to work, such like fork, setuid and getuid headers.

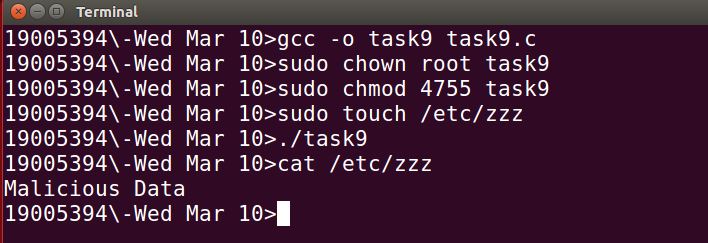


Figure 43

Yes, the file /etc/zzz has been modified, the particular reason for that circumstance is the file /etc/zzz has opened before the set uid because of the O\_RDWR privilege that /etc/zzz file has. **(setuid the program has opened the /etc/zzz file with O\_RDWR privilege)**

## Task 10: Lab Clean-up

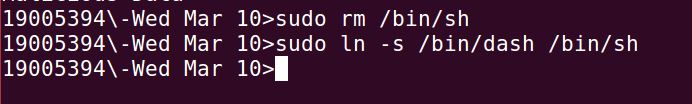
****

Figure 44