PES UNIVERSITY

UE19CS346
Information Security

Lab - 01
Environment Variable and Set-UID Program Lab

Name : Suhan B Revankar
SRN : PES2UG19CS412
Section : G Section

Environment Variable and Set-UID Program Lab

In this lab, students will understand

- How environment variables work
- How they are propagated from parent process to child
- How they affect system/program behavior

This lab is particularly oriented in how environment variables affect the behavior of Set-UID programs, which are usually privileged programs.

Table of Contents

Overview

On September 24, 2014, a severe vulnerability in Bash was identified. Nicknamed Shellshock, this vulnerability can exploit many systems and be launched either remotely or from a local machine. In this lab, students need to work on this attack, so they can understand the Shellshock vulnerability. The learning objective of this lab is for students to get first-hand experience on this interesting attack, understand how it works, and think about the lessons that we can get out of this attack. This lab covers the following topics:

- Shellshock
- Environment variables
- Function definition in Bash
- Apache and CGI programs

Lab environment. This lab has been tested on our pre-built Ubuntu 16.04 VM, which can be downloaded from the SEED website. https://seedsecuritylabs.org/lab_env.html. Download the June 2019 version of ubuntu 16.04

Lab Tasks

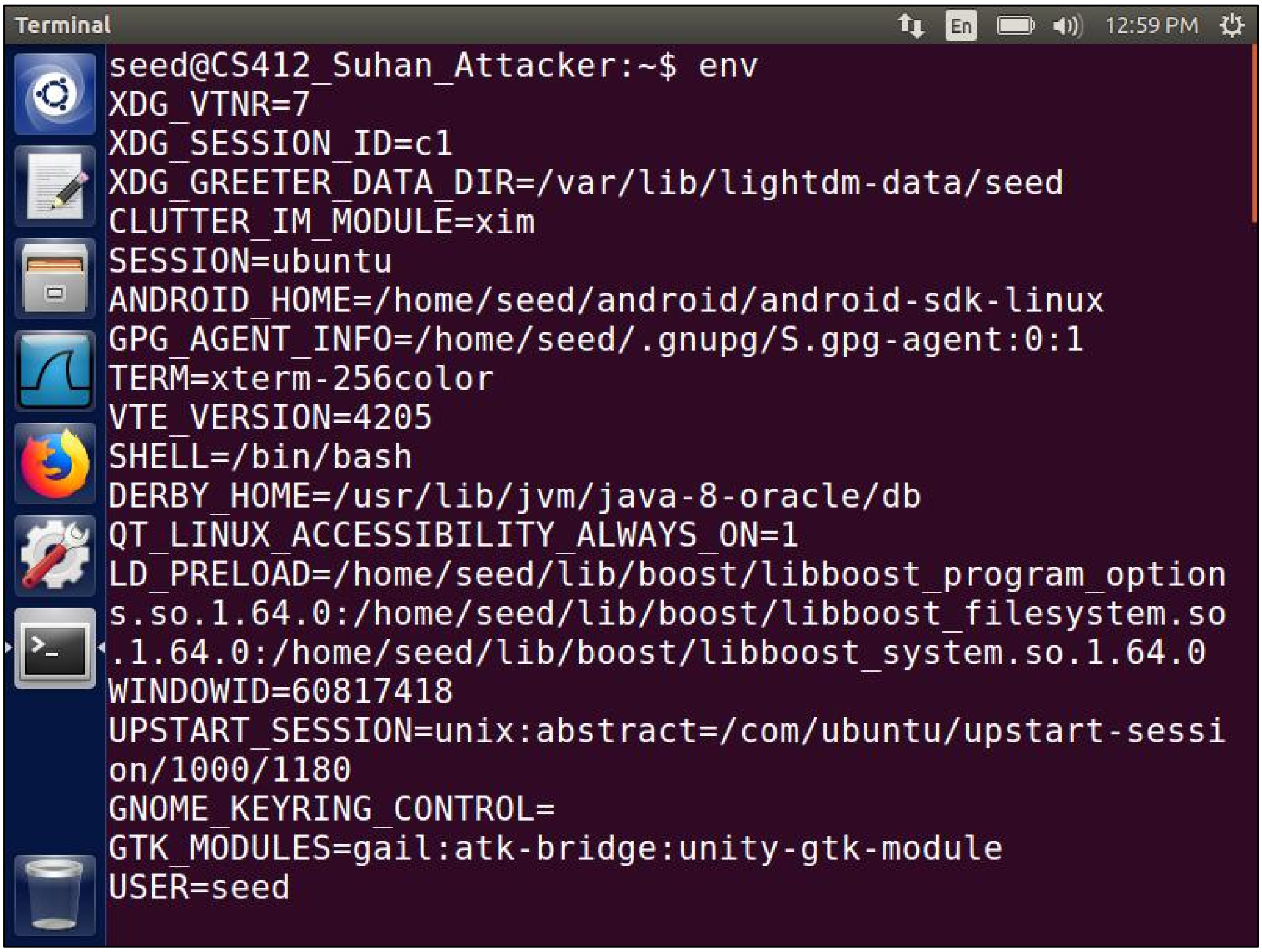
Task 1: Manipulating environment variables

In this task, Study the commands that can be used to set and unset environment variables. Here using Bash in the seed account. The default shell that a user uses is set in the /etc/passwd file (the last field of each entry). You can change this to another shell program using the command chsh (please do not do it for this lab). Please do the following tasks:

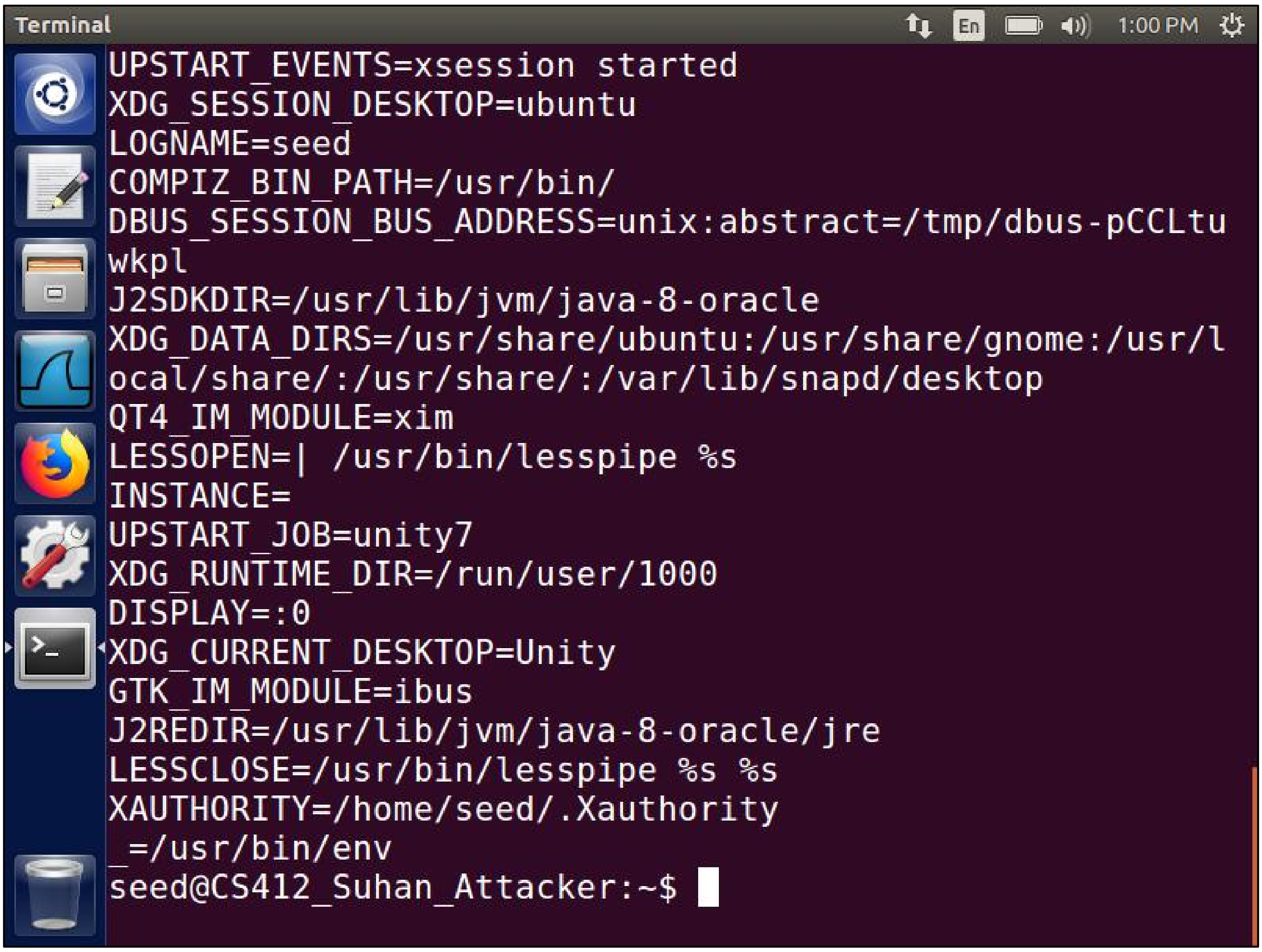
Use printenv or env command to print out the environment variables. If you are interested in some particular environment variables, such as PWD, you can use

Command:

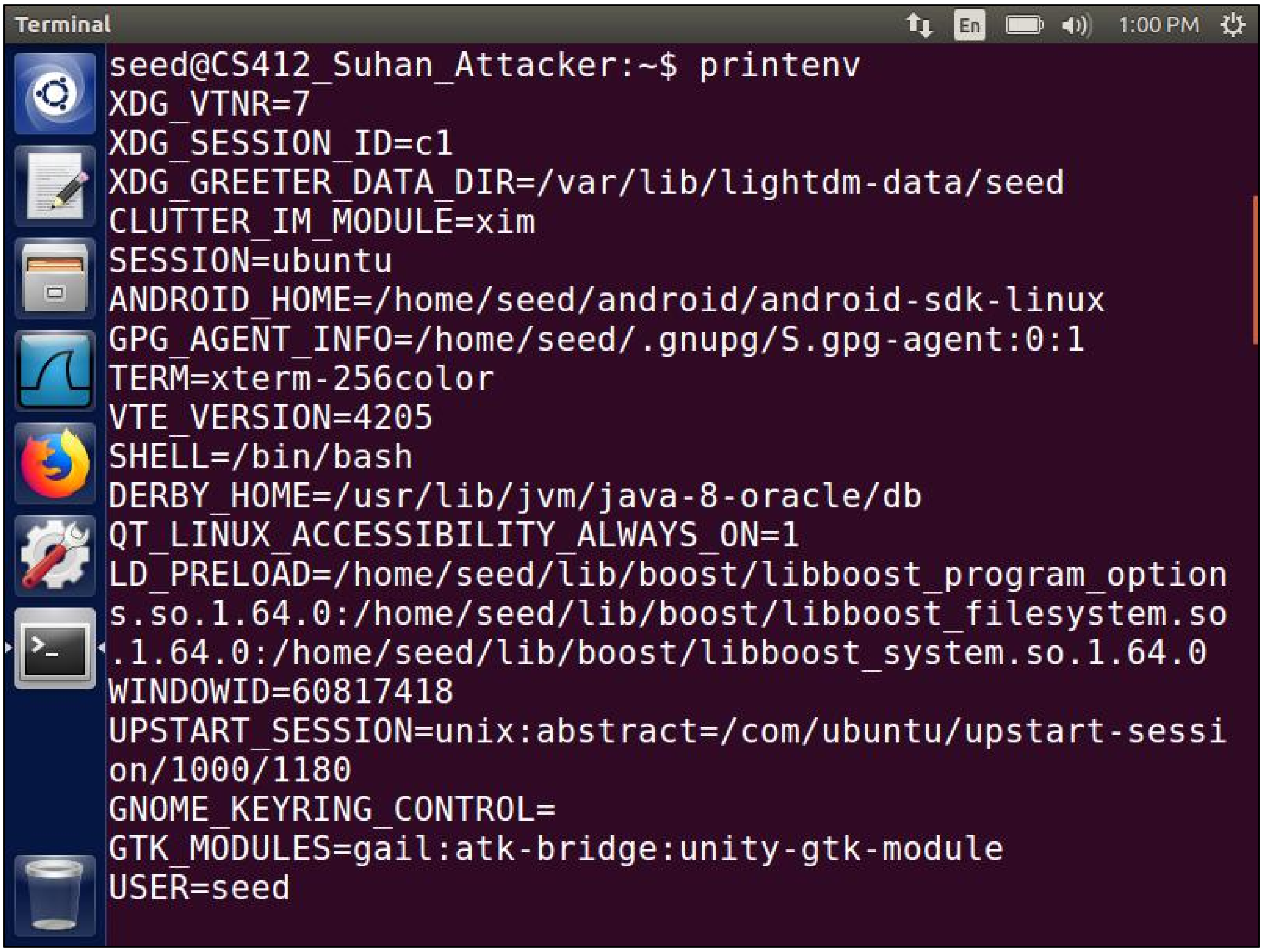
```
$ printenv PWD  (or)
$ env | grep PWD
```



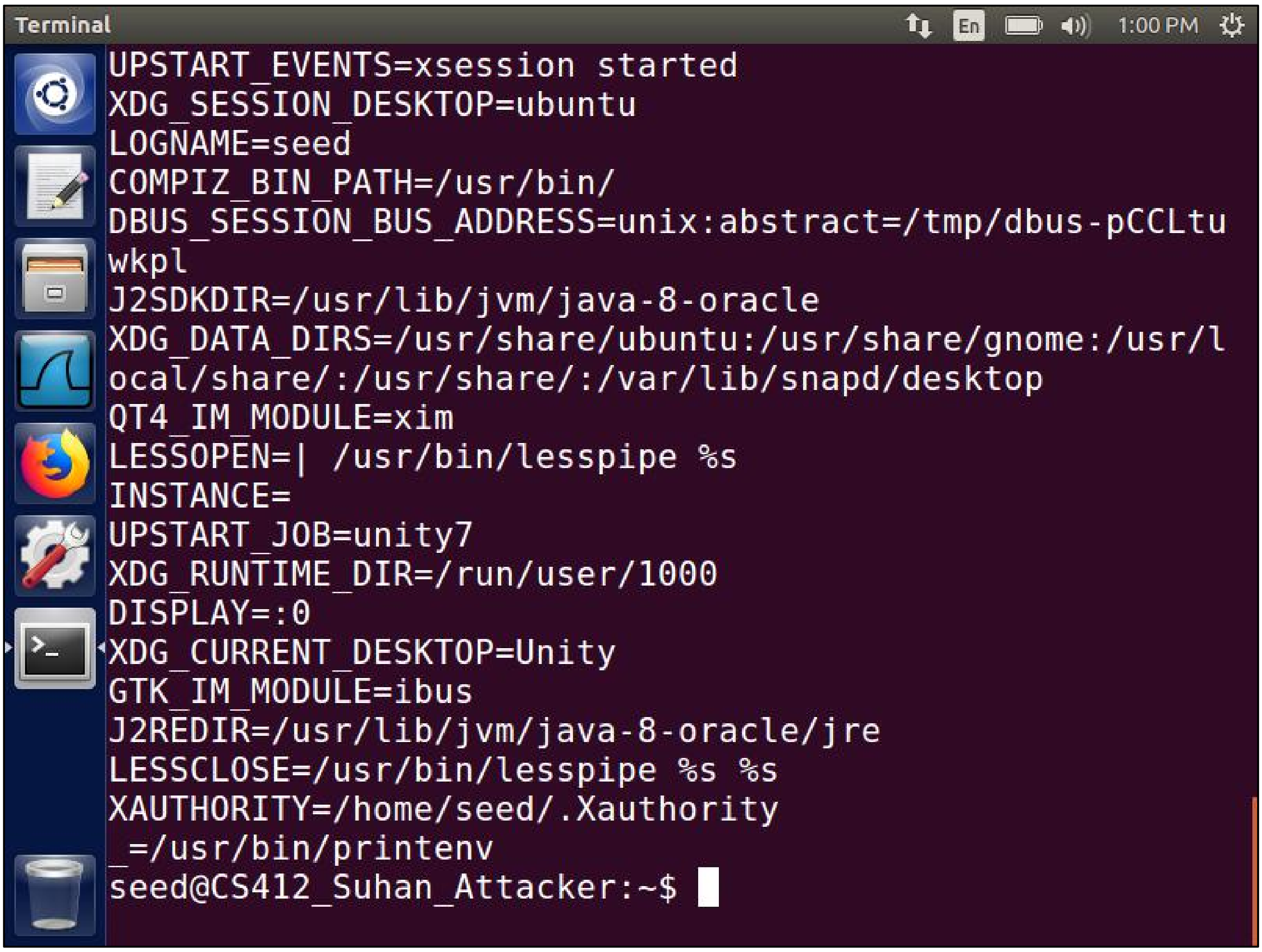
```
seed@CS412_Suhan_Attacker:~$ env
XDG_VTNR=7
XDG_SESSION_ID=c1
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed
CLUTTER_IM_MODULE=xim
SESSION=ubuntu
ANDROID_HOME=/home/seed/android/android-sdk-linux
GPG_AGENT_INFO=/home/seed/.gnupg/S.gpg-agent:0:1
TERM=xterm-256color
VTE_VERSION=4205
SHELL=/bin/bash
DERBY_HOME=/usr/lib/jvm/java-8-oracle/db
QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
LD_PRELOAD=/home/seed/lib/boost/libboost_program_option
s.so.1.64.0:/home/seed/lib/boost/libboost_filesystem.so
.1.64.0:/home/seed/lib/boost/libboost_system.so.1.64.0
WINDOWID=60817418
UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-sessi
on/1000/1180
GNOME_KEYRING_CONTROL=
GTK_MODULES=gail:atk-bridge:unity-gtk-module
USER=seed
```



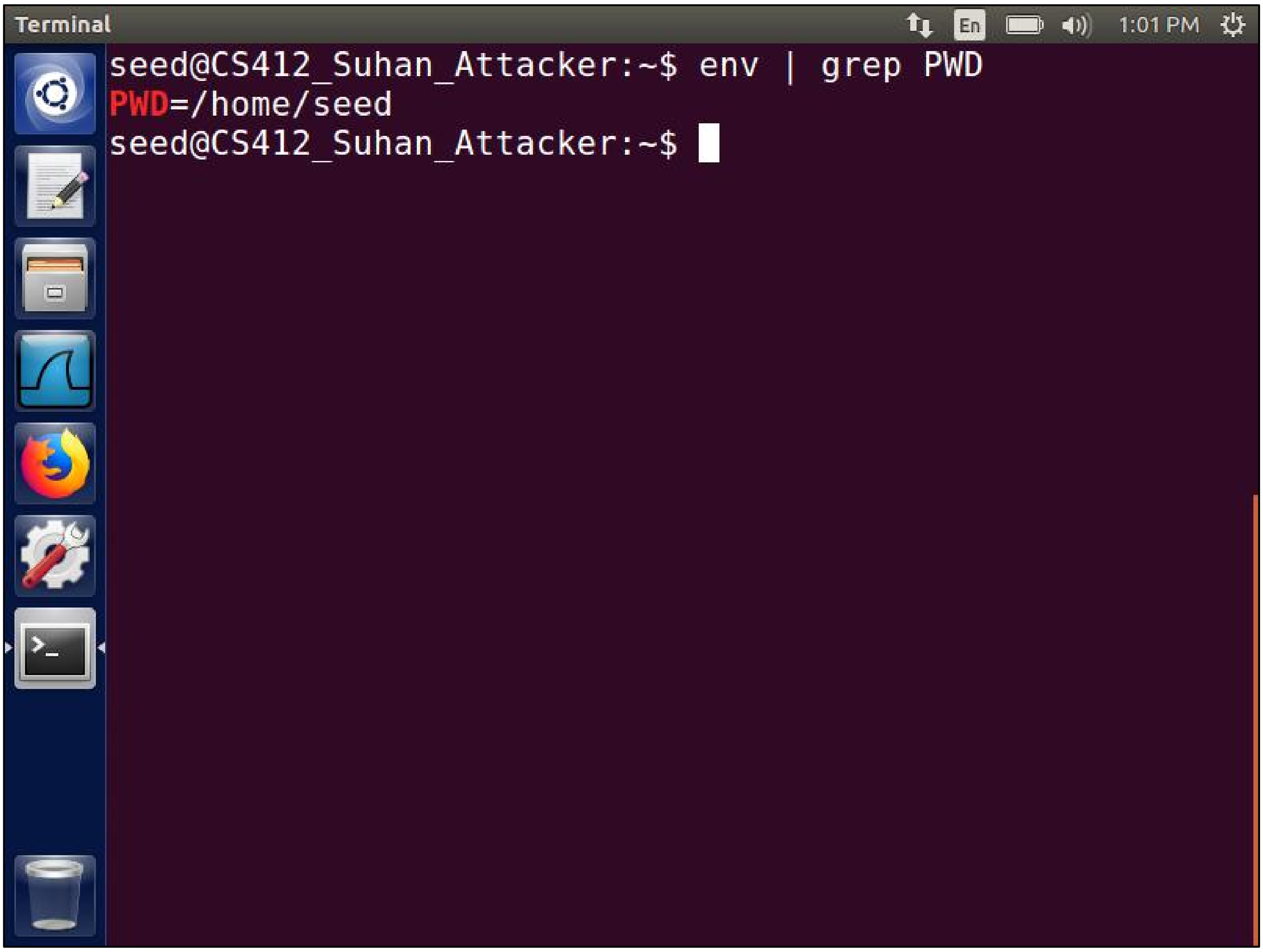
```
UPSTART_EVENTS=xsession started
XDG_SESSION_DESKTOP=ubuntu
LOGNAME=seed
COMPIZ_BIN_PATH=/usr/bin/
DBUS_SESSION_BUS_ADDRESS=unix:abstract=/tmp/dbus-pCCLtu
wkpl
J2SDKDIR=/usr/lib/jvm/java-8-oracle
XDG_DATA_DIRS=/usr/share/ubuntu:/usr/share/gnome:/usr/l
ocal/share/:/usr/share/:/var/lib/snapd/desktop
QT4_IM_MODULE=xim
LESSOPEN=| /usr/bin/lesspipe %s
INSTANCE=
UPSTART_JOB=unity7
XDG_RUNTIME_DIR=/run/user/1000
DISPLAY=:0
XDG_CURRENT_DESKTOP=Unity
GTK_IM_MODULE=ibus
J2REDIR=/usr/lib/jvm/java-8-oracle/jre
LESSCLOSE=/usr/bin/lesspipe %s %s
XAUTHORITY=/home/seed/.Xauthority
_=/usr/bin/env
seed@CS412_Suhan_Attacker:~$ 
```



```
seed@CS412_Suhan_Attacker:~$ printenv
XDG_VTNR=7
XDG_SESSION_ID=c1
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed
CLUTTER_IM_MODULE=xim
SESSION=ubuntu
ANDROID_HOME=/home/seed/android/android-sdk-linux
GPG_AGENT_INFO=/home/seed/.gnupg/S.gpg-agent:0:1
TERM=xterm-256color
VTE_VERSION=4205
SHELL=/bin/bash
DERBY_HOME=/usr/lib/jvm/java-8-oracle/db
QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
LD_PRELOAD=/home/seed/lib/boost/libboost_program_option
s.so.1.64.0:/home/seed/lib/boost/libboost_filesystem.so
.1.64.0:/home/seed/lib/boost/libboost_system.so.1.64.0
WINDOWID=60817418
UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-sessi
on/1000/1180
GNOME_KEYRING_CONTROL=
GTK_MODULES=gail:atk-bridge:unity-gtk-module
USER=seed
```



```
UPSTART_EVENTS=xsession started
XDG_SESSION_DESKTOP=ubuntu
LOGNAME=seed
COMPIZ_BIN_PATH=/usr/bin/
DBUS_SESSION_BUS_ADDRESS=unix:abstract=/tmp/dbus-pCCLtu
wkpl
J2SDKDIR=/usr/lib/jvm/java-8-oracle
XDG_DATA_DIRS=/usr/share/ubuntu:/usr/share/gnome:/usr/l
ocal/share/:/usr/share/:/var/lib/snapd/desktop
QT4_IM_MODULE=xim
LESSOPEN=| /usr/bin/lesspipe %s
INSTANCE=
UPSTART_JOB=unity7
XDG_RUNTIME_DIR=/run/user/1000
DISPLAY=:0
XDG_CURRENT_DESKTOP=Unity
GTK_IM_MODULE=ibus
J2REDIR=/usr/lib/jvm/java-8-oracle/jre
LESSCLOSE=/usr/bin/lesspipe %s %s
XAUTHORITY=/home/seed/.Xauthority
_=/usr/bin/printenv
seed@CS412_Suhan_Attacker:~$
```

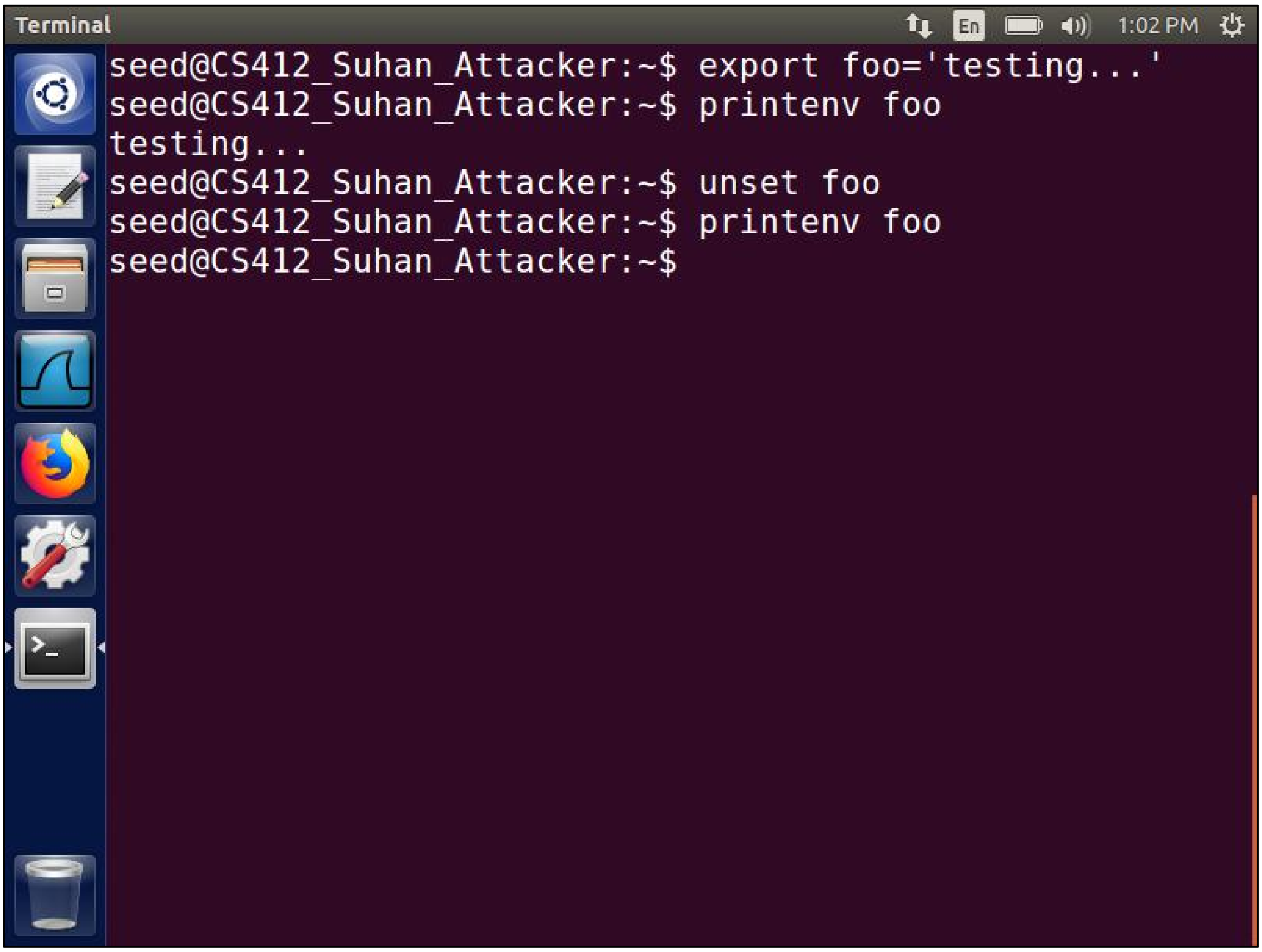


```
seed@CS412_Suhan_Attacker:~$ env | grep PWD
PWD=/home/seed
seed@CS412_Suhan_Attacker:~$
```

- Use export and unset to set environment variables. It should be noted that these two commands are not separate programs; they are two of Bash's internal commands (you will not be able to find them outside of Bash). Use unset to unset the variable

Command:

```
$export foo=’test string’
$printenv foo

$ unset foo
$ printenv foo
```

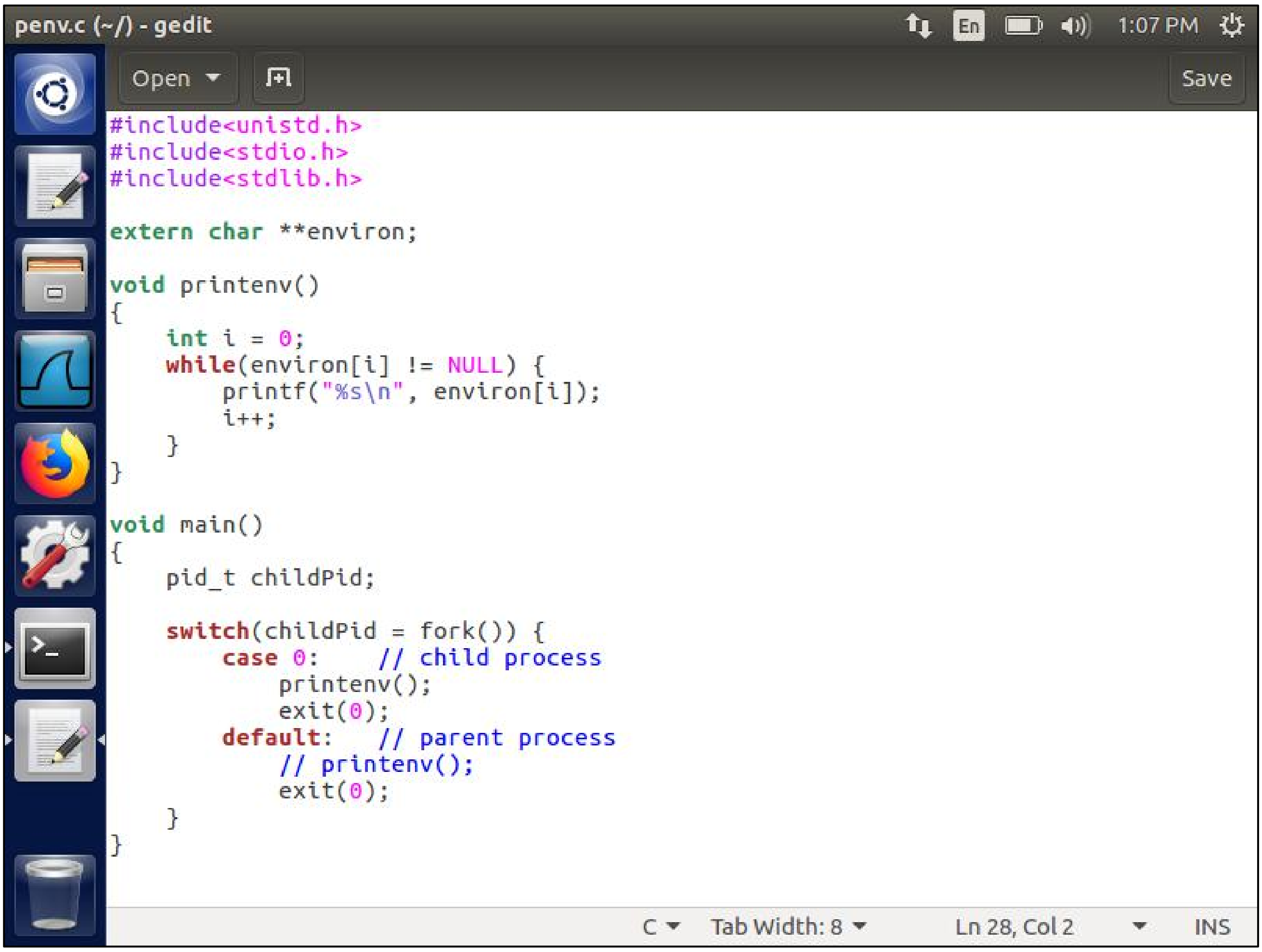


```
seed@CS412_Suhan_Attacker:~$ export foo='testing...'
seed@CS412_Suhan_Attacker:~$ printenv foo
testing...
seed@CS412_Suhan_Attacker:~$ unset foo
seed@CS412_Suhan_Attacker:~$ printenv foo
seed@CS412_Suhan_Attacker:~$
```

Task 2: Inheriting environment variables from parents

In this task, Study how environment variables are inherited by child processes from their parents. In Unix, fork() creates a new process by duplicating the calling process. The new process, referred to as the child, is an exact duplicate of the calling process, referred to as the parent; however, several things are not inherited by the child (please see the manual of fork() by typing the following command: man fork).

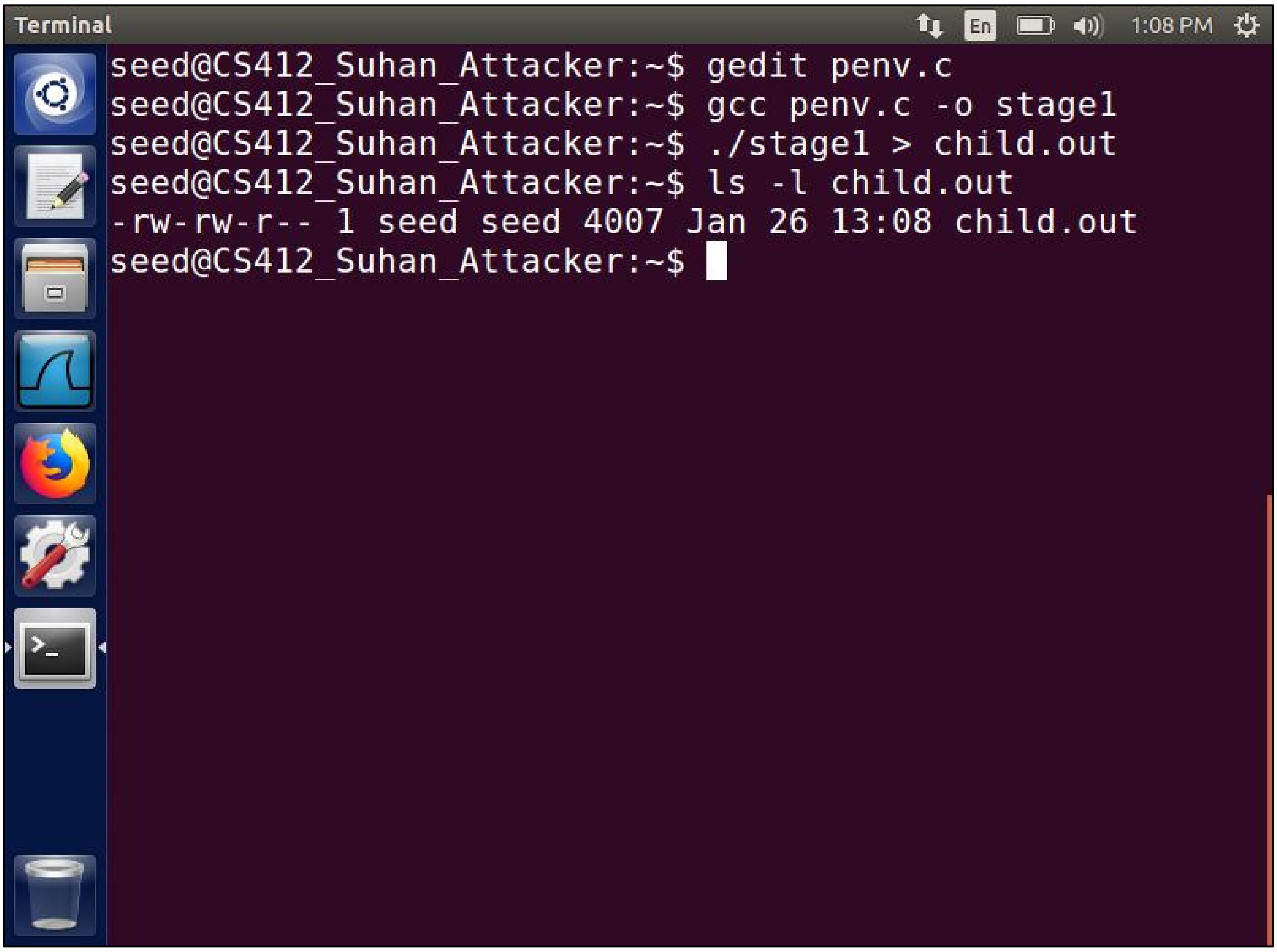
Step 1: Please compile and run the following program, and describe your observation. Because the output contains many strings, you should save the output into a file, such as using a.out > child (assuming that a.out is your executable file name).



```c
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>

extern char **environ;

void printenv()
{
    int i = 0;
    while(environ[i] != NULL) {
        printf("%s\n", environ[i]);
        i++;
    }
}

void main()
{
    pid_t childPid;

    switch(childPid = fork()) {
        case 0:    // child process
            printenv();
            exit(0);
        default:   // parent process
            // printenv();
            exit(0);
    }
}
```

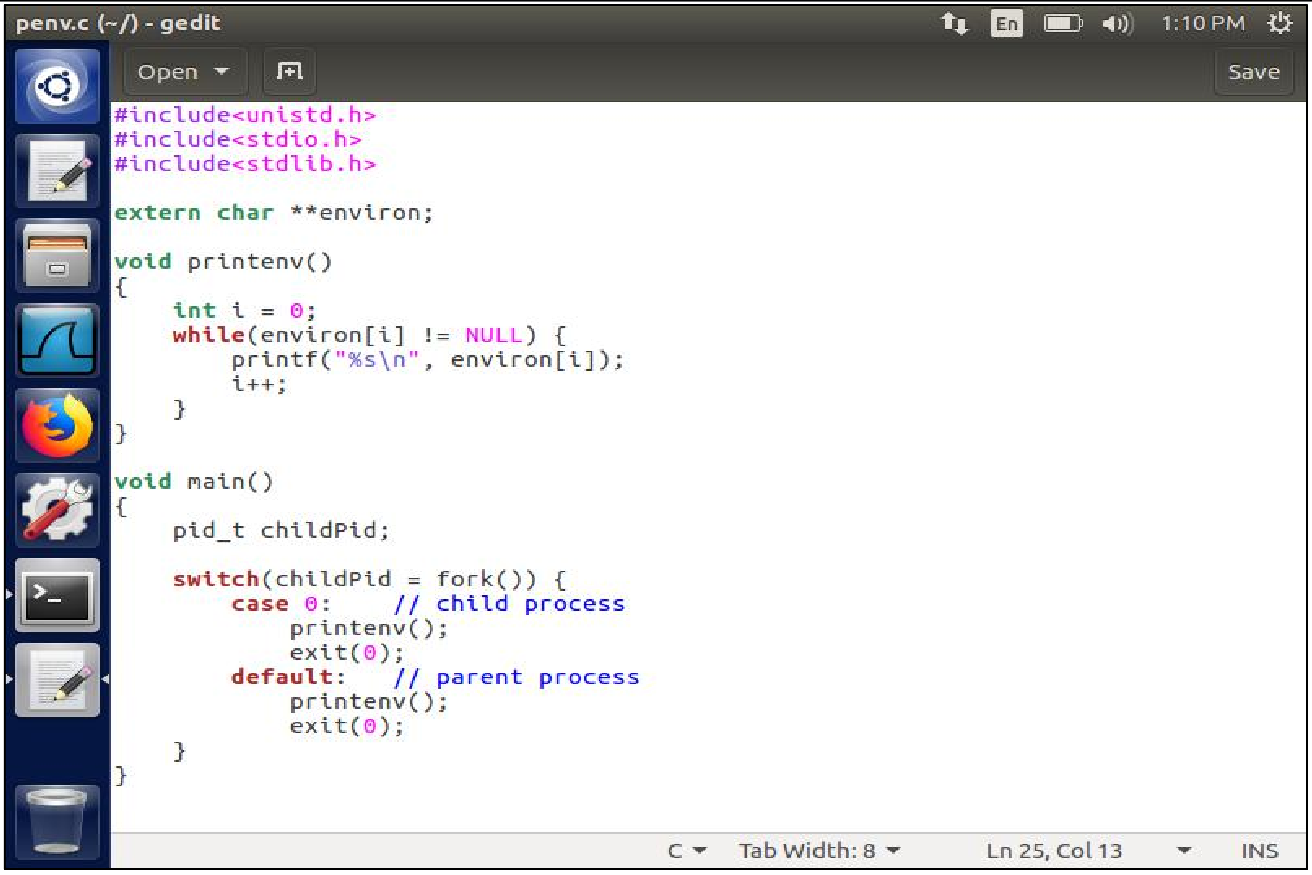
Commands:

```
$gcc penv.c
$a.out>child
$ls -l child
```



```
seed@CS412_Suhan_Attacker:~$ gedit penv.c
seed@CS412_Suhan_Attacker:~$ gcc penv.c -o stage1
seed@CS412_Suhan_Attacker:~$ ./stage1 > child.out
seed@CS412_Suhan_Attacker:~$ ls -l child.out
-rw-rw-r-- 1 seed seed 4007 Jan 26 13:08 child.out
seed@CS412_Suhan_Attacker:~$ 
```

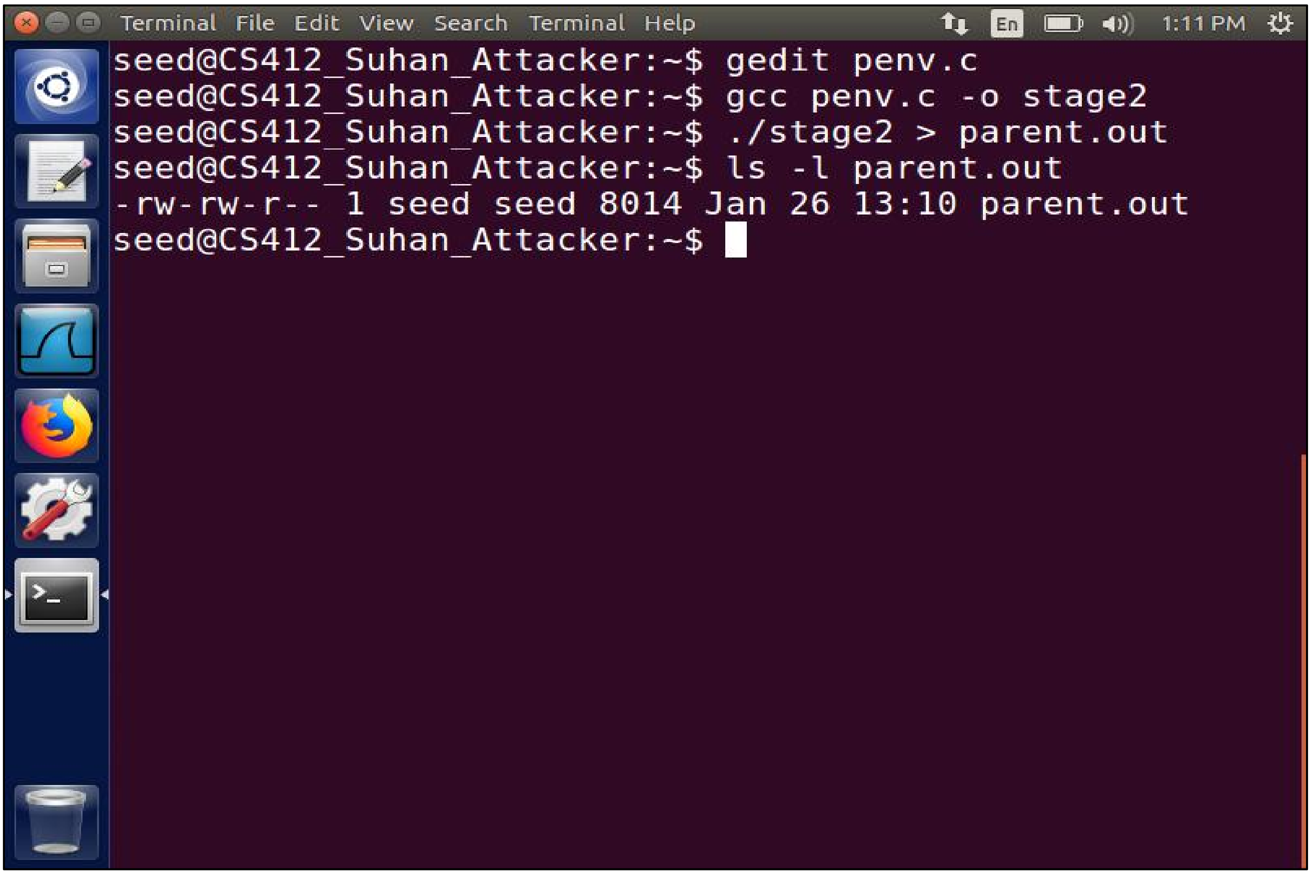
Step 2: Now comment out the printenv() statement in the child process case, and uncomment the printenv() statement in the parent process case. Compile and run the code, and describe your observation. Save the output in another file.



```c
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>

extern char **environ;

void printenv()
{
    int i = 0;
    while(environ[i] != NULL) {
        printf("%s\n", environ[i]);
        i++;
    }
}

void main()
{
    pid_t childPid;

    switch(childPid = fork()) {
        case 0:     // child process
            printenv();
            exit(0);
        default:    // parent process
            printenv();
            exit(0);
    }
}
```

Commands:

```
$ gcc penv.c
$a.out>parent
$ls -l parent
```

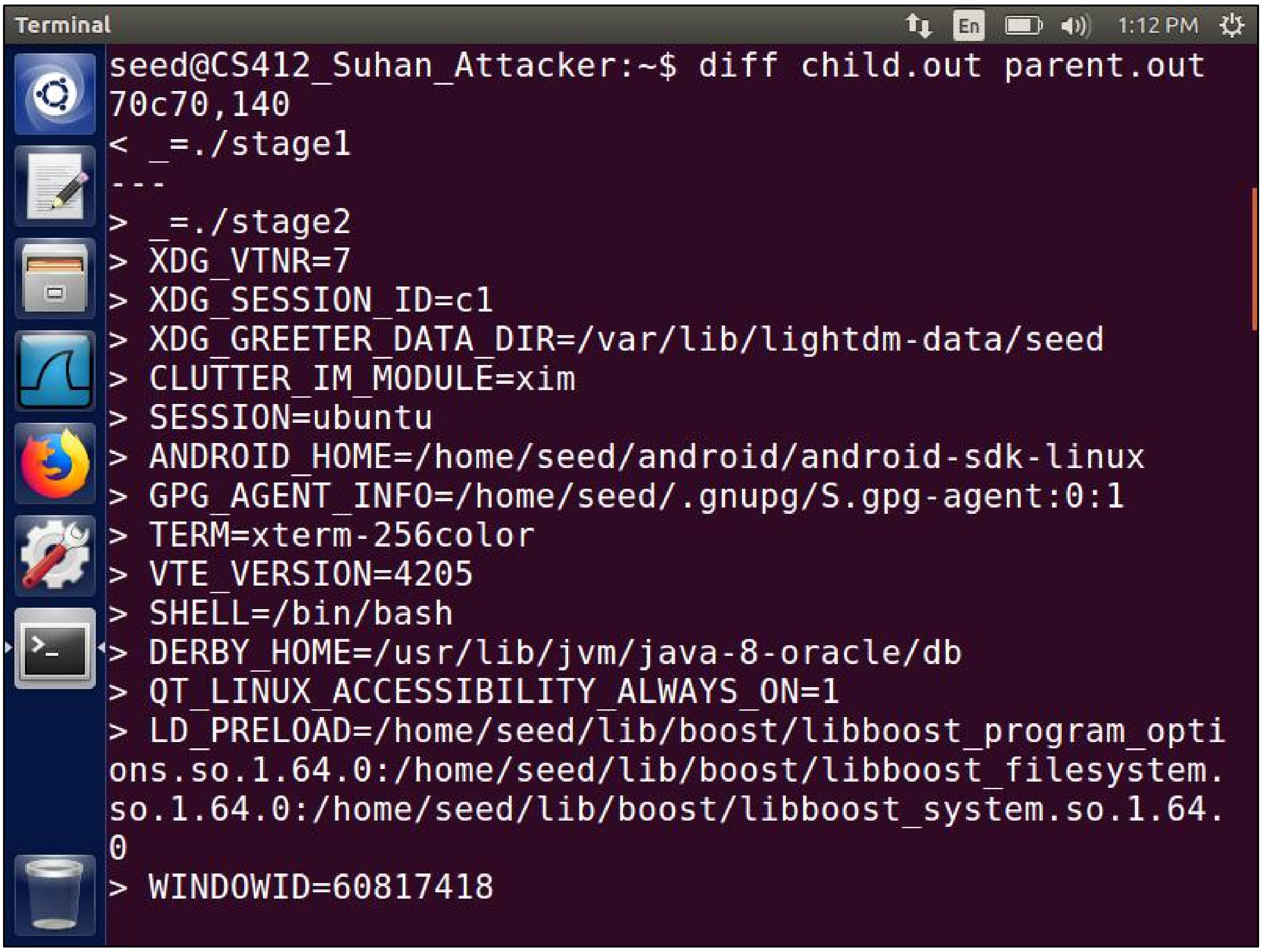


```
seed@CS412_Suhan_Attacker:~$ gedit penv.c
seed@CS412_Suhan_Attacker:~$ gcc penv.c -o stage2
seed@CS412_Suhan_Attacker:~$ ./stage2 > parent.out
seed@CS412_Suhan_Attacker:~$ ls -l parent.out
-rw-rw-r-- 1 seed seed 8014 Jan 26 13:10 parent.out
seed@CS412_Suhan_Attacker:~$
```

Step 3: Compare the difference between these two files using the diff command. Please draw your conclusion.

Command:

$ diff child parent



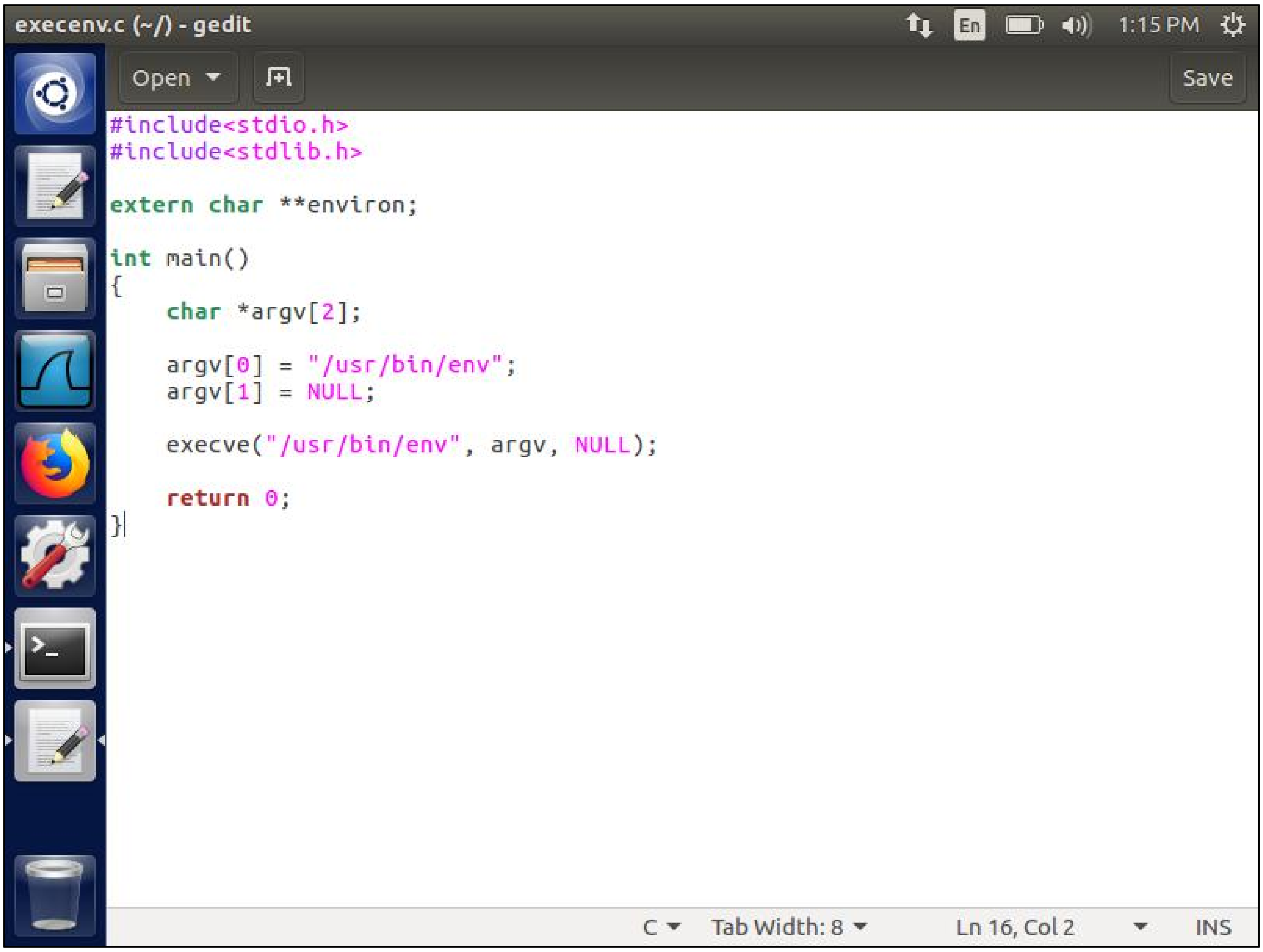
```
seed@CS412_Suhan_Attacker:~$ diff child.out parent.out
70c70,140
< _=./stage1
---
> _=./stage2
> XDG_VTNR=7
> XDG_SESSION_ID=c1
> XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed
> CLUTTER_IM_MODULE=xim
> SESSION=ubuntu
> ANDROID_HOME=/home/seed/android/android-sdk-linux
> GPG_AGENT_INFO=/home/seed/.gnupg/S.gpg-agent:0:1
> TERM=xterm-256color
> VTE_VERSION=4205
> SHELL=/bin/bash
> DERBY_HOME=/usr/lib/jvm/java-8-oracle/db
> QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
> LD_PRELOAD=/home/seed/lib/boost/libboost_program_opti
ons.so.1.64.0:/home/seed/lib/boost/libboost_filesystem.
so.1.64.0:/home/seed/lib/boost/libboost_system.so.1.64.
0
> WINDOWID=60817418
```

Task 3: Environment variables and execve()

In this task, Study how environment

variables are affected when a new program is executed via execve(). The function execve() calls a system call to load a new command and execute it; this function never returns. No new process is created; instead, the calling process's text, data, bss, and stack are overwritten by that of the program loaded. Essentially, execve() runs the new program inside the calling process. Here our interest is what happens to the environment variables; are they automatically inherited by the new program?

Step 1: Please compile and run the following program, and describe your observation. This program simply executes a program called /usr/bin/env, which prints out the environment variables of the current process.

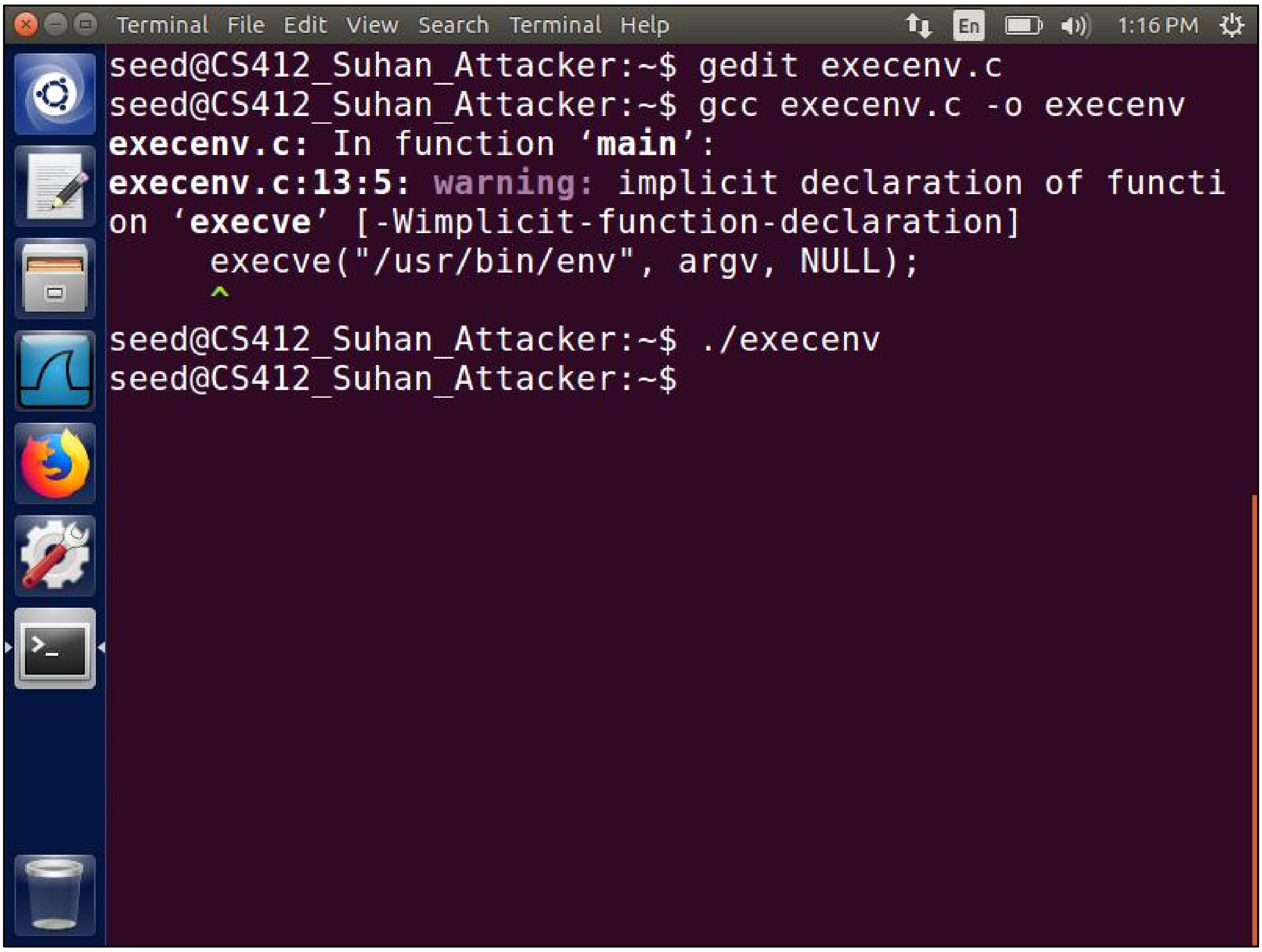


```
#include<stdio.h>
#include<stdlib.h>

extern char **environ;

int main()
{
	char *argv[2];

	argv[0] = "/usr/bin/env";
	argv[1] = NULL;

	execve("/usr/bin/env", argv, NULL);

	return 0;
}
```

Commands:

$gcc execenv.c -o execenv

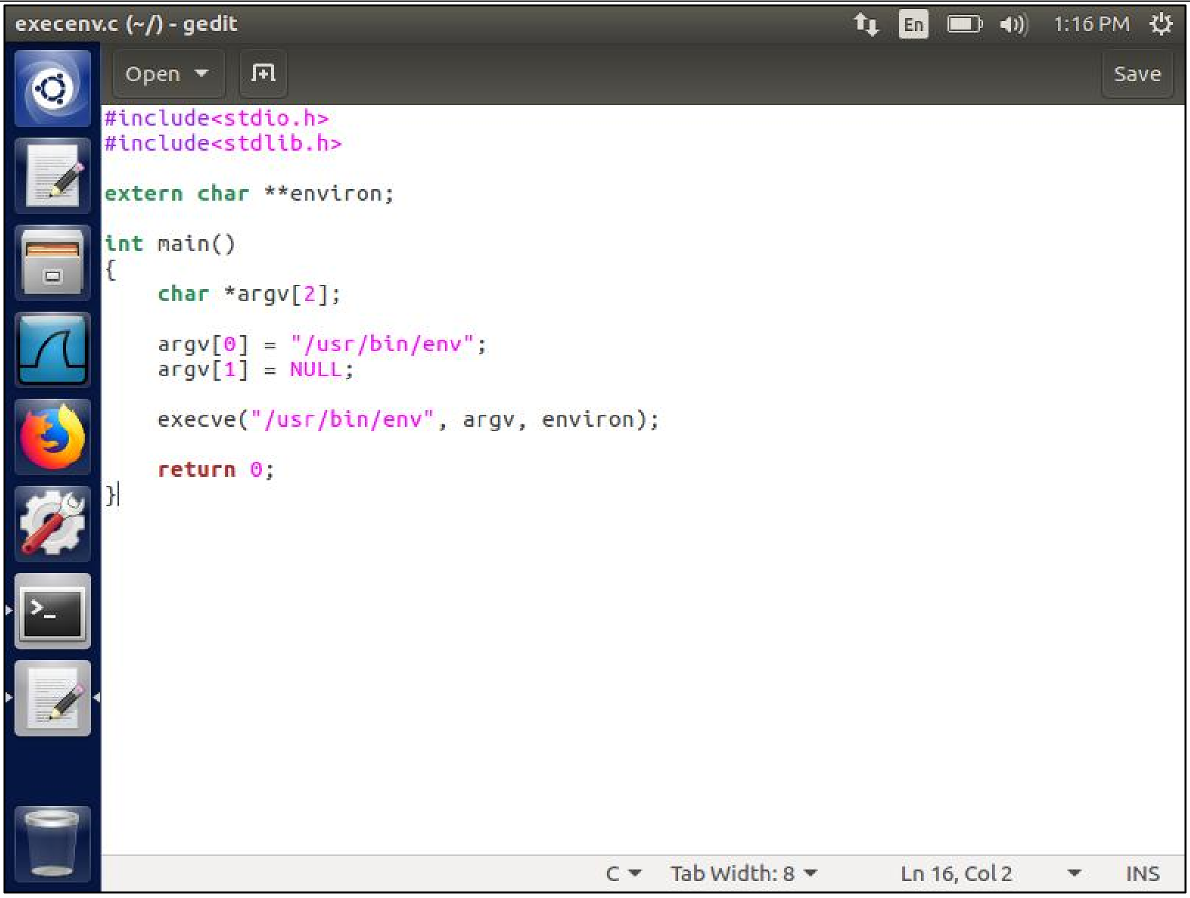
$./execenv



```
seed@CS412_Suhan_Attacker:~$ gedit execenv.c
seed@CS412_Suhan_Attacker:~$ gcc execenv.c -o execenv
execenv.c: In function ‘main’:
execenv.c:13:5: warning: implicit declaration of functi
on ‘execve’ [-Wimplicit-function-declaration]
     execve("/usr/bin/env", argv, NULL);
     ^
seed@CS412_Suhan_Attacker:~$ ./execenv
seed@CS412_Suhan_Attacker:~$
```

Step 2: Now, change the invocation of execve() to the following, and describe your observation. (make changes in the program given above)

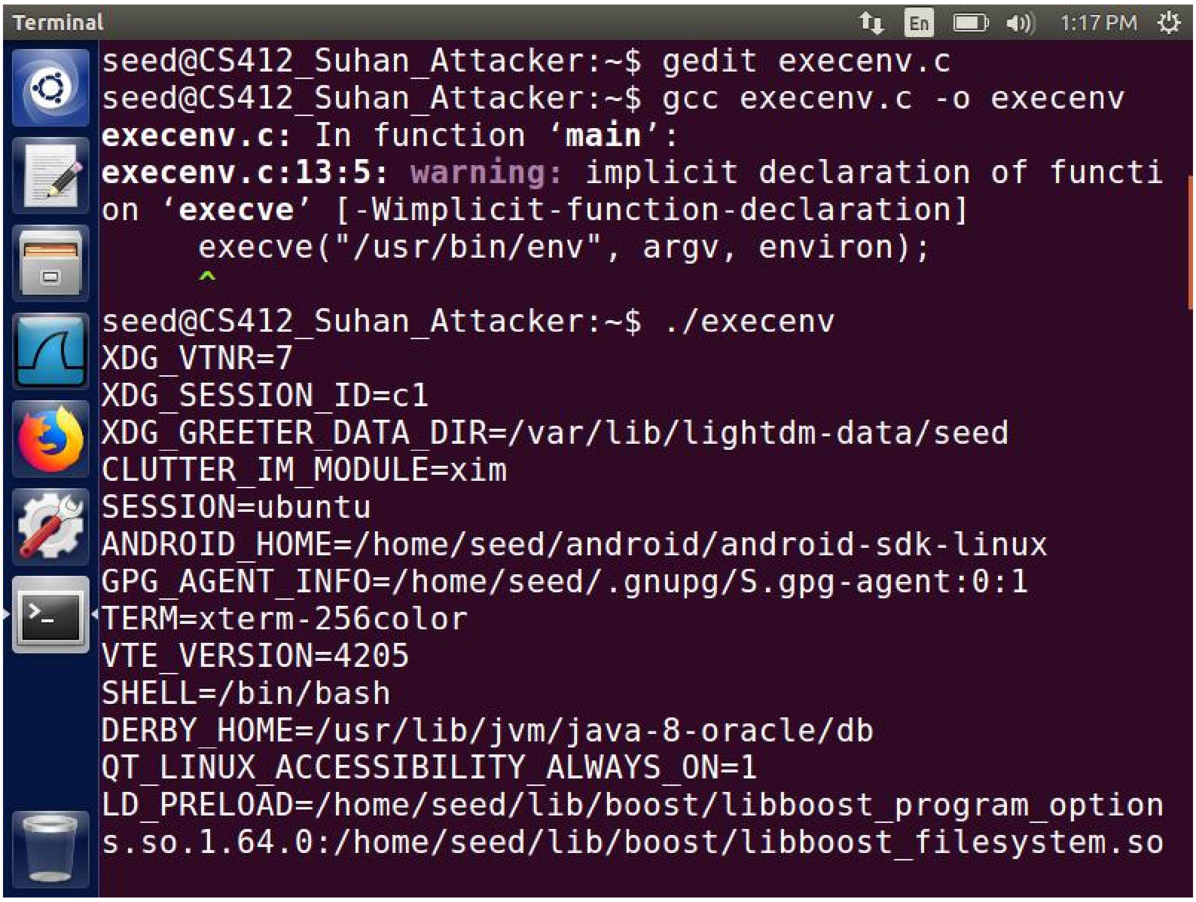
execve("/usr/bin/env", argv, environ);



```
#include<stdio.h>
#include<stdlib.h>

extern char **environ;

int main()
{
	char *argv[2];

	argv[0] = "/usr/bin/env";
	argv[1] = NULL;

	execve("/usr/bin/env", argv, environ);

	return 0;
}
```

Commands:

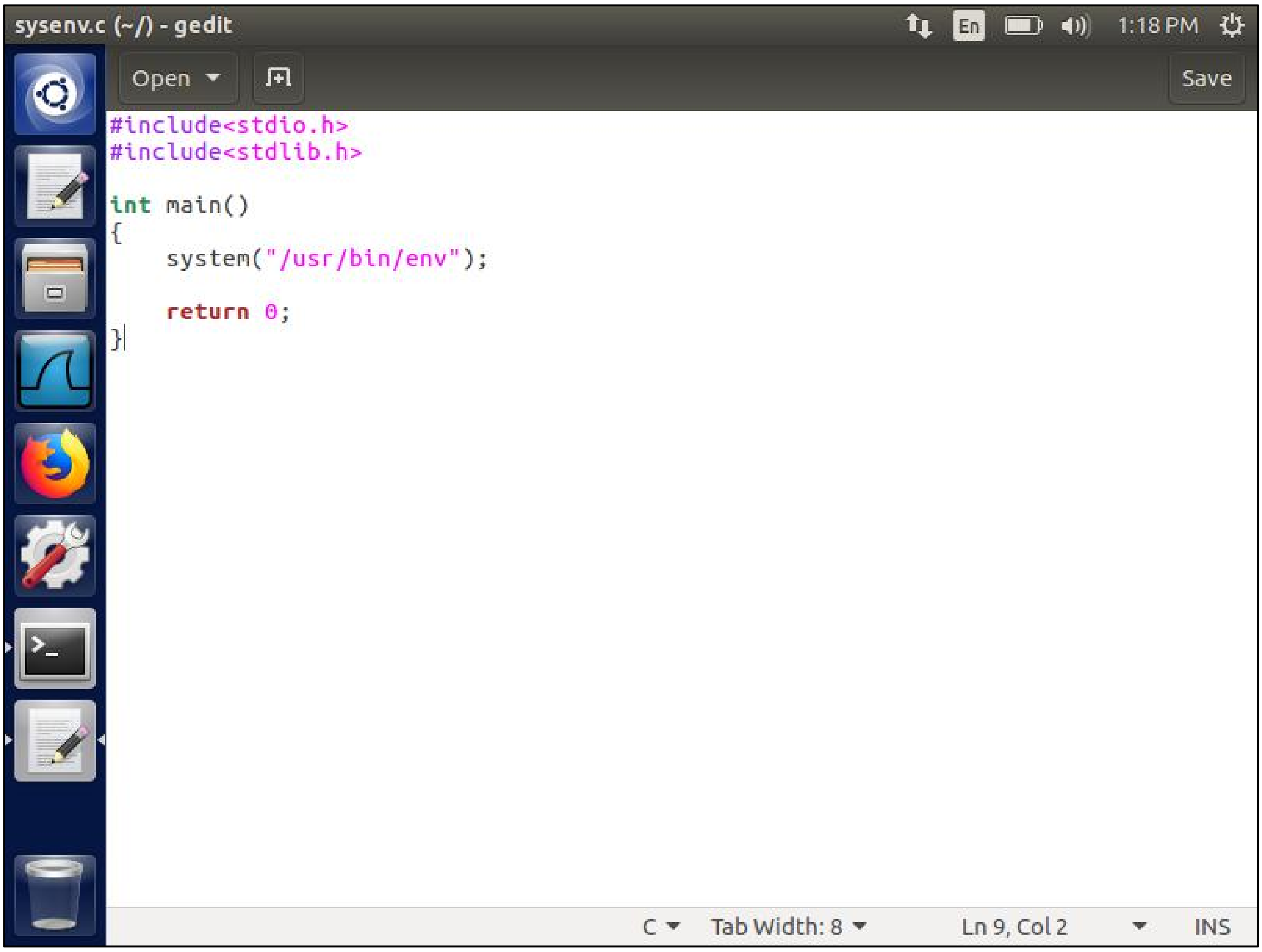
```
$gcc execenv.c -o execenv
$./execenv
```



```
seed@CS412_Suhan_Attacker:~$ gedit execenv.c
seed@CS412_Suhan_Attacker:~$ gcc execenv.c -o execenv
execenv.c: In function ‘main’:
execenv.c:13:5: warning: implicit declaration of functi
on ‘execve’ [-Wimplicit-function-declaration]
     execve("/usr/bin/env", argv, environ);
     ^
seed@CS412_Suhan_Attacker:~$ ./execenv
XDG_VTNR=7
XDG_SESSION_ID=c1
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed
CLUTTER_IM_MODULE=xim
SESSION=ubuntu
ANDROID_HOME=/home/seed/android/android-sdk-linux
GPG_AGENT_INFO=/home/seed/.gnupg/S.gpg-agent:0:1
TERM=xterm-256color
VTE_VERSION=4205
SHELL=/bin/bash
DERBY_HOME=/usr/lib/jvm/java-8-oracle/db
QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
LD_PRELOAD=/home/seed/lib/boost/libboost_program_option
s.so.1.64.0:/home/seed/lib/boost/libboost_filesystem.so
```

Task 4: Environment variables and system()

In this task, Study how environment variables are affected when a new program is executed via the system() function. This function is used to execute a command, but unlike execve(), which directly executes a command, system() actually executes "/bin/sh -c command", i.e., it executes /bin/sh, and asks the shell to execute the command. If you look at the implementation of the system() function, you will see that it uses execl() to execute /bin/sh; excel() calls execve(), passing to it the environment variables array. Therefore using system(), the environment variables of the calling process are passed to the new program /bin/sh. Please compile and run the following program to verify this.

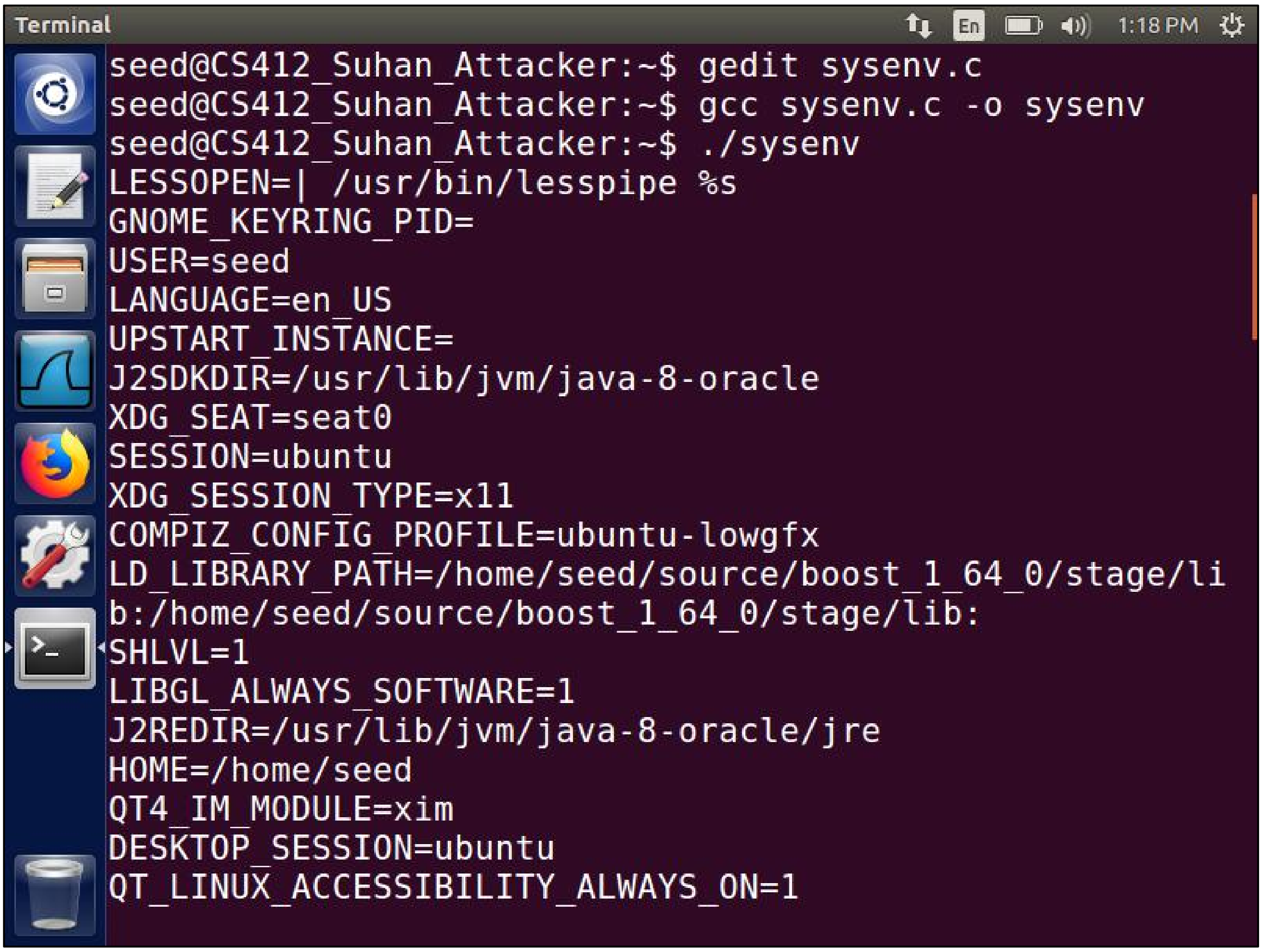


```
#include<stdio.h>
#include<stdlib.h>

int main()
{
	system("/usr/bin/env");

	return 0;
}
```

Commands:

Cd Desktop/environ_set_uid/ use this location before executing prog

```
$gcc sysenv.c
-sysenv
$ ./sysenv
```

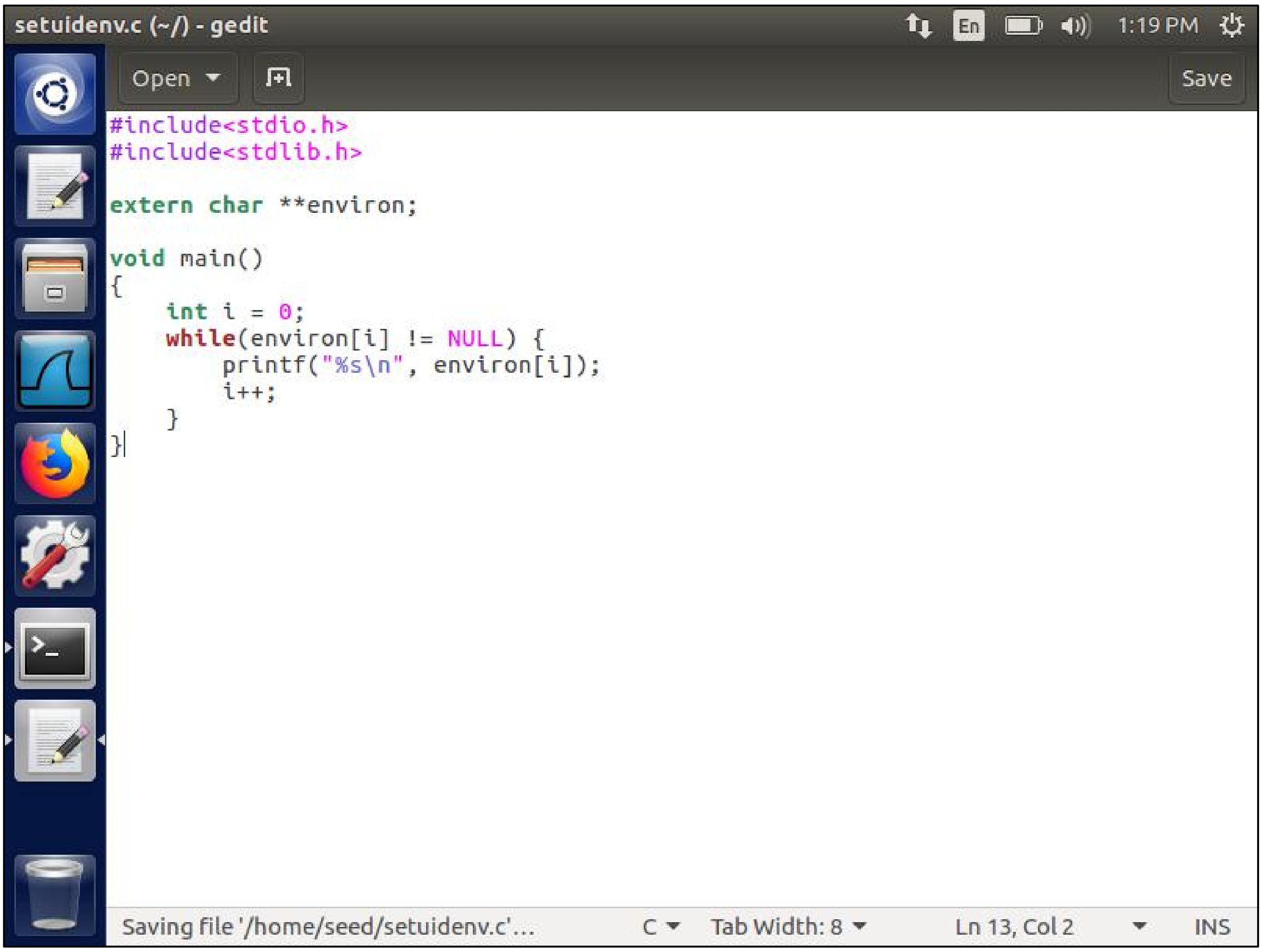


```
seed@CS412_Suhan_Attacker:~$ gedit sysenv.c
seed@CS412_Suhan_Attacker:~$ gcc sysenv.c -o sysenv
seed@CS412_Suhan_Attacker:~$ ./sysenv
LESSOPEN=| /usr/bin/lesspipe %s
GNOME_KEYRING_PID=
USER=seed
LANGUAGE=en_US
UPSTART_INSTANCE=
J2SDKDIR=/usr/lib/jvm/java-8-oracle
XDG_SEAT=seat0
SESSION=ubuntu
XDG_SESSION_TYPE=x11
COMPIZ_CONFIG_PROFILE=ubuntu-lowgfx
LD_LIBRARY_PATH=/home/seed/source/boost_1_64_0/stage/li
b:/home/seed/source/boost_1_64_0/stage/lib:
SHLVL=1
LIBGL_ALWAYS_SOFTWARE=1
J2REDIR=/usr/lib/jvm/java-8-oracle/jre
HOME=/home/seed
QT4_IM_MODULE=xim
DESKTOP_SESSION=ubuntu
QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
```

Task 5: Environment variable and Set-UID Programs

Set-UID is an important security mechanism in Unix operating systems. When a Set-UID program runs, it assumes the owner's privileges. For example, if the program's owner is root, then when anyone runs this program, the program gains the root's privileges during its execution. Set-UID allows us to do many interesting things, but it escalates the user's privilege when executed, making it quite risky. Although the behaviors of Set-UID programs are decided by their program logic, not by users, users can indeed affect the behaviors via environment variables. To understand how Set-UID programs are affected, let us first figure out whether environment variables are inherited by the Set-UID program's process from the user's process.

Step 1: We are going to write a program that can print out all the environment variables in the current process.



```
#include<stdio.h>
#include<stdlib.h>

extern char **environ;

void main()
{
        int i = 0;
        while(environ[i] != NULL) {
                printf("%s\n", environ[i]);
                i++;
        }
}
```

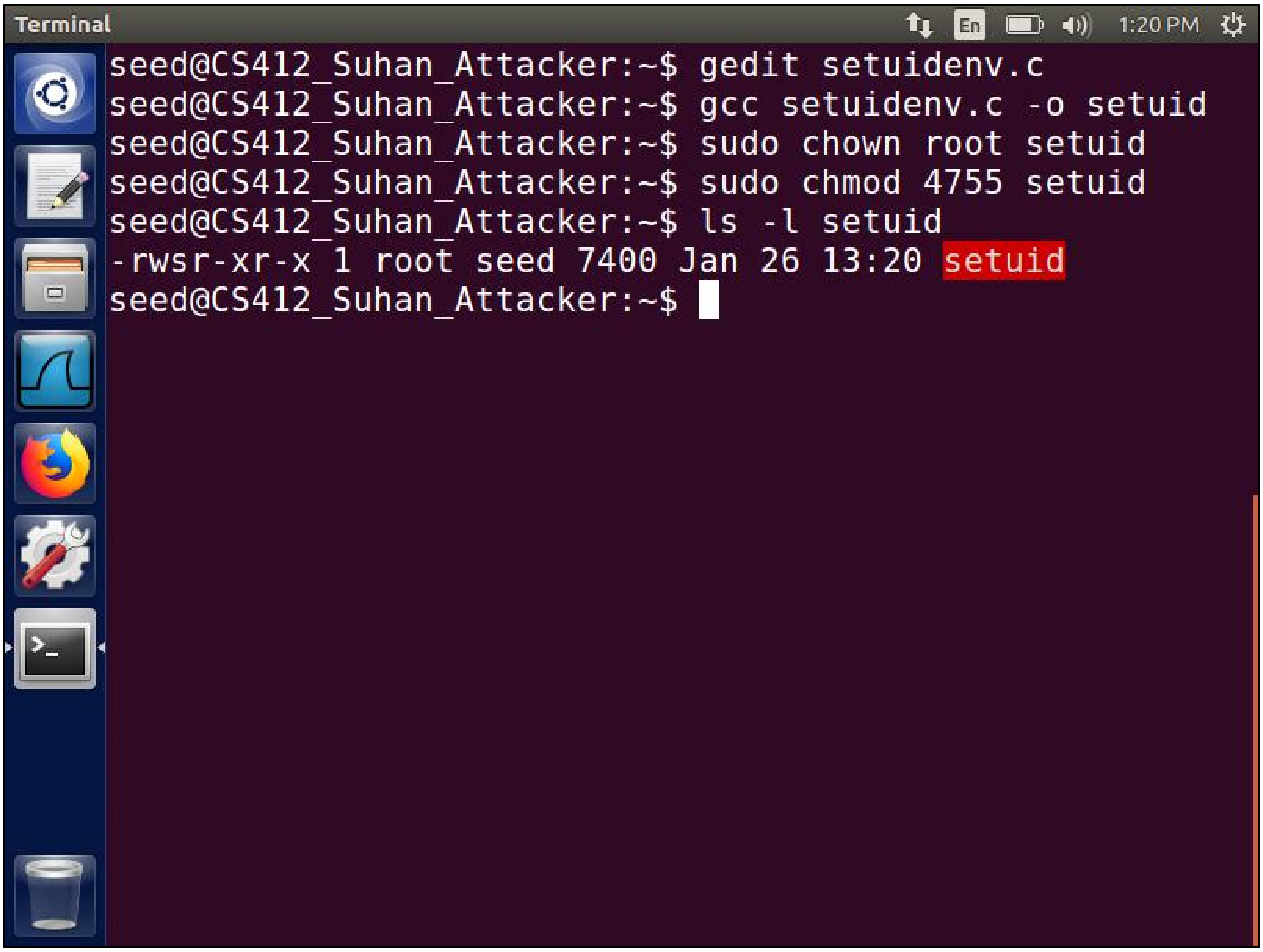
Commands:

$gcc setuidenv.c -o setuid

$sudo chown root setuid

$sudo chmod 4755 setuid

$la -l setuid



```
seed@CS412_Suhan_Attacker:~$ gedit setuidenv.c
seed@CS412_Suhan_Attacker:~$ gcc setuidenv.c -o setuid
seed@CS412_Suhan_Attacker:~$ sudo chown root setuid
seed@CS412_Suhan_Attacker:~$ sudo chmod 4755 setuid
seed@CS412_Suhan_Attacker:~$ ls -l setuid
-rwsr-xr-x 1 root seed 7400 Jan 26 13:20 setuid
seed@CS412_Suhan_Attacker:~$ 
```

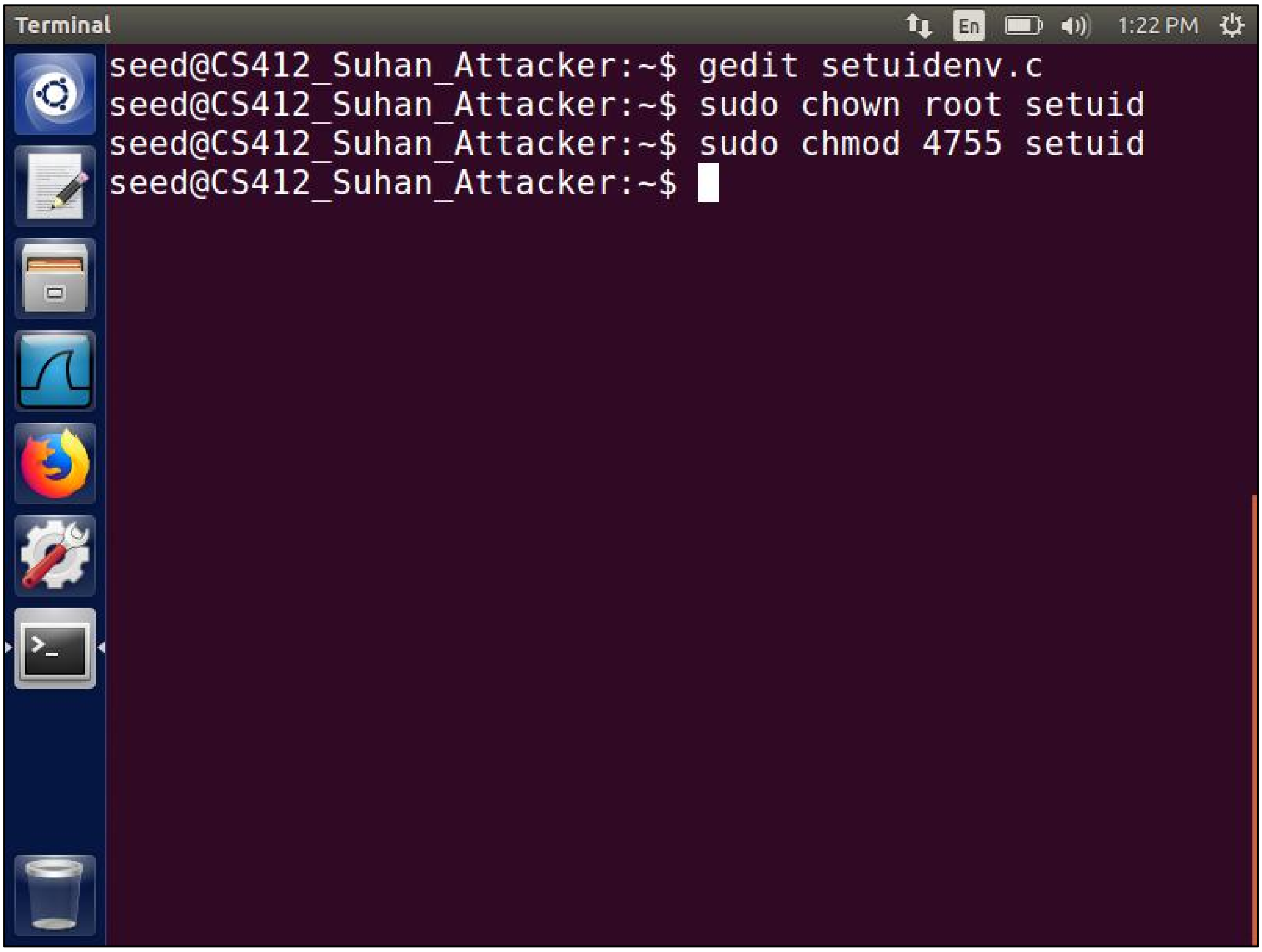
Step 2: Compile the above program, change its ownership to root, and make it a Set-UID program.

Commands:

$gcc setuidenv.c -o setuidenv

$sudo chmod 5744 setuidenv

$la -l setuidenv



```
seed@CS412_Suhan_Attacker:~$ gedit setuidenv.c
seed@CS412_Suhan_Attacker:~$ sudo chown root setuid
seed@CS412_Suhan_Attacker:~$ sudo chmod 4755 setuid
seed@CS412_Suhan_Attacker:~$ 
```

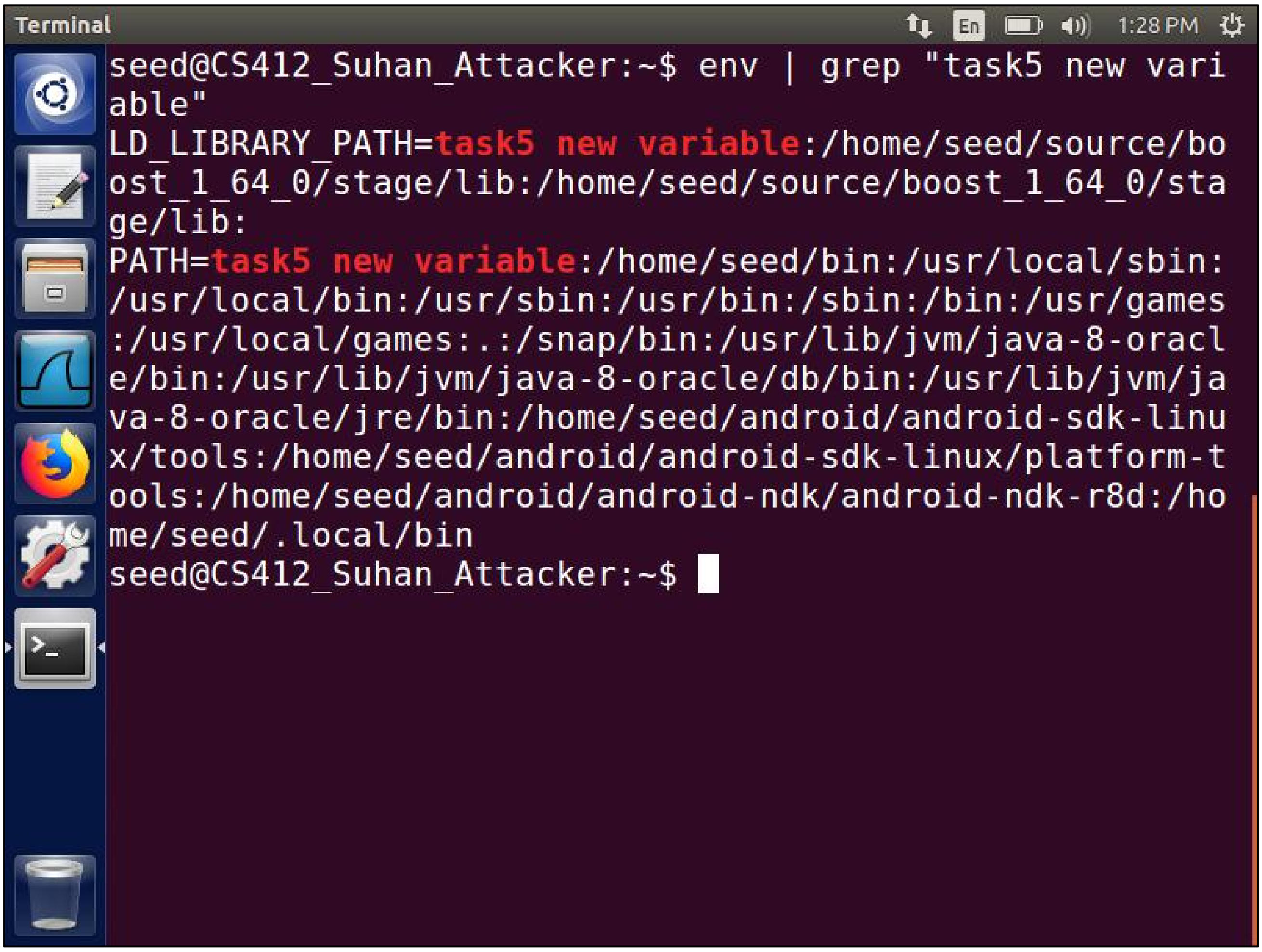
Step 3: In your Bash shell (you need to be in a normal user account, not the root account), use the export command to set the following environment variables (they may have already exist):

- PATH
- LD LIBRARY PATH
- ANY NAME (this is an environment variable defined by you, so pick whatever name you want).

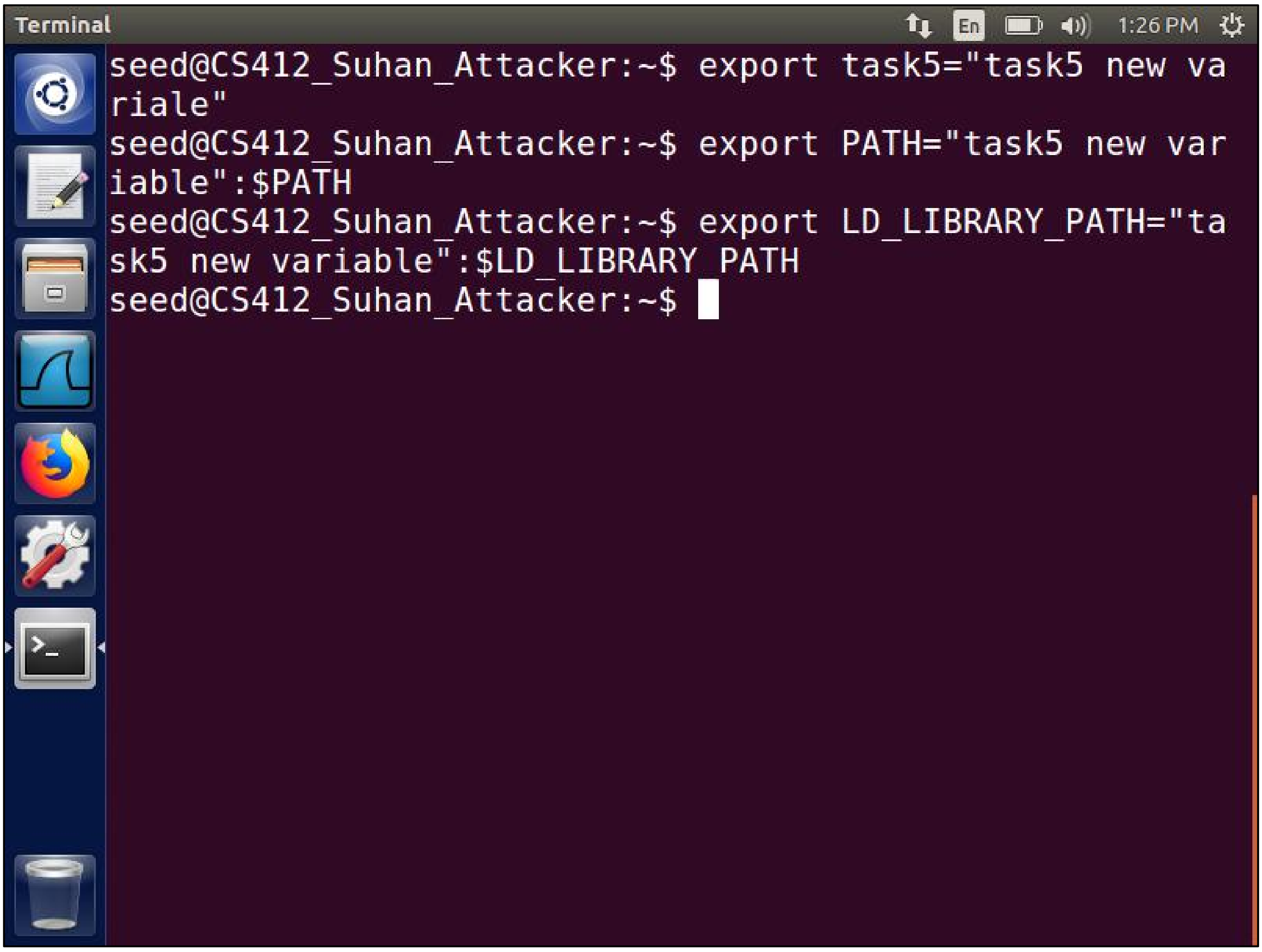
Commands:

```
 $printenv PATH
$printenv LD_LIBRARY_PATH
$export LD_LIBRARY_PATH=/home/seed:$LD_LIBRARY_PATH
$printenv LD_LIBRARY_PATH
$printenv task5
$export task5='task5 new variable'
$printenv task5

$env > env_result
$diff setuidenv env_result
$setuidenv > setuidenv_res
$diff setuidenv_res env_result
```

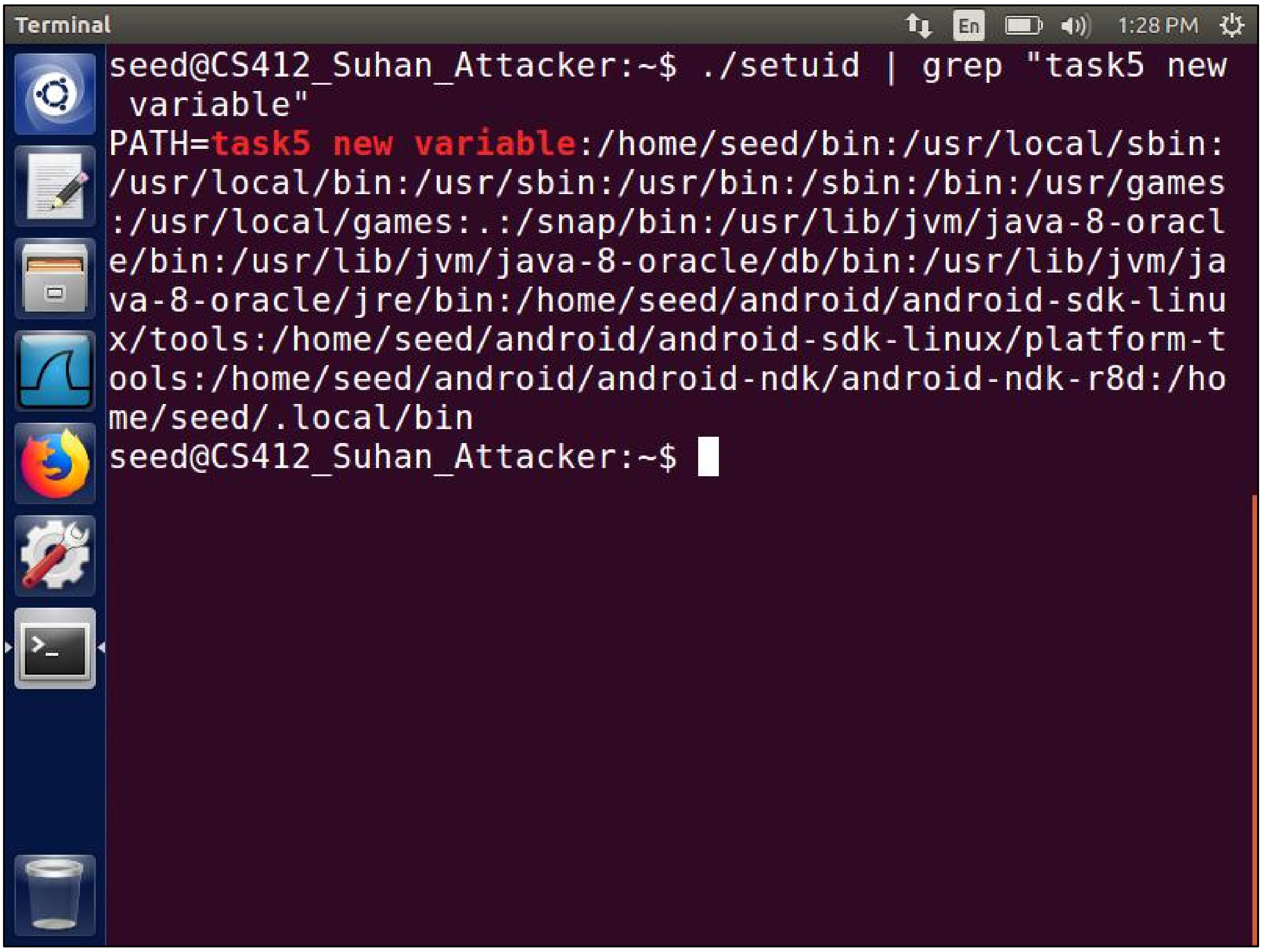
These environment variables are set in the user's shell process. Now, run the Set-UID program from Step 2 in your shell. After you type the name of the program in your shell, the shell forks a child process, and uses the child process to run the program. Please check whether all the environment variables you set in the shell process (parent) get into the Set-UID child process. Describe your observation. If there are surprises to you, describe them.



```
seed@CS412_Suhan_Attacker:~$ env | grep "task5 new vari
able"
LD_LIBRARY_PATH=task5 new variable:/home/seed/source/bo
ost_1_64_0/stage/lib:/home/seed/source/boost_1_64_0/sta
ge/lib:
PATH=task5 new variable:/home/seed/bin:/usr/local/sbin:
/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
:/usr/local/games:.:/snap/bin:/usr/lib/jvm/java-8-oracl
e/bin:/usr/lib/jvm/java-8-oracle/db/bin:/usr/lib/jvm/ja
va-8-oracle/jre/bin:/home/seed/android/android-sdk-linu
x/tools:/home/seed/android/android-sdk-linux/platform-t
ools:/home/seed/android/android-ndk/android-ndk-r8d:/ho
me/seed/.local/bin
seed@CS412_Suhan_Attacker:~$ 
```



```
seed@CS412_Suhan_Attacker:~$ export task5="task5 new va
riale"
seed@CS412_Suhan_Attacker:~$ export PATH="task5 new var
iable":$PATH
seed@CS412_Suhan_Attacker:~$ export LD_LIBRARY_PATH="ta
sk5 new variable":$LD_LIBRARY_PATH
seed@CS412_Suhan_Attacker:~$ 
```



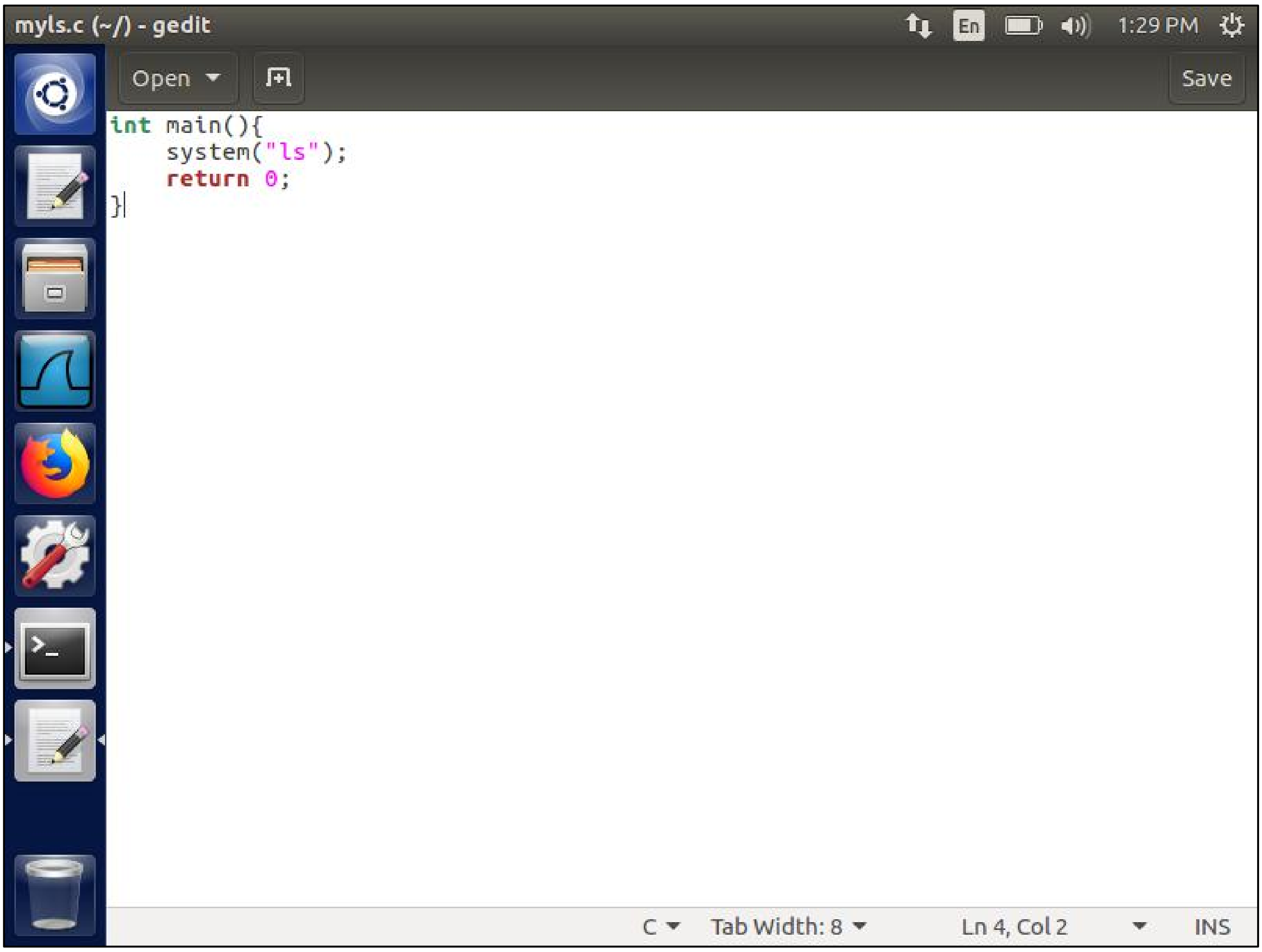
```
seed@CS412_Suhan_Attacker:~$ ./setuid | grep "task5 new
 variable"
PATH=task5 new variable:/home/seed/bin:/usr/local/sbin:
/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
:/usr/local/games:.:/snap/bin:/usr/lib/jvm/java-8-oracl
e/bin:/usr/lib/jvm/java-8-oracle/db/bin:/usr/lib/jvm/ja
va-8-oracle/jre/bin:/home/seed/android/android-sdk-linu
x/tools:/home/seed/android/android-sdk-linux/platform-t
ools:/home/seed/android/android-ndk/android-ndk-r8d:/ho
me/seed/.local/bin
seed@CS412_Suhan_Attacker:~$ 
```

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

Because of the shell program invoked, calling system() within a Set-UID program is quite dangerous. This is because the actual behavior of the shell program can be affected by environment variables, such as PATH; these environment variables are provided by the user, who may be malicious. By changing these variables, malicious users can control the behavior of the Set-UID program. In Bash, you can change the PATH environment variable in the following way (this example adds the directory /home/seed to the beginning of the PATH environment variable):

```
$ export PATH=/home/seed:$PATH
```

The Set-UID program below is supposed to execute the /bin/ls command; however, the programmer only uses the relative path for the ls command, rather than the absolute path:

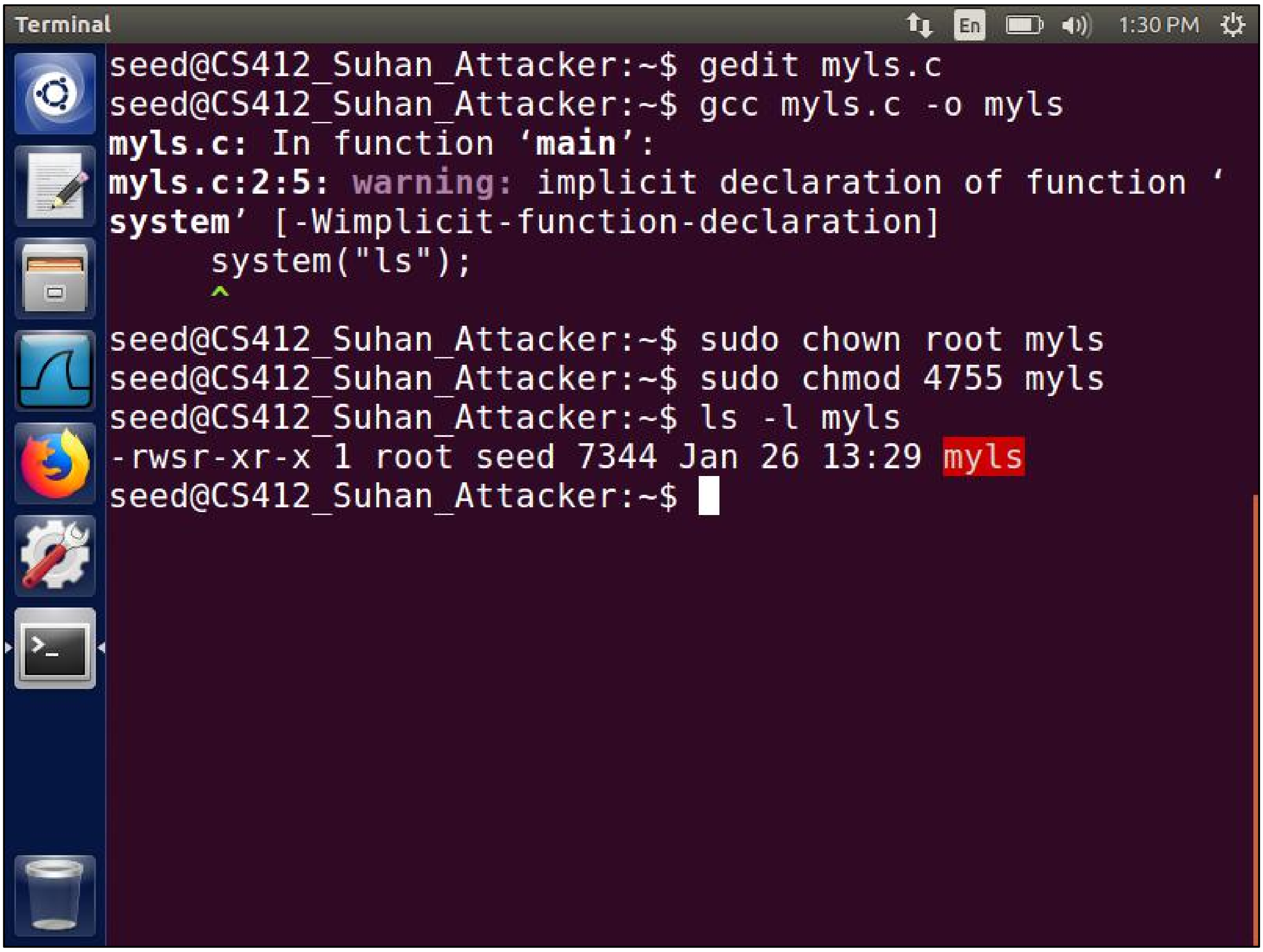


```
int main(){
	system("ls");
	return 0;
}
```

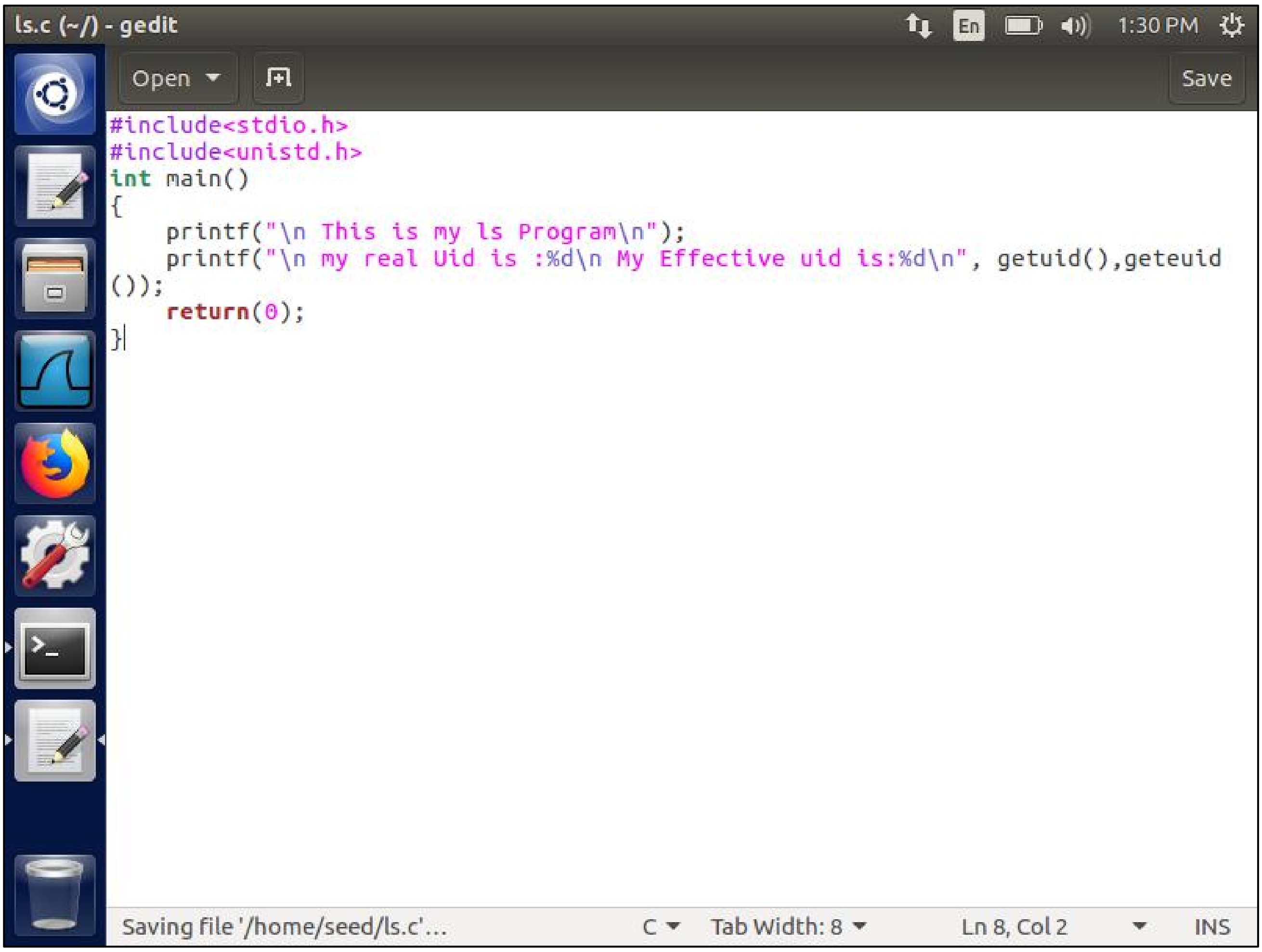
Please compile the above program, and change its owner to root, and make it a Set-UID program. Can you let this Set-UID program run your code instead of /bin/ls? If you can, is your code running with the root privilege? Describe and explain your observations.

Commands:

```
$ gcc myls.c -o myls
$sudo chown root myls
$sudo chmod 4755 myls
$ls -l myls
```



```
seed@CS412_Suhan_Attacker:~$ gedit myls.c
seed@CS412_Suhan_Attacker:~$ gcc myls.c -o myls
myls.c: In function ‘main’:
myls.c:2:5: warning: implicit declaration of function ‘
system’ [-Wimplicit-function-declaration]
     system("ls");
     ^
seed@CS412_Suhan_Attacker:~$ sudo chown root myls
seed@CS412_Suhan_Attacker:~$ sudo chmod 4755 myls
seed@CS412_Suhan_Attacker:~$ ls -l myls
-rwsr-xr-x 1 root seed 7344 Jan 26 13:29 myls
seed@CS412_Suhan_Attacker:~$ 
```

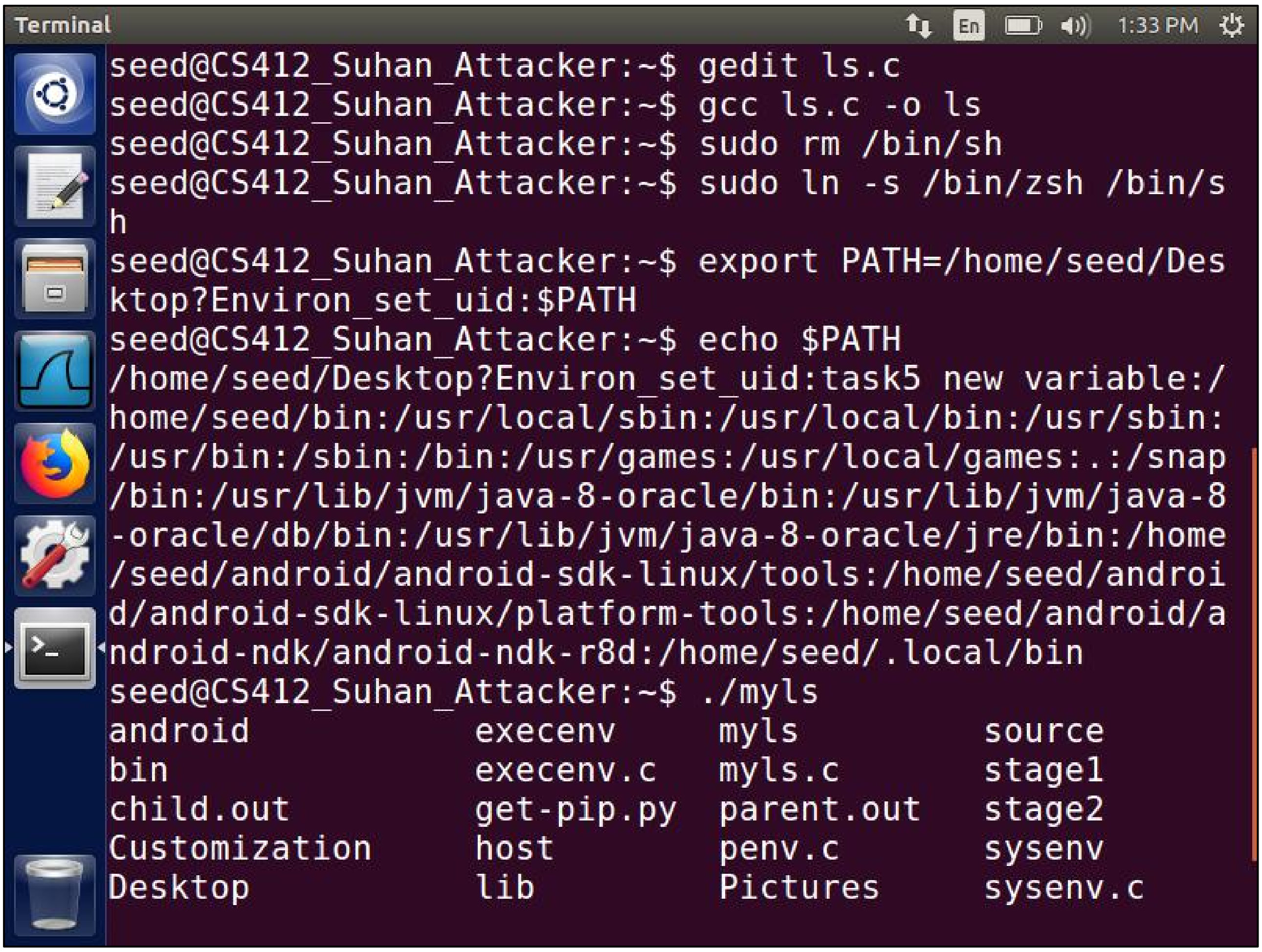


```
#include<stdio.h>
#include<unistd.h>
int main()
{
	printf("\n This is my ls Program\n");
	printf("\n my real Uid is :%d\n My Effective uid is:%d\n", getuid(),geteuid
());
	return(0);
}
```

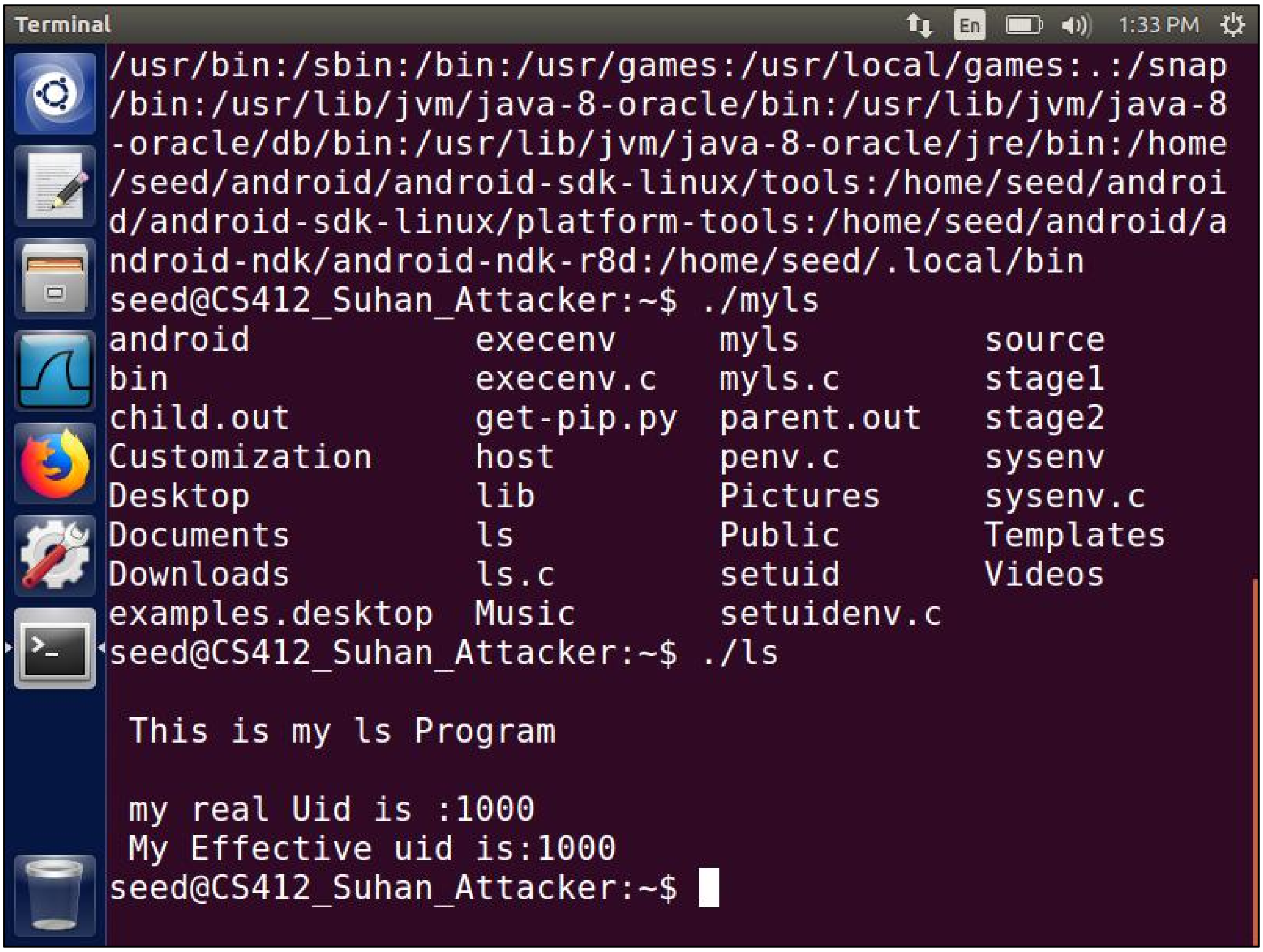
Saving file '/home/seed/ls.c'… C Tab Width: 8 Ln 8, Col 2 INS

Commands:

```
$ gcc ls.c -o ls
$ sudo rm /bin/sh
$ sudo ln -s /bin/zsh /bin/sh
$export PATH=/home/seed/Desktop/Environ_set_uid:$PATH
$echo $PATH
$./myls
$./ls
```



```
seed@CS412_Suhan_Attacker:~$ gedit ls.c
seed@CS412_Suhan_Attacker:~$ gcc ls.c -o ls
seed@CS412_Suhan_Attacker:~$ sudo rm /bin/sh
seed@CS412_Suhan_Attacker:~$ sudo ln -s /bin/zsh /bin/s
h
seed@CS412_Suhan_Attacker:~$ export PATH=/home/seed/Des
ktop?Environ_set_uid:$PATH
seed@CS412_Suhan_Attacker:~$ echo $PATH
/home/seed/Desktop?Environ_set_uid:task5 new variable:/
home/seed/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:
/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:.:/snap
/bin:/usr/lib/jvm/java-8-oracle/bin:/usr/lib/jvm/java-8
-oracle/db/bin:/usr/lib/jvm/java-8-oracle/jre/bin:/home
/seed/android/android-sdk-linux/tools:/home/seed/androi
d/android-sdk-linux/platform-tools:/home/seed/android/a
ndroid-ndk/android-ndk-r8d:/home/seed/.local/bin
seed@CS412_Suhan_Attacker:~$ ./myls
android            execenv     myls          source
bin                execenv.c   myls.c        stage1
child.out          get-pip.py  parent.out    stage2
Customization      host        penv.c        sysenv
Desktop            lib         Pictures      sysenv.c
```



```
/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:.:/snap
/bin:/usr/lib/jvm/java-8-oracle/bin:/usr/lib/jvm/java-8
-oracle/db/bin:/usr/lib/jvm/java-8-oracle/jre/bin:/home
/seed/android/android-sdk-linux/tools:/home/seed/androi
d/android-sdk-linux/platform-tools:/home/seed/android/a
ndroid-ndk/android-ndk-r8d:/home/seed/.local/bin
seed@CS412_Suhan_Attacker:~$ ./myls
android            execenv     myls          source
bin                execenv.c   myls.c        stage1
child.out          get-pip.py  parent.out    stage2
Customization      host        penv.c        sysenv
Desktop            lib         Pictures      sysenv.c
Documents          ls          Public        Templates
Downloads          ls.c        setuid        Videos
examples.desktop   Music       setuidenv.c
seed@CS412_Suhan_Attacker:~$ ./ls

 This is my ls Program

 my real Uid is :1000
 My Effective uid is:1000
seed@CS412_Suhan_Attacker:~$ 
```

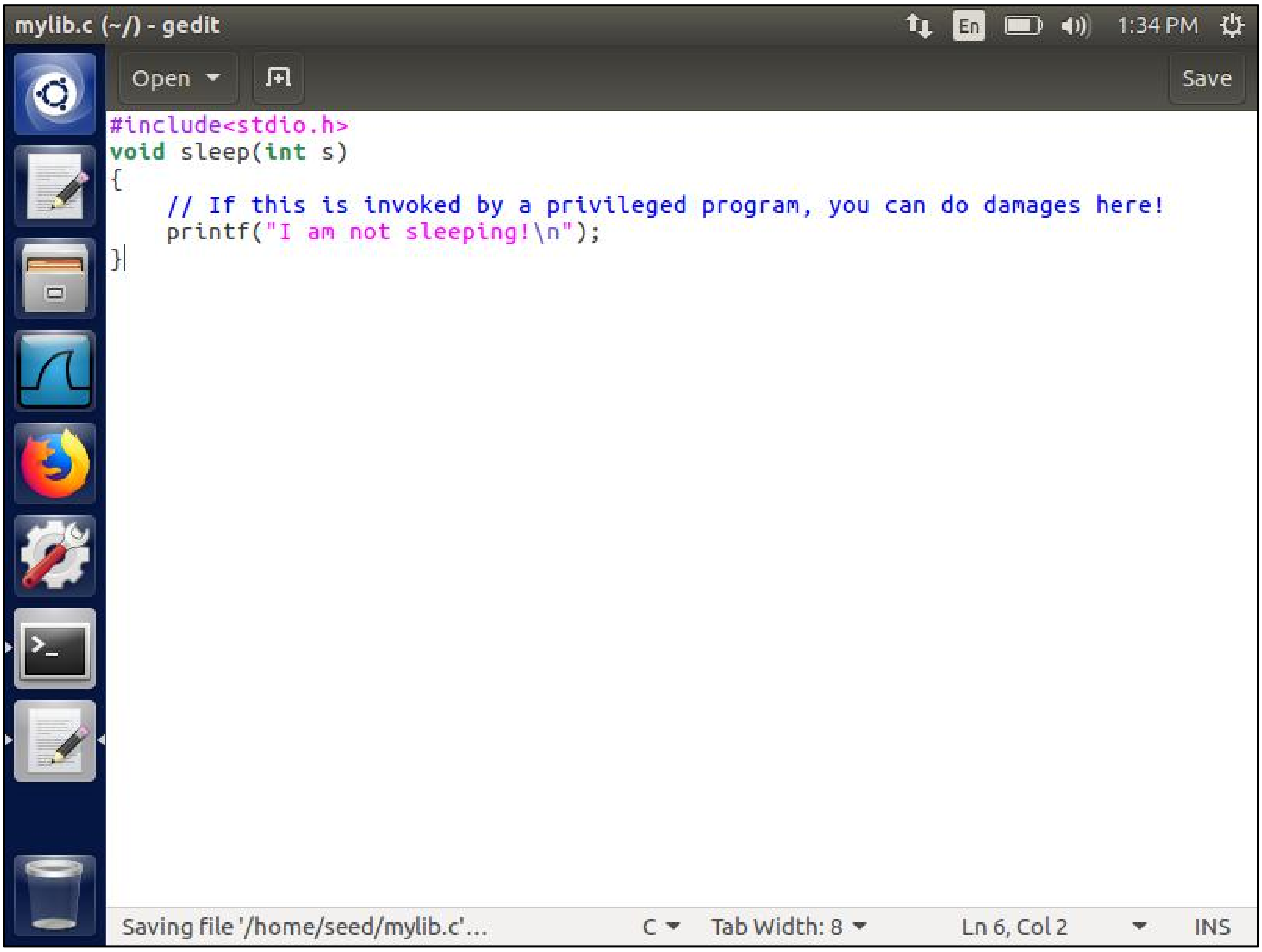
Task 7: The LD PRELOAD environment variable and Set-UID Programs

In this task, study how Set-UID programs deal with some of the environment variables. Several environment variables, including LD_PRELOAD, LD_LIBRARY PATH, and other LD influence the behavior of dynamic loader/linker. A dynamic loader/linker is the part of an operating system (OS) that loads (from persistent storage to RAM) and links the shared libraries needed by an executable at runtime.

In Linux, ld.so or ld-linux.so, are the dynamic loader/linker (each for different types of binary). Among the environment variables that affect their behaviors, LD LIBRARY PATH and LD PRELOAD are the two that we are concerned with in this lab. In Linux, LD LIBRARY PATH is a colon separated set of directories where libraries should be searched for first, before the standard set of directories. LD_PRELOAD specifies a list of additional, user-specified, shared libraries to be loaded before all others. In this task, students will only study LD_PRELOAD.

Step 1: First, see how these environment variables influence the behavior of dynamic loader/linker when running a normal program. Please follow these steps:

1. Build a dynamic link library. Create the following program, and name it mylib.c. It basically overrides the sleep() function in libc:

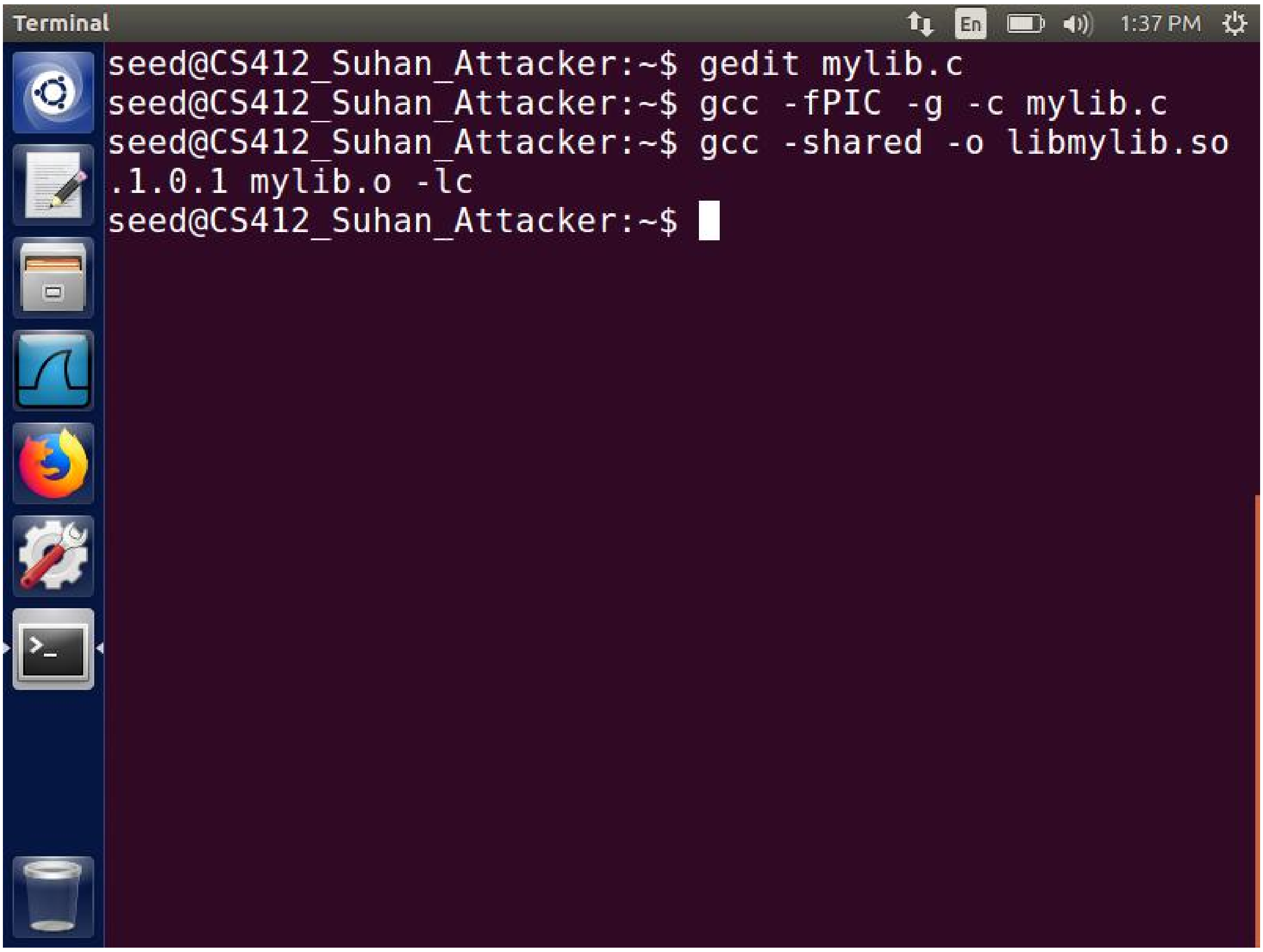


```
#include<stdio.h>
void sleep(int s)
{
    // If this is invoked by a privileged program, you can do damages here!
    printf("I am not sleeping!\n");
}
```

2. Compile the above program using the following commands (in the -lc argument, the second character is `):

   Command:

   ```
   $ gcc -fPIC -g -c mylib.c
   $ gcc -shared -o libmylib.so.1.0.1 mylib.o -lc
   ```

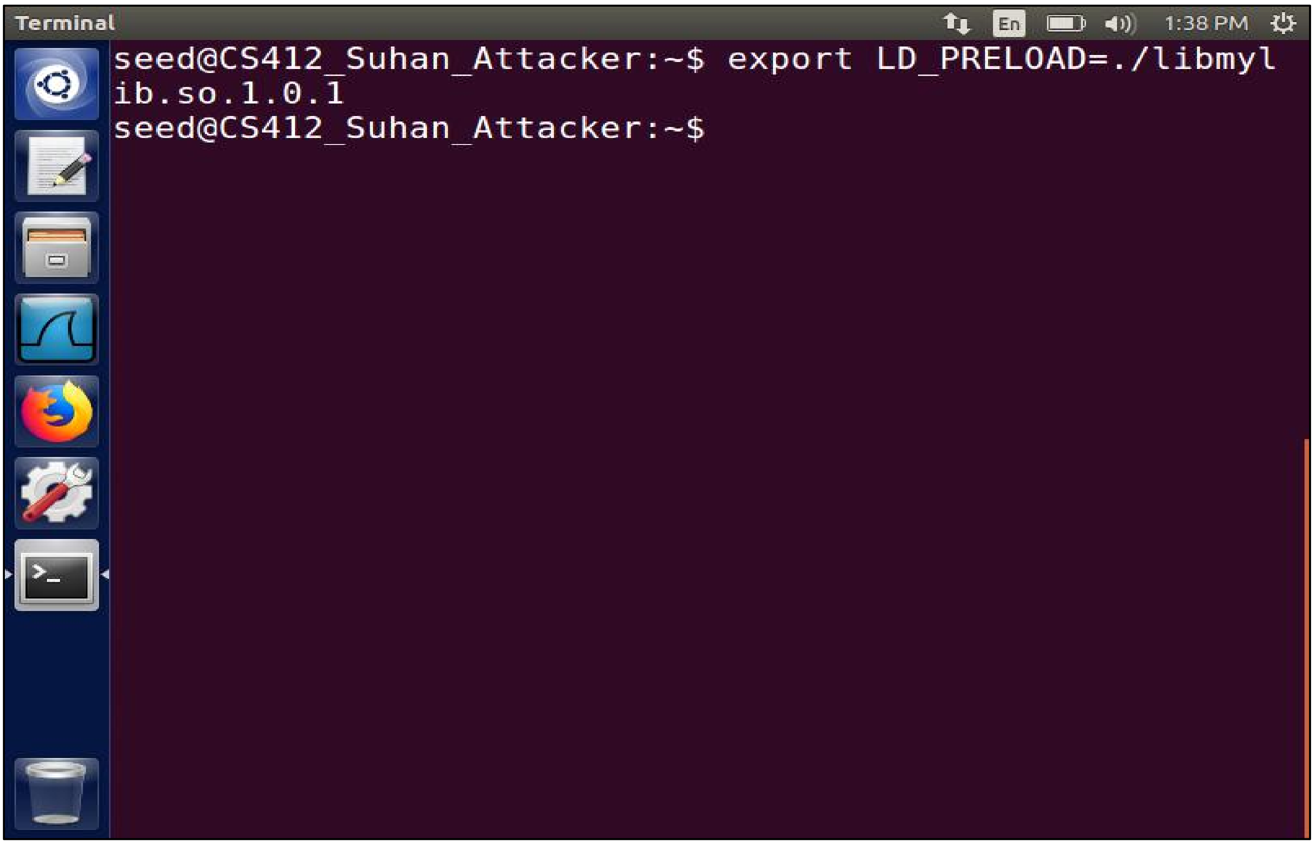


```
seed@CS412_Suhan_Attacker:~$ gedit mylib.c
seed@CS412_Suhan_Attacker:~$ gcc -fPIC -g -c mylib.c
seed@CS412_Suhan_Attacker:~$ gcc -shared -o libmylib.so
.1.0.1 mylib.o -lc
seed@CS412_Suhan_Attacker:~$ 
```

3. Now, set the LD_PRELOAD environment variable:

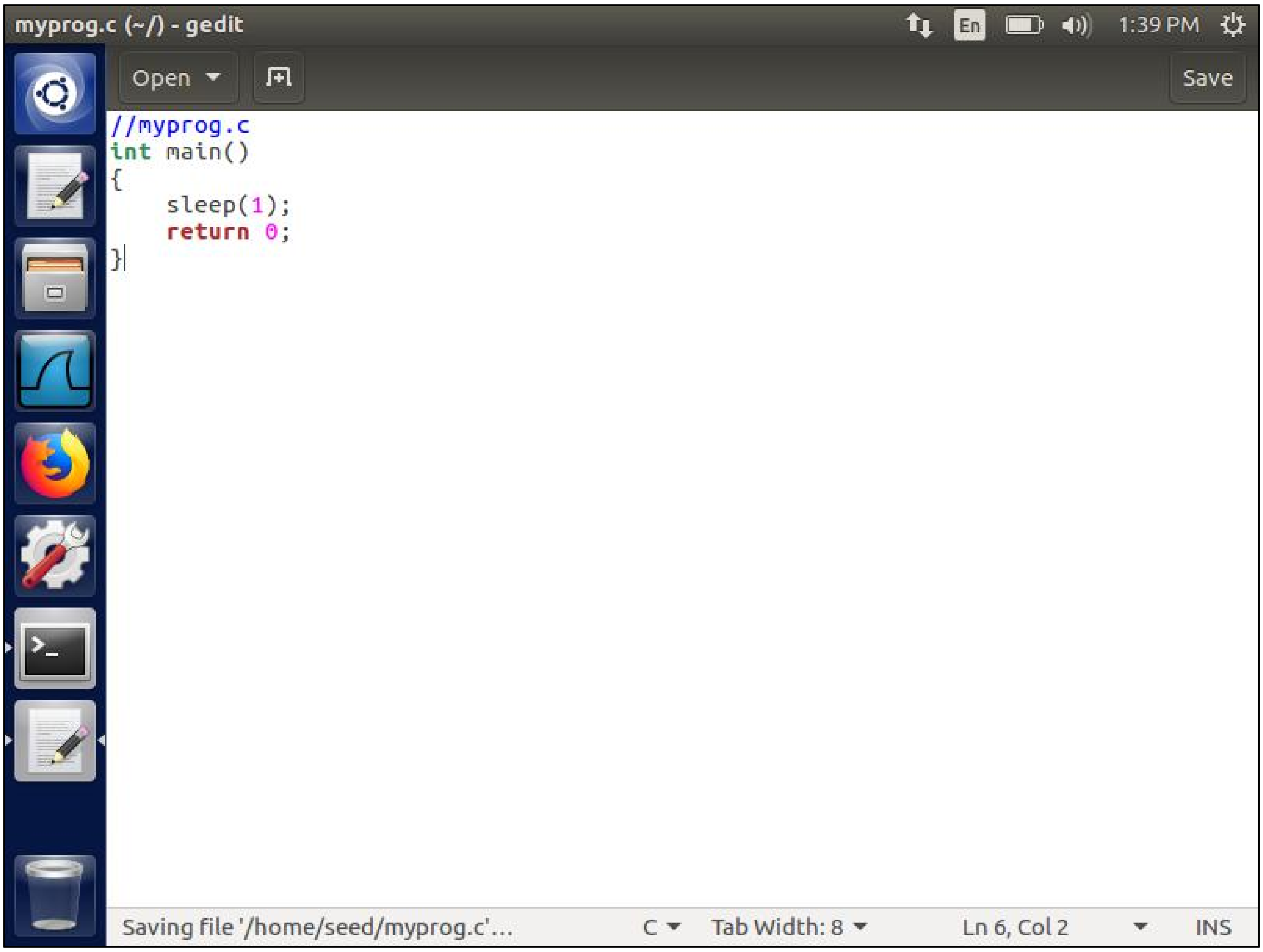
Command:

$export LD_PRELOAD=./libmylib.so.1.0.1



```
seed@CS412_Suhan_Attacker:~$ export LD_PRELOAD=./libmyl
ib.so.1.0.1
seed@CS412_Suhan_Attacker:~$
```

4. Finally, compile the following program myprog, and it in the same directory as the above dynamic link library libmylib.so.1.0.1:



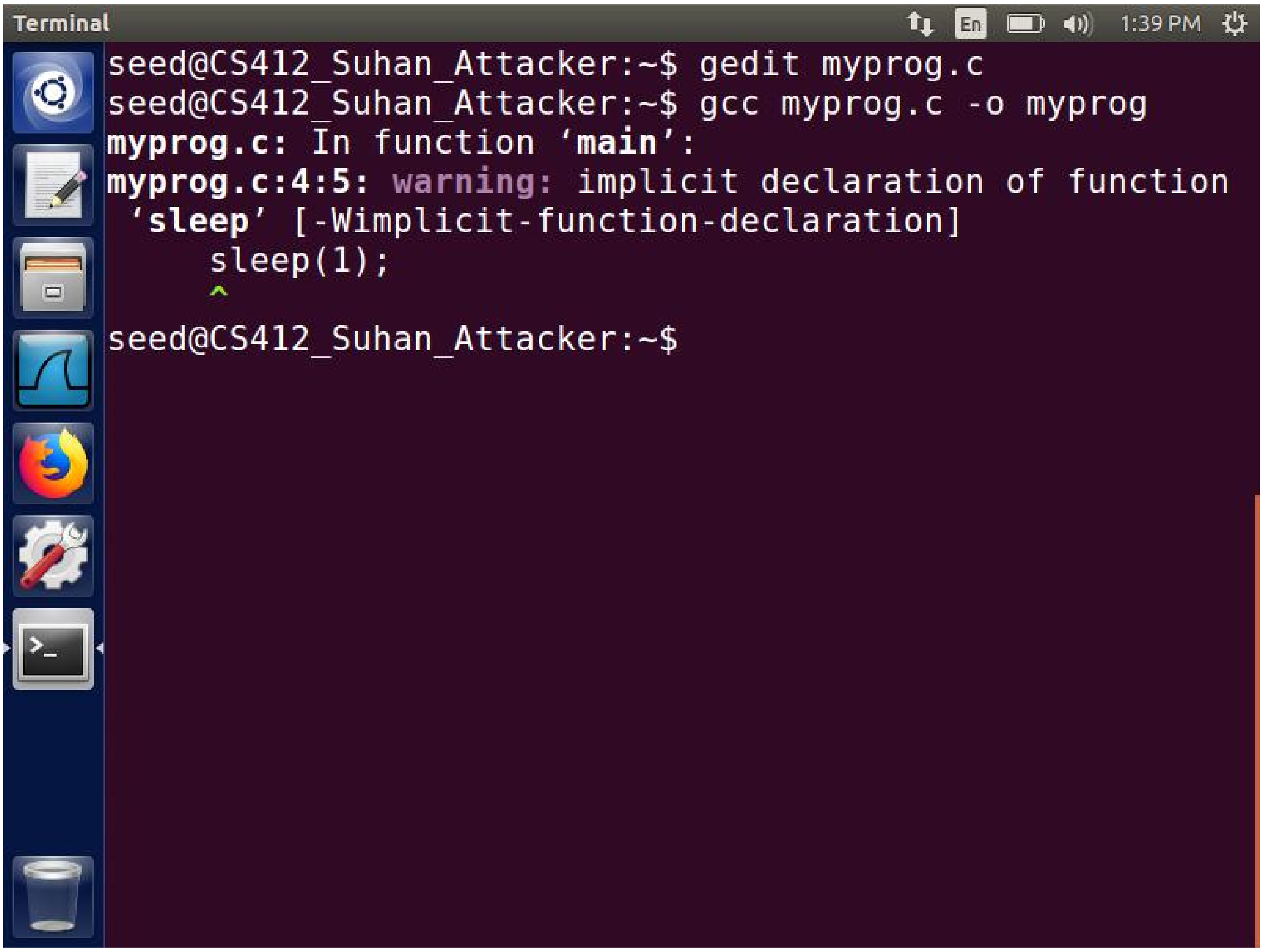
```
//myprog.c
int main()
{
	sleep(1);
	return 0;
}
```

Step 2: After you have done the above, please run myprog under the following conditions, and observe what happens.

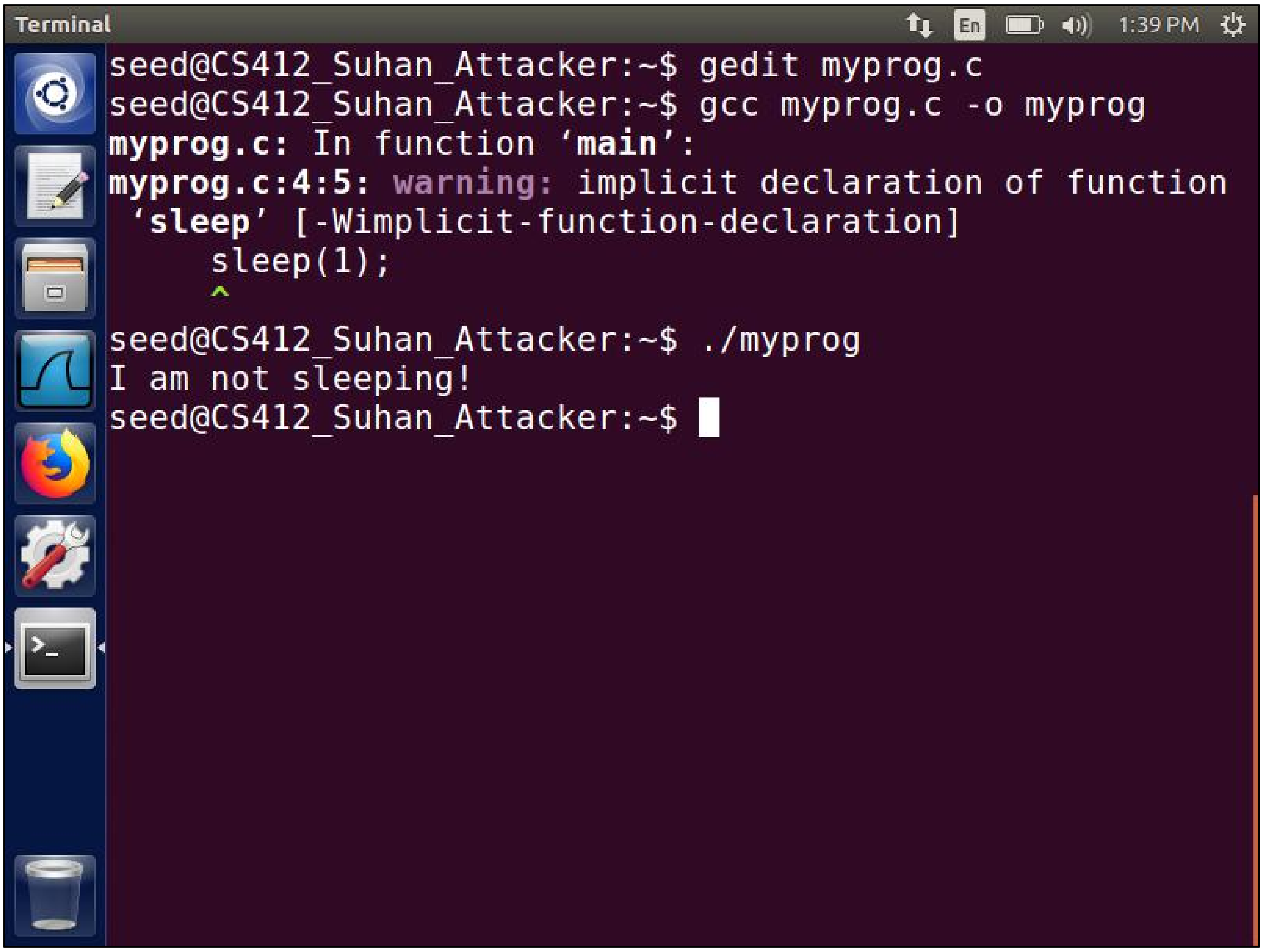
- Make myprog a regular program, and run it as a normal user.
- Make myprog a Set-UID root program, and run it as a normal user.
- Make myprog a Set-UID root program, export the LD_PRELOAD environment variable again in the root account and run it.
- Make myprog a Set-UID user1 program (i.eThe owner is 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.

Command:

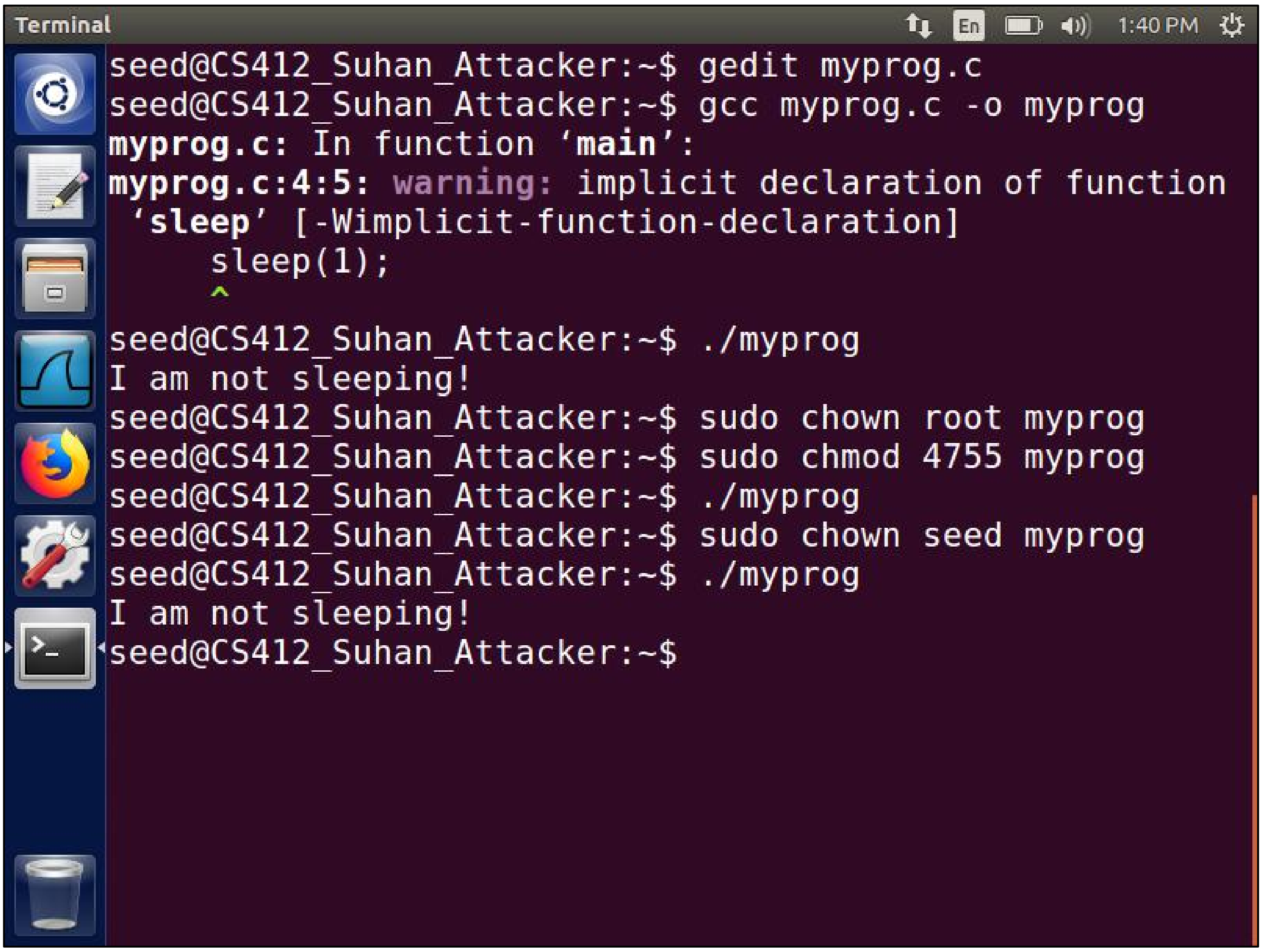
```
$gcc myprog.c -o myprog
$ ./myprog
```



```
seed@CS412_Suhan_Attacker:~$ gedit myprog.c
seed@CS412_Suhan_Attacker:~$ gcc myprog.c -o myprog
myprog.c: In function ‘main’:
myprog.c:4:5: warning: implicit declaration of function
 ‘sleep’ [-Wimplicit-function-declaration]
     sleep(1);
     ^
seed@CS412_Suhan_Attacker:~$
```



```
seed@CS412_Suhan_Attacker:~$ gedit myprog.c
seed@CS412_Suhan_Attacker:~$ gcc myprog.c -o myprog
myprog.c: In function ‘main’:
myprog.c:4:5: warning: implicit declaration of function
 ‘sleep’ [-Wimplicit-function-declaration]
     sleep(1);
     ^
seed@CS412_Suhan_Attacker:~$ ./myprog
I am not sleeping!
seed@CS412_Suhan_Attacker:~$ 
```



```
seed@CS412_Suhan_Attacker:~$ gedit myprog.c
seed@CS412_Suhan_Attacker:~$ gcc myprog.c -o myprog
myprog.c: In function ‘main’:
myprog.c:4:5: warning: implicit declaration of function
 ‘sleep’ [-Wimplicit-function-declaration]
     sleep(1);
     ^
seed@CS412_Suhan_Attacker:~$ ./myprog
I am not sleeping!
seed@CS412_Suhan_Attacker:~$ sudo chown root myprog
seed@CS412_Suhan_Attacker:~$ sudo chmod 4755 myprog
seed@CS412_Suhan_Attacker:~$ ./myprog
seed@CS412_Suhan_Attacker:~$ sudo chown seed myprog
seed@CS412_Suhan_Attacker:~$ ./myprog
I am not sleeping!
seed@CS412_Suhan_Attacker:~$
```

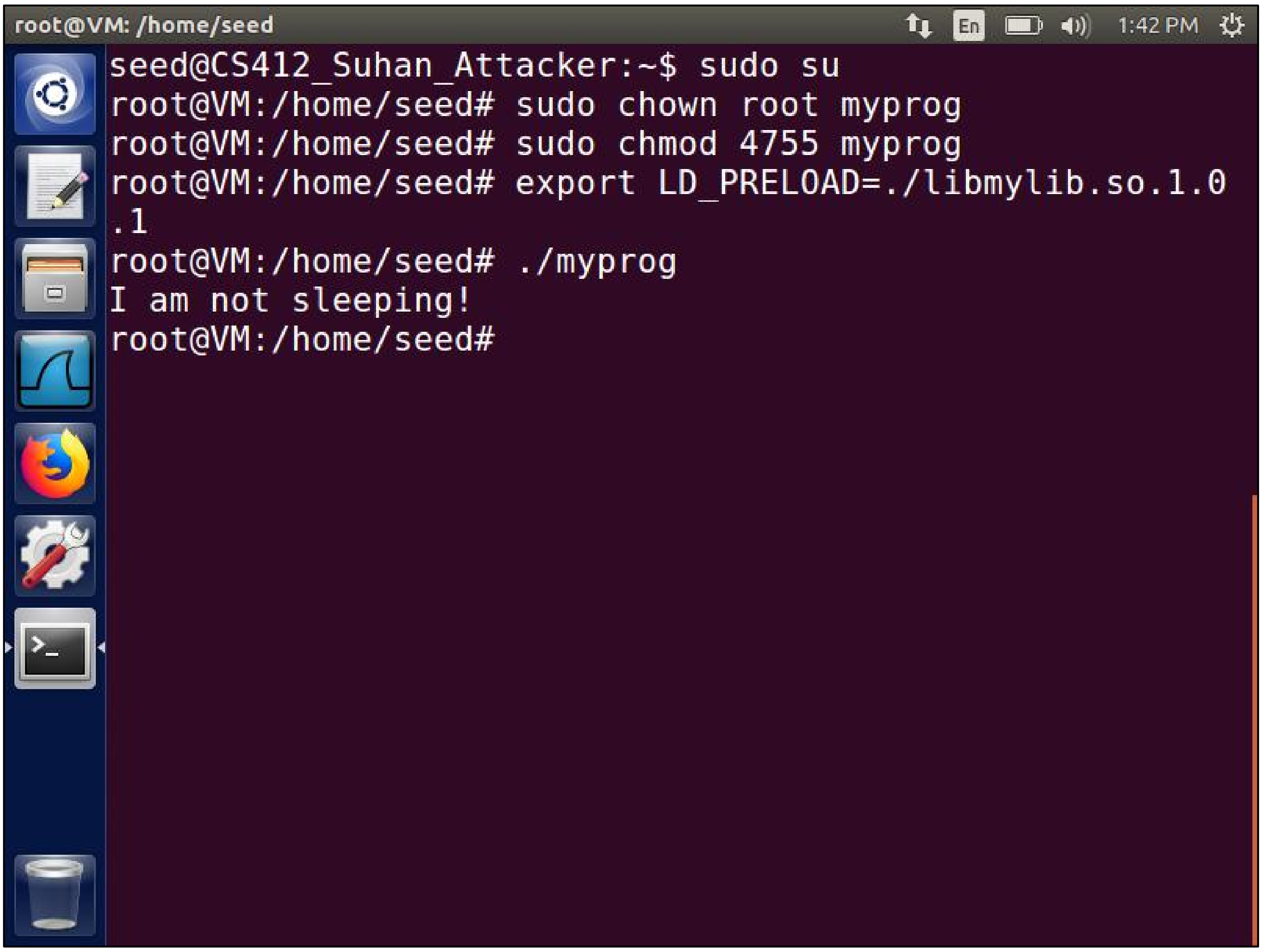
Step 3: You should be able to observe different behaviors in the scenarios described above, even though you are running the same program. You need to figure out what causes the difference. Environment variables play a role here. Please design an experiment to figure out the main causes, and explain why the behaviors in Step 2 are different. (Hint: the child process may not inherit the LD * environment variables).

In root environment execute the myprog.c program

Command:

```
$ gcc myprog.c -o myprog
$ chmod 4755 mypro
$ ls -l myprog
$export LD_PRELOAD=./libmylib.so.1.0.1
```

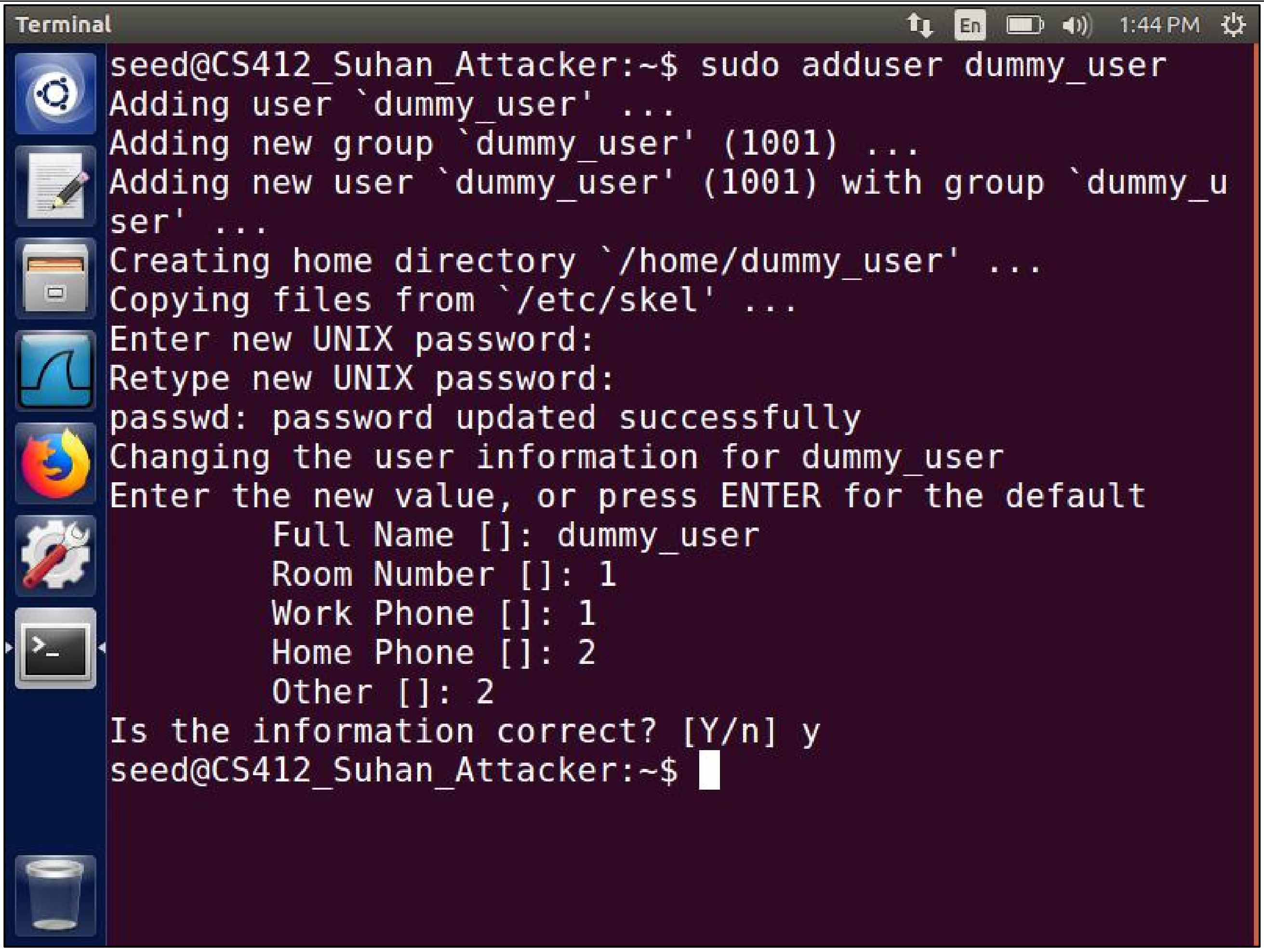
come out of root and check the behavior



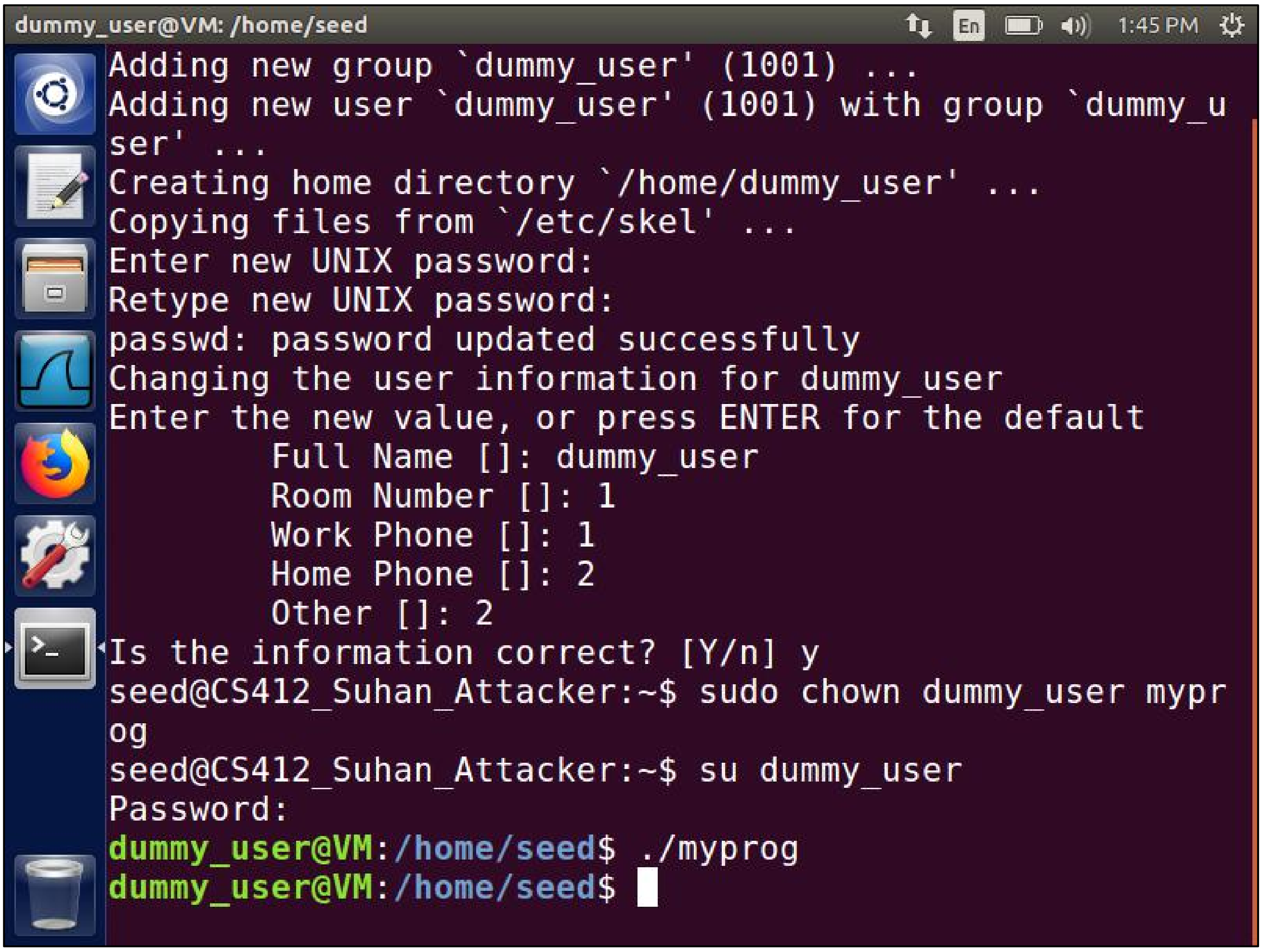
```
seed@CS412_Suhan_Attacker:~$ sudo su
root@VM:/home/seed# sudo chown root myprog
root@VM:/home/seed# sudo chmod 4755 myprog
root@VM:/home/seed# export LD_PRELOAD=./libmylib.so.1.0
.1
root@VM:/home/seed# ./myprog
I am not sleeping!
root@VM:/home/seed#
```

Command:

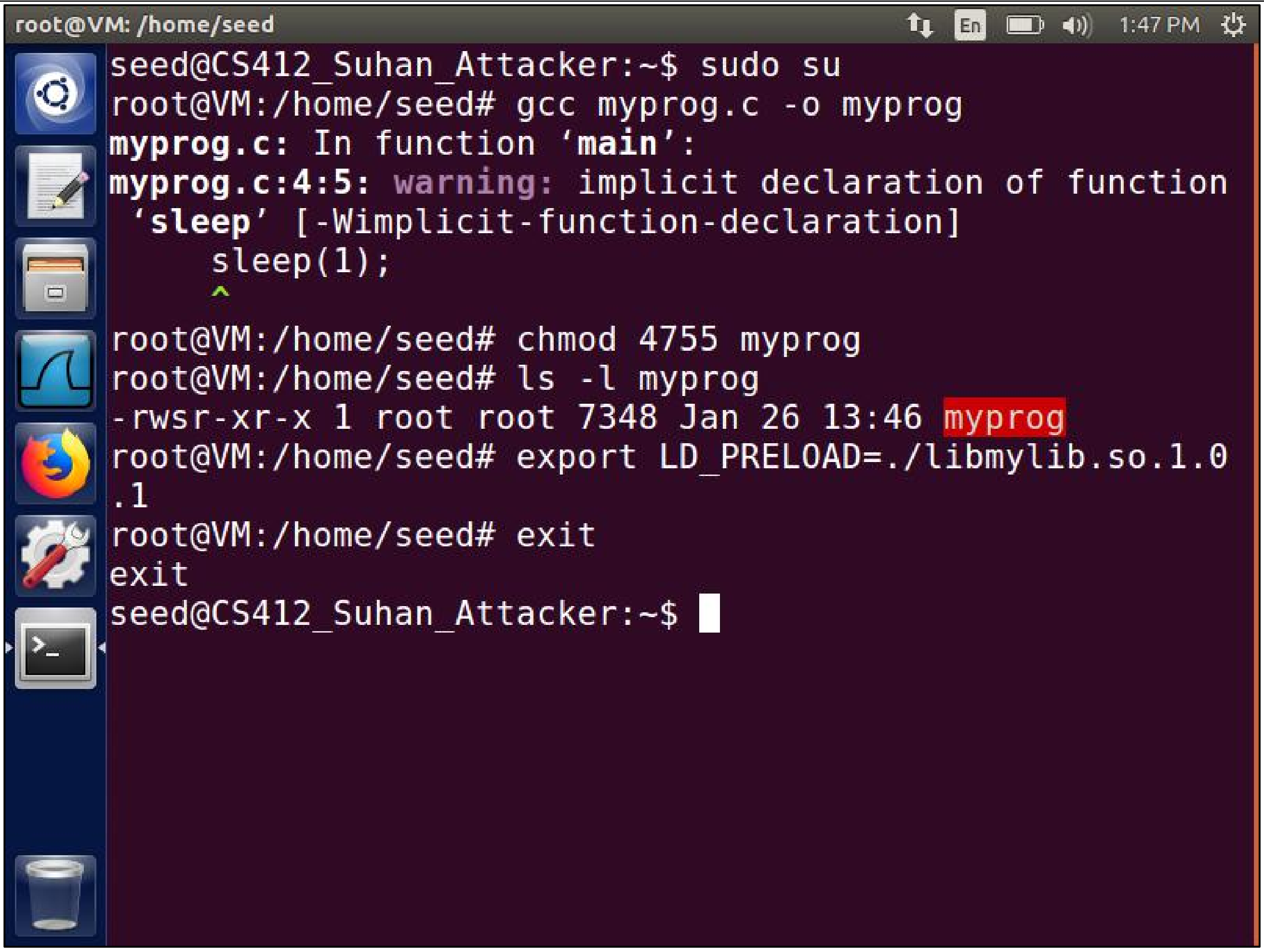
```
$ ls -l myprog
$export
LD_PRELOAD=./libmylib.so.1.0.1
$ whoami
$ seed
$ ./myprog
```



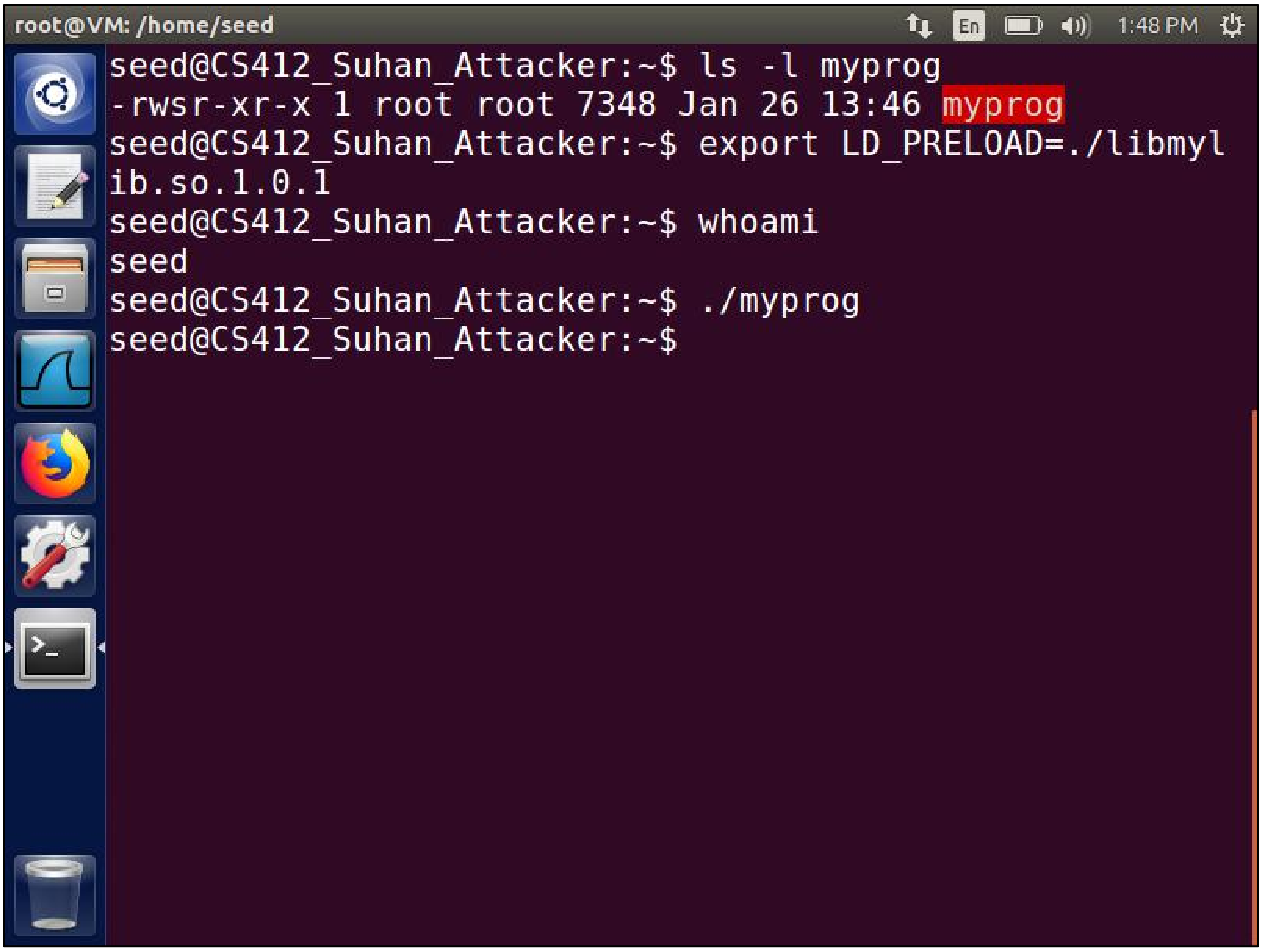
```
seed@CS412_Suhan_Attacker:~$ sudo adduser dummy_user
Adding user `dummy_user' ...
Adding new group `dummy_user' (1001) ...
Adding new user `dummy_user' (1001) with group `dummy_u
ser' ...
Creating home directory `/home/dummy_user' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for dummy_user
Enter the new value, or press ENTER for the default
        Full Name []: dummy_user
        Room Number []: 1
        Work Phone []: 1
        Home Phone []: 2
        Other []: 2
Is the information correct? [Y/n] y
seed@CS412_Suhan_Attacker:~$ 
```



```
Adding new group `dummy_user' (1001) ...
Adding new user `dummy_user' (1001) with group `dummy_u
ser' ...
Creating home directory `/home/dummy_user' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for dummy_user
Enter the new value, or press ENTER for the default
        Full Name []: dummy_user
        Room Number []: 1
        Work Phone []: 1
        Home Phone []: 2
        Other []: 2
Is the information correct? [Y/n] y
seed@CS412_Suhan_Attacker:~$ sudo chown dummy_user mypr
og
seed@CS412_Suhan_Attacker:~$ su dummy_user
Password:
dummy_user@VM:/home/seed$ ./myprog
dummy_user@VM:/home/seed$ 
```



```
seed@CS412_Suhan_Attacker:~$ sudo su
root@VM:/home/seed# gcc myprog.c -o myprog
myprog.c: In function ‘main’:
myprog.c:4:5: warning: implicit declaration of function
 ‘sleep’ [-Wimplicit-function-declaration]
     sleep(1);
     ^
root@VM:/home/seed# chmod 4755 myprog
root@VM:/home/seed# ls -l myprog
-rwsr-xr-x 1 root root 7348 Jan 26 13:46 myprog
root@VM:/home/seed# export LD_PRELOAD=./libmylib.so.1.0
.1
root@VM:/home/seed# exit
exit
seed@CS412_Suhan_Attacker:~$ 
```



```
seed@CS412_Suhan_Attacker:~$ ls -l myprog
-rwsr-xr-x 1 root root 7348 Jan 26 13:46 myprog
seed@CS412_Suhan_Attacker:~$ export LD_PRELOAD=./libmyl
ib.so.1.0.1
seed@CS412_Suhan_Attacker:~$ whoami
seed
seed@CS412_Suhan_Attacker:~$ ./myprog
seed@CS412_Suhan_Attacker:~$
```

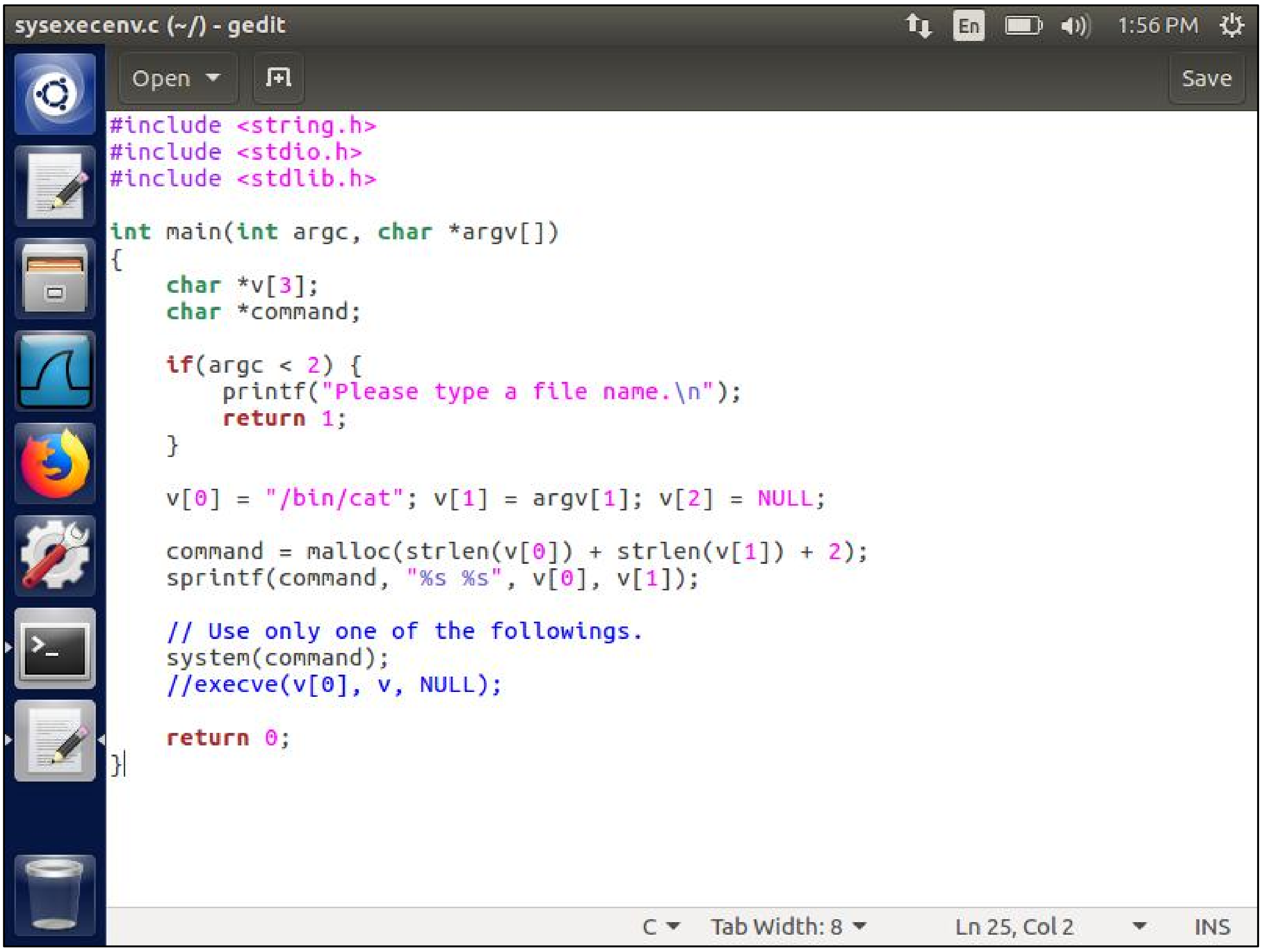
Task 8: Invoking external programs using system() versus execve()

Although system() and execve() can both be used to run new programs, system() is quite dangerous if used in a privileged program, such as Set-UID programs. We have seen how the PATH environment variable affects the behavior of system(), because the variable affects how the shell works. execve() does not have the problem, because it does not invoke shell. Invoking a shell has another dangerous consequence, and this time, it has nothing to do with environment variables.

Look at the following scenario:

Bob works for an auditing agency, and he needs to investigate a company for a suspected fraud. For the investigation purpose, Bob needs to be able to read all the files in the company's Unix system; on the other hand, to protect the integrity of the system, Bob should not be able to modify any file. To achieve this goal, Vince, the superuser of the system, wrote a special set-root uid program (see below), and then gave the executable permission to Bob.

This program requires Bob to type a file name at the command line, and then it will run /bin/cat to display the specified file. Since the program is running as a root, it can display any file Bob specifies. However, since the program has no write operations, Vince is very sure that Bob cannot use this special program to modify any file.

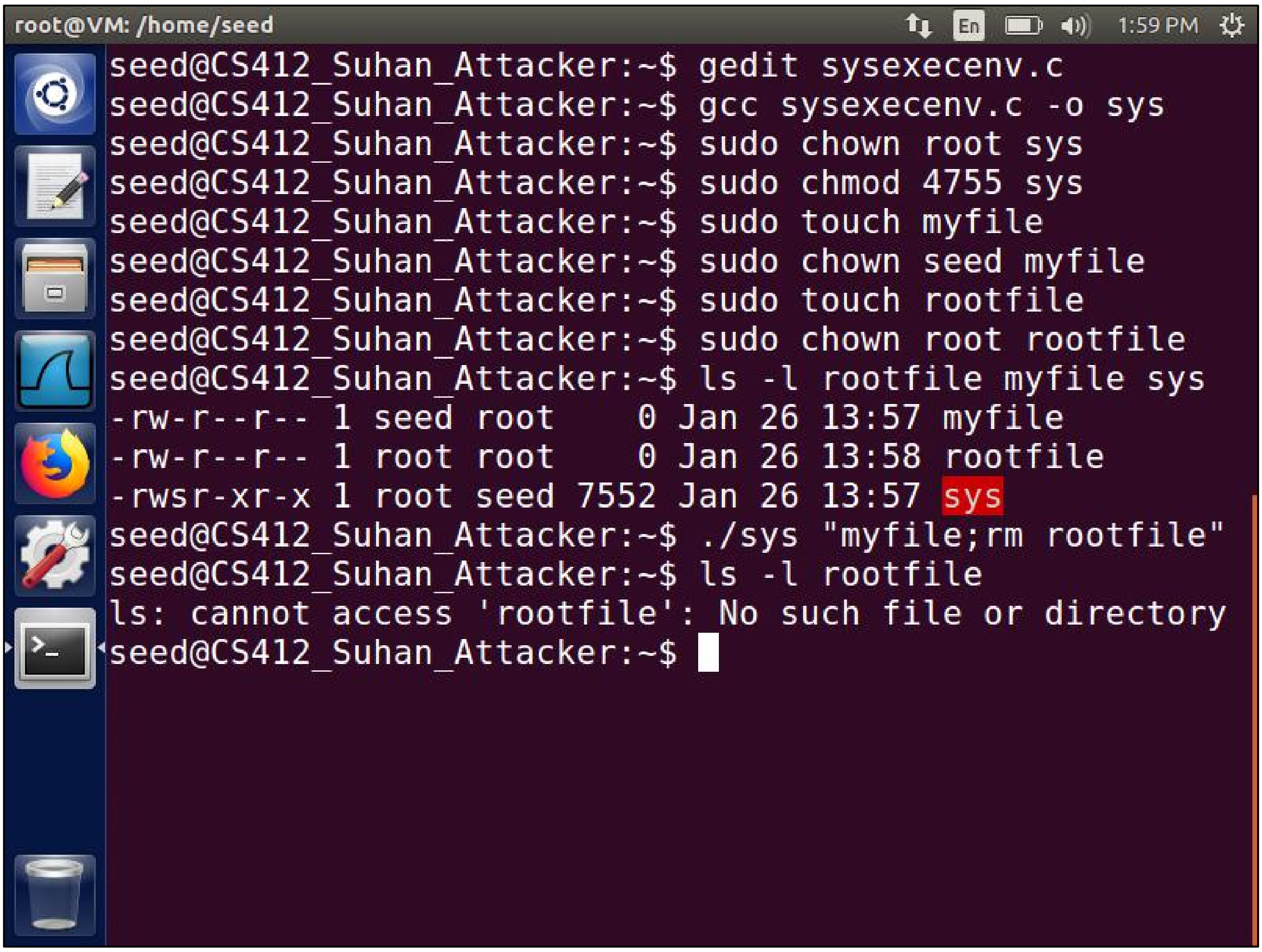


```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
	char *v[3];
	char *command;

	if(argc < 2) {
		printf("Please type a file name.\n");
		return 1;
	}

	v[0] = "/bin/cat"; v[1] = argv[1]; v[2] = NULL;

	command = malloc(strlen(v[0]) + strlen(v[1]) + 2);
	sprintf(command, "%s %s", v[0], v[1]);

	// Use only one of the followings.
	system(command);
	//execve(v[0], v, NULL);

	return 0;
}
```

Step 1: Compile the above program, make root its owner, and change it to a Set-UID program. The program will use system() to invoke the command. If you were Bob, can you compromise the integrity of the system? For example, can you remove a file that is not writable to you? (create two files "myfile"(owner is seed) and "root file"(owner is root - using chown cmd))

Command:

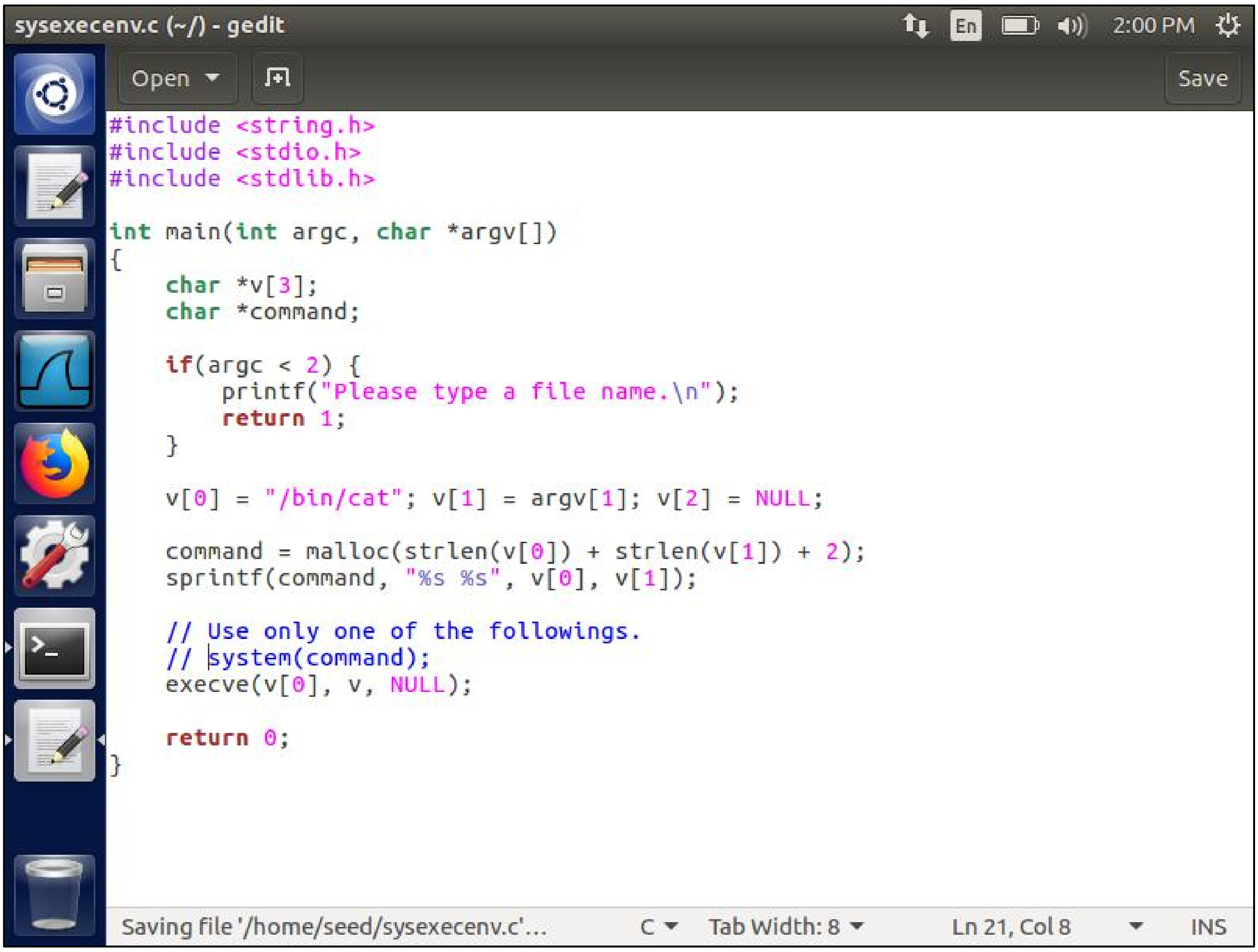
```
$gcc sysexecenv.c -o sys
$sudo chown root sys
$ sudo chmod 4755 sys
$ ls -l rootfile myfile
sys
$ ./sysexecenv "myfile;rm
rootfile" $ ls -l rootfile
```



root@VM: /home/seed

```
seed@CS412_Suhan_Attacker:~$ gedit sysexecenv.c
seed@CS412_Suhan_Attacker:~$ gcc sysexecenv.c -o sys
seed@CS412_Suhan_Attacker:~$ sudo chown root sys
seed@CS412_Suhan_Attacker:~$ sudo chmod 4755 sys
seed@CS412_Suhan_Attacker:~$ sudo touch myfile
seed@CS412_Suhan_Attacker:~$ sudo chown seed myfile
seed@CS412_Suhan_Attacker:~$ sudo touch rootfile
seed@CS412_Suhan_Attacker:~$ sudo chown root rootfile
seed@CS412_Suhan_Attacker:~$ ls -l rootfile myfile sys
-rw-r--r-- 1 seed root    0 Jan 26 13:57 myfile
-rw-r--r-- 1 root root    0 Jan 26 13:58 rootfile
-rwsr-xr-x 1 root seed 7552 Jan 26 13:57 sys
seed@CS412_Suhan_Attacker:~$ ./sys "myfile;rm rootfile"
seed@CS412_Suhan_Attacker:~$ ls -l rootfile
ls: cannot access 'rootfile': No such file or directory
seed@CS412_Suhan_Attacker:~$ 
```

Step 2: Comment out the system(command) statement, and uncomment the execve() statement; the program will use execve() to invoke the command. Compile the program, and make it SetUID (owned by root). Do your attacks in Step 1 still work? Please describe and explain your observations.



sysexecenv.c (~/) - gedit
2:00 PM
Open
Save

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
        char *v[3];
        char *command;

        if(argc < 2) {
                printf("Please type a file name.\n");
                return 1;
        }

        v[0] = "/bin/cat"; v[1] = argv[1]; v[2] = NULL;

        command = malloc(strlen(v[0]) + strlen(v[1]) + 2);
        sprintf(command, "%s %s", v[0], v[1]);

        // Use only one of the followings.
        // system(command);
        execve(v[0], v, NULL);

        return 0;
}
```

Saving file '/home/seed/sysexecenv.c'...
C
Tab Width: 8
Ln 21, Col 8
INS

Command:

$gcc sysexecenv.c -o exec

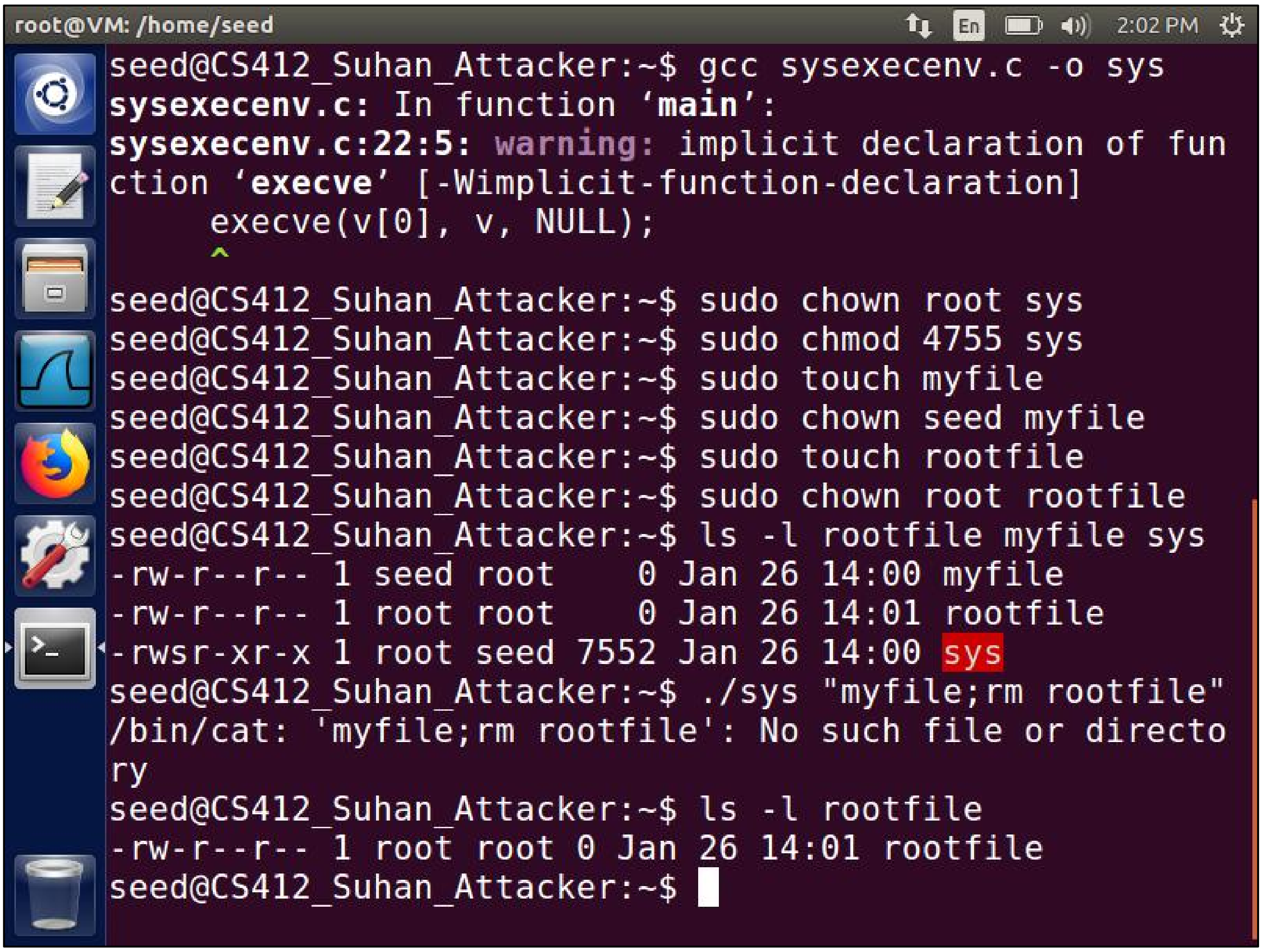
$sudo chown root exec

$ sudo chmod 4755

exec $ ls -l rootfile

myfile exec

$ ./sysexecenv “myfile;rm
rootfile” $ ls -l rootfile



```
seed@CS412_Suhan_Attacker:~$ gcc sysexecenv.c -o sys
sysexecenv.c: In function ‘main’:
sysexecenv.c:22:5: warning: implicit declaration of fun
ction ‘execve’ [-Wimplicit-function-declaration]
     execve(v[0], v, NULL);
     ^
seed@CS412_Suhan_Attacker:~$ sudo chown root sys
seed@CS412_Suhan_Attacker:~$ sudo chmod 4755 sys
seed@CS412_Suhan_Attacker:~$ sudo touch myfile
seed@CS412_Suhan_Attacker:~$ sudo chown seed myfile
seed@CS412_Suhan_Attacker:~$ sudo touch rootfile
seed@CS412_Suhan_Attacker:~$ sudo chown root rootfile
seed@CS412_Suhan_Attacker:~$ ls -l rootfile myfile sys
-rw-r--r-- 1 seed root    0 Jan 26 14:00 myfile
-rw-r--r-- 1 root root    0 Jan 26 14:01 rootfile
-rwsr-xr-x 1 root seed 7552 Jan 26 14:00 sys
seed@CS412_Suhan_Attacker:~$ ./sys "myfile;rm rootfile"
/bin/cat: 'myfile;rm rootfile': No such file or directo
ry
seed@CS412_Suhan_Attacker:~$ ls -l rootfile
-rw-r--r-- 1 root root 0 Jan 26 14:01 rootfile
seed@CS412_Suhan_Attacker:~$
```

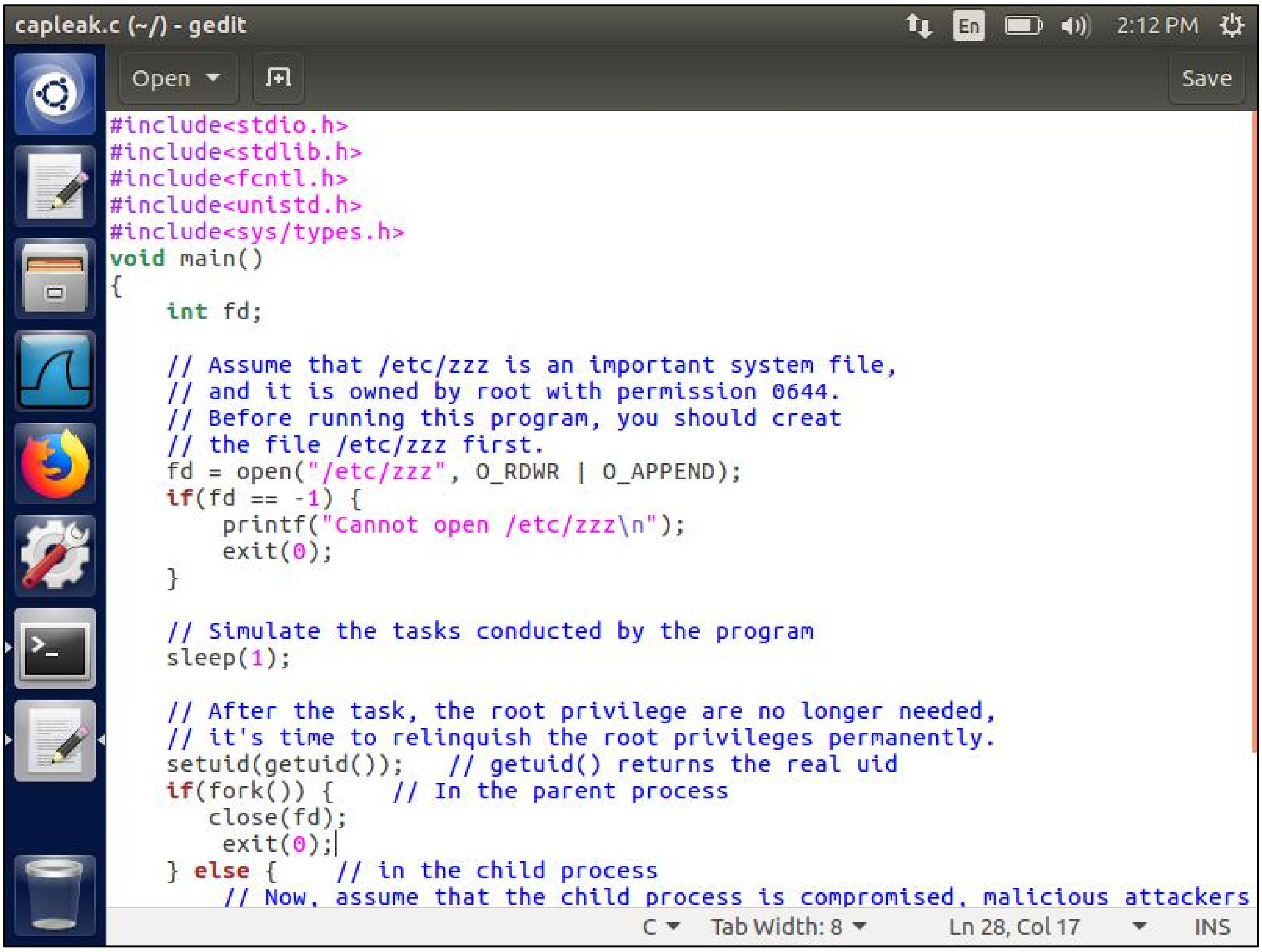
Task 9: Capability Leaking

To follow the Principle of Least Privilege, Set-UID programs often permanently relinquish their root privileges if such privileges are not needed anymore. Moreover, sometimes, the program needs to hand over its control to the user; in this case, root privileges must be revoked. The

setuid() system call can be used to revoke the privileges. According to the manual, “setuid() sets the effective user ID of the calling process. If the effective UID of the caller is root, the real UID and saved set-user-ID are also set”. Therefore, if a Set-UID program with effective UID 0 calls setuid(n), the process will become a normal process, with all its UIDs being set to n.

When revoking the privilege, one of the common mistakes is capability leaking. The process may have gained some privileged capabilities when it was still privileged; when the privileged is downgraded, if the program does not clean up those capabilities, they may still be accessible by the non-privileged process. In other words, although the effective user ID of the process becomes non-privileged, the process is still privileged because it possesses privileged capabilities.

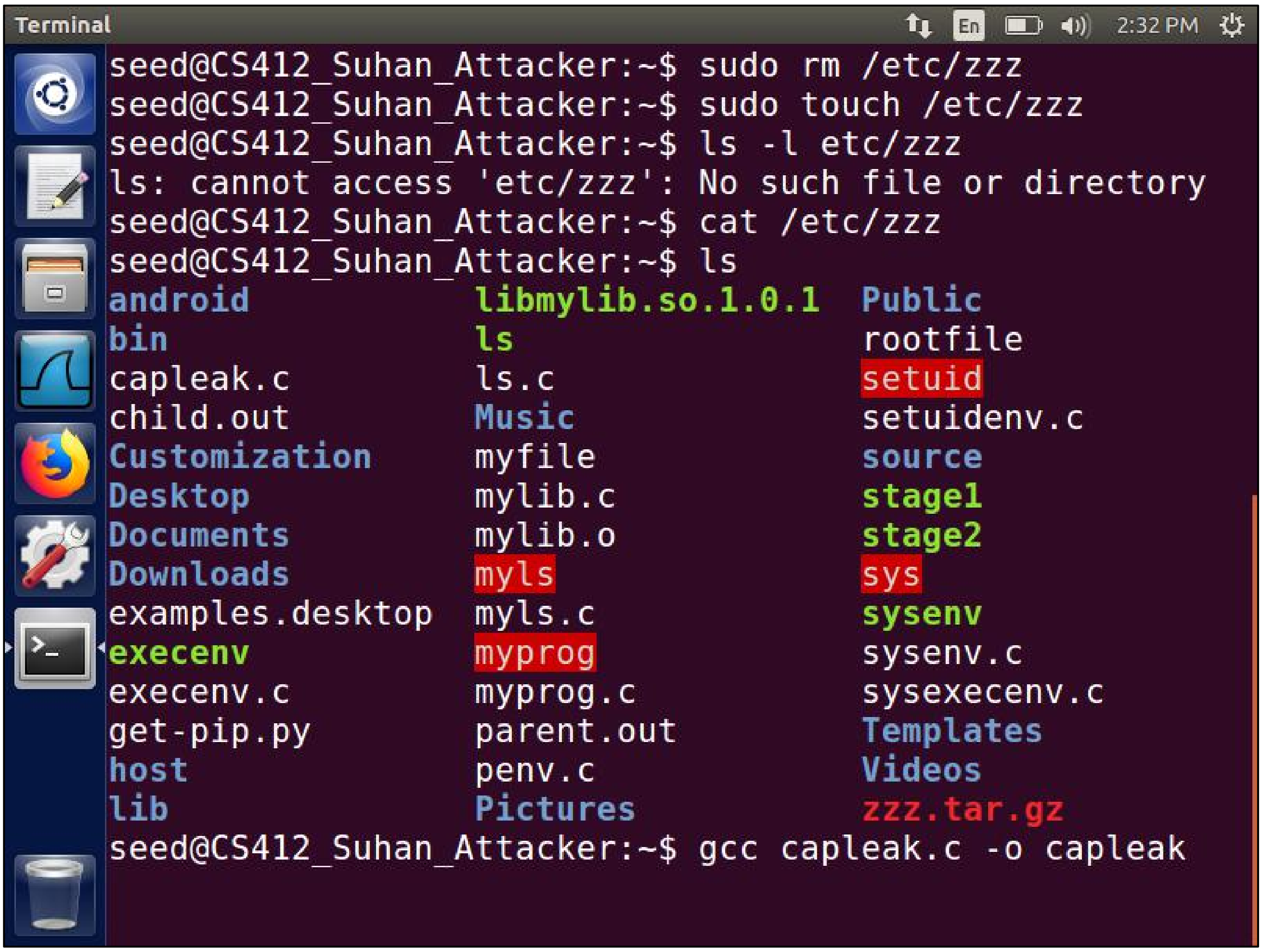
Compile the following program, change its owner to root, and make it a Set-UID program. Run the program as a normal user, and describe what you have observed. Will the file /etc/zzz be modified? Please explain your observation.



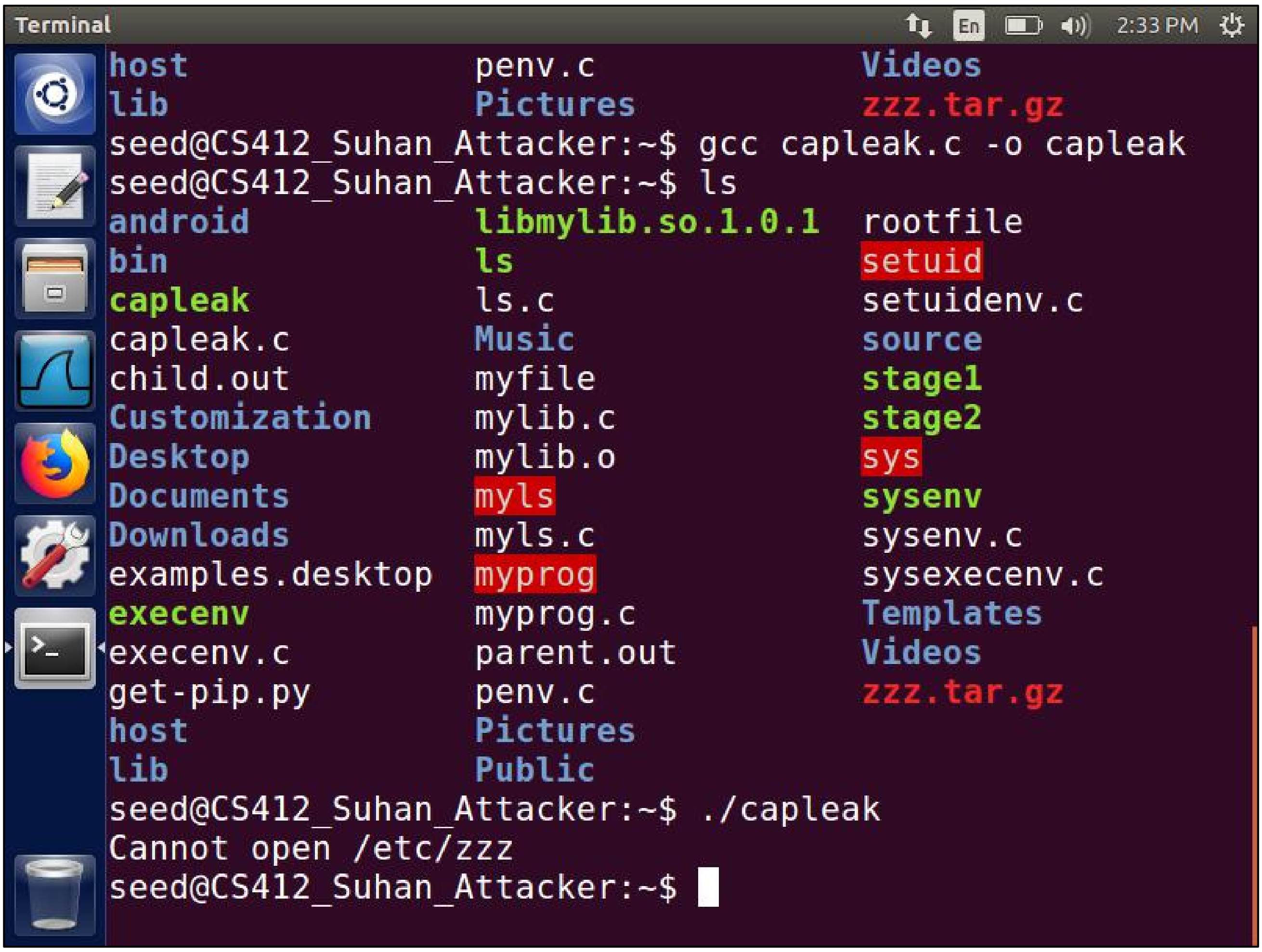
```
#include<stdio.h>
#include<stdlib.h>
#include<fcntl.h>
#include<unistd.h>
#include<sys/types.h>
void main()
{
        int fd;

        // Assume that /etc/zzz is an important system file,
        // and it is owned by root with permission 0644.
        // Before running this program, you should creat
        // the file /etc/zzz first.
        fd = open("/etc/zzz", O_RDWR | O_APPEND);
        if(fd == -1) {
            printf("Cannot open /etc/zzz\n");
            exit(0);
        }

        // Simulate the tasks conducted by the program
        sleep(1);

        // After the task, the root privilege are no longer needed,
        // it's time to relinquish the root privileges permanently.
        setuid(getuid());    // getuid() returns the real uid
        if(fork()) {     // In the parent process
           close(fd);
            exit(0);
        } else {    // in the child process
            // Now, assume that the child process is compromised, malicious attackers
```

Command:

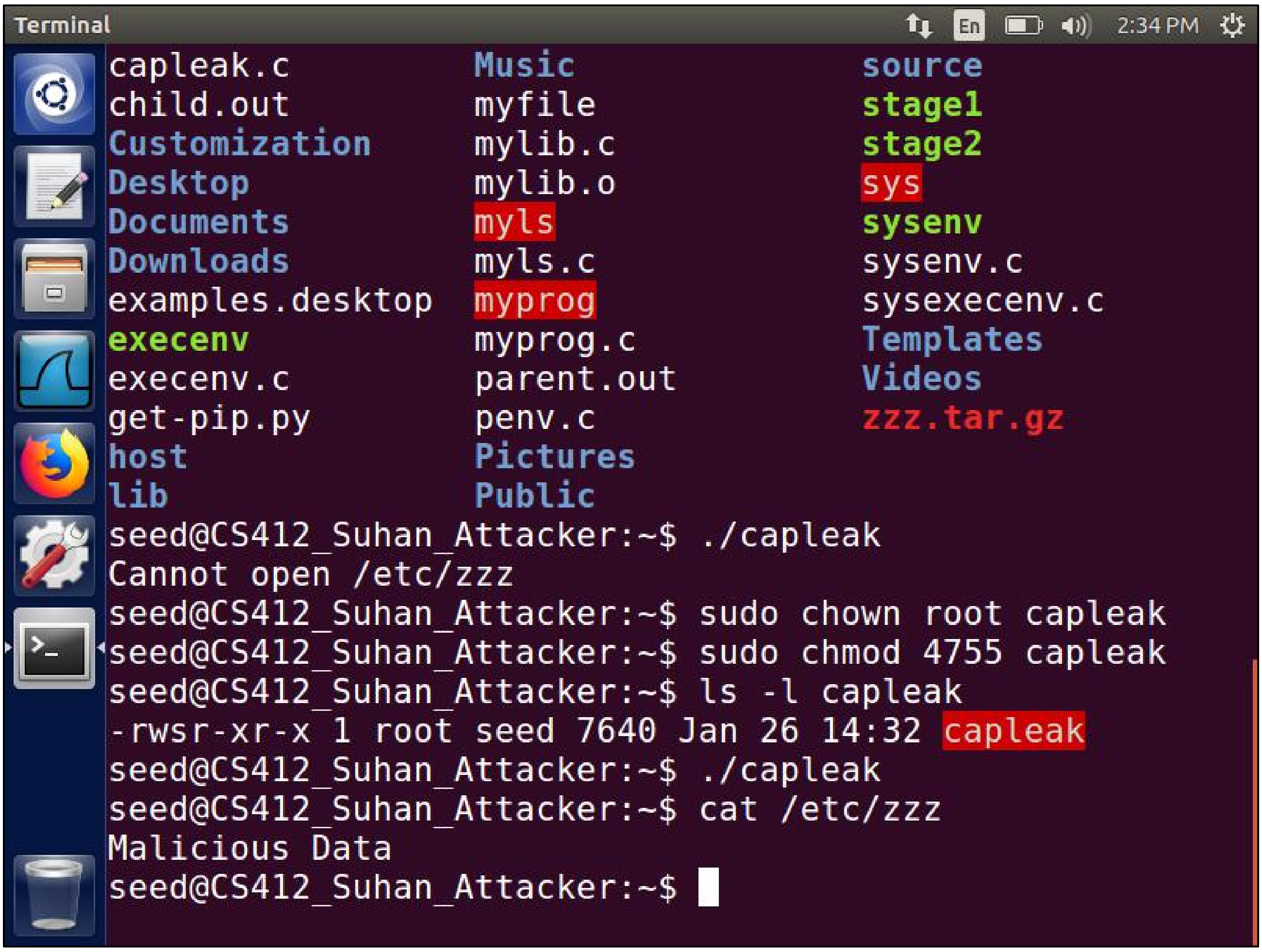
```
$gcc capleak.c -o capleak
$sudo chown root capleak
$sudo chmod 4755 capleak
$ls -l
$ capleak
$ $ cat
/etc/zzz
$./capleak
$cat /etc/zzz
```



```
seed@CS412_Suhan_Attacker:~$ sudo rm /etc/zzz
seed@CS412_Suhan_Attacker:~$ sudo touch /etc/zzz
seed@CS412_Suhan_Attacker:~$ ls -l etc/zzz
ls: cannot access 'etc/zzz': No such file or directory
seed@CS412_Suhan_Attacker:~$ cat /etc/zzz
seed@CS412_Suhan_Attacker:~$ ls
android           libmylib.so.1.0.1  Public
bin               ls                 rootfile
capleak.c         ls.c               setuid
child.out         Music              setuidenv.c
Customization     myfile             source
Desktop           mylib.c            stage1
Documents         mylib.o            stage2
Downloads         myls               sys
examples.desktop  myls.c             sysenv
execenv           myprog             sysenv.c
execenv.c         myprog.c           sysexecenv.c
get-pip.py        parent.out         Templates
host              penv.c             Videos
lib               Pictures           zzz.tar.gz
seed@CS412_Suhan_Attacker:~$ gcc capleak.c -o capleak
```



```
host              penv.c             Videos
lib               Pictures           zzz.tar.gz
seed@CS412_Suhan_Attacker:~$ gcc capleak.c -o capleak
seed@CS412_Suhan_Attacker:~$ ls
android           libmylib.so.1.0.1  rootfile
bin               ls                 setuid
capleak           ls.c               setuidenv.c
capleak.c         Music              source
child.out         myfile             stage1
Customization     mylib.c            stage2
Desktop           mylib.o            sys
Documents         myls               sysenv
Downloads         myls.c             sysenv.c
examples.desktop  myprog             sysexecenv.c
execenv           myprog.c           Templates
execenv.c         parent.out         Videos
get-pip.py        penv.c             zzz.tar.gz
host              Pictures
lib               Public
seed@CS412_Suhan_Attacker:~$ ./capleak
Cannot open /etc/zzz
seed@CS412_Suhan_Attacker:~$ 
```



```
capleak.c         Music              source
child.out         myfile             stage1
Customization     mylib.c            stage2
Desktop           mylib.o            sys
Documents         myls               sysenv
Downloads         myls.c             sysenv.c
examples.desktop  myprog             sysexecenv.c
execenv           myprog.c           Templates
execenv.c         parent.out         Videos
get-pip.py        penv.c             zzz.tar.gz
host              Pictures
lib               Public
seed@CS412_Suhan_Attacker:~$ ./capleak
Cannot open /etc/zzz
seed@CS412_Suhan_Attacker:~$ sudo chown root capleak
seed@CS412_Suhan_Attacker:~$ sudo chmod 4755 capleak
seed@CS412_Suhan_Attacker:~$ ls -l capleak
-rwsr-xr-x 1 root seed 7640 Jan 26 14:32 capleak
seed@CS412_Suhan_Attacker:~$ ./capleak
seed@CS412_Suhan_Attacker:~$ cat /etc/zzz
Malicious Data
seed@CS412_Suhan_Attacker:~$ 
```

******