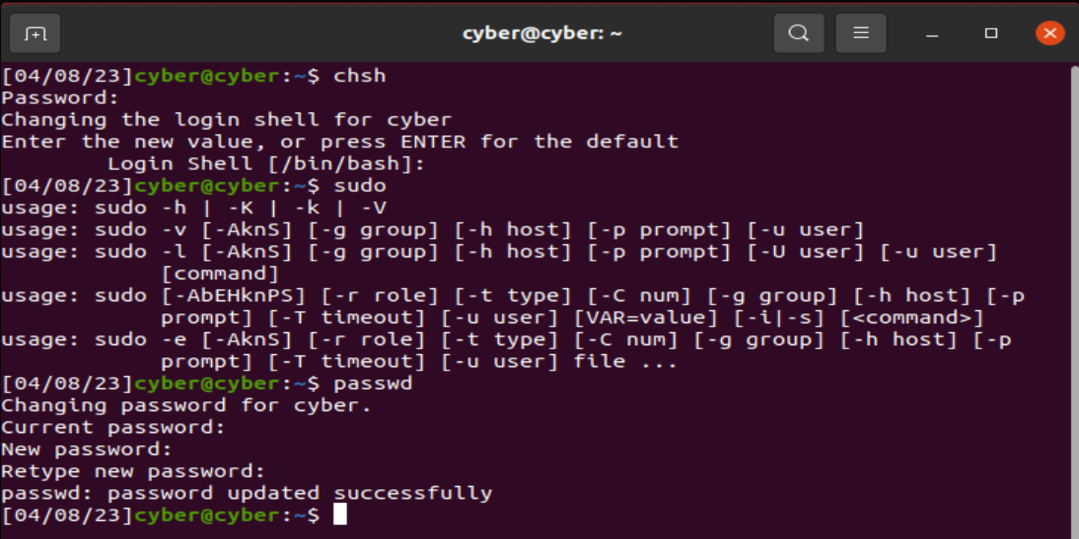
**Mini Project 3: Set-UID Program Vulnerability**

**Task 1: Explore SetUID Programs**

**Question 1.** Did the programs work appropriately in both cases? Please briefly justify your observations.

* Yes, the programs work appropriately in both cases. From my observations I can say that the Set-UID programs: **passwd**, **chsh**, and **sudo** work the same way no matter which directories they are in.



Text

Description automatically generated

**Task 2: Exploring Environment Variables**

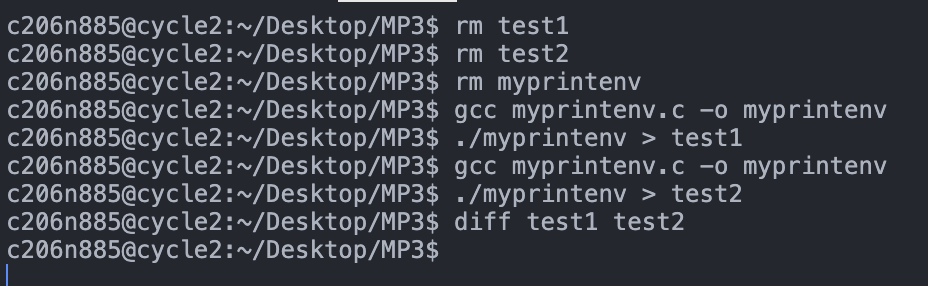
**Question 2.** Please set an environment variable called “foo” with a value of your choice, show its value, and unset it. Show your results with screenshots.

Text

Description automatically generated

**2.2 Passing Environment Variables from Parent Process to Child Process**

**Question 3.** Compare the difference of the two output files using the diff command. Please de- scribe your observations.



* After following the instruction, there was no difference in both outputs. So, we can say that the parent’s environment variables are inherited by the child process.

**2.3 Environment Variables and execve()**

**Question 4.** How does the new program get its environment variables? Please explain based on your observations.

#include <unistd.h>

extern char \*\*environ;

int main()

{

char \*argv[2];

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

argv[1] = NULL;

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

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

return 0 ;

}

* Since the original **myenv.c** file have the **execve()** function with the **envp** argument set to **NULL** which mean that the child process create by **fock()** has no access to the environment variables of the parent process. As a result, there is no output produced when the program is run.
* On the other hand, the second **execve()** function have the **environ** variable to pass the parent environment to the child process when called. This allows the child process to access the environment variables. Here the output

SHELL=/bin/bash

COLORTERM=truecolor

TERM\_PROGRAM\_VERSION=1.77.2

GTK\_MODULES=appmenu-gtk-module:appmenu-gtk-module

PWD=/home/c206n885/Desktop/MP3

KRB5CCNAME=FILE:/tmp/krb5cc\_100599096\_Jcjy9b

LOGNAME=c206n885

XDG\_SESSION\_TYPE=tty

VSCODE\_GIT\_ASKPASS\_NODE=/home/c206n885/.vscode-server/bin/e344f1f539a80912a0e9357cec841f36ce97a4e2/node

MOTD\_SHOWN=pam

HOME=/home/c206n885

LANG=en\_US.UTF-8

GIT\_ASKPASS=/home/c206n885/.vscode-server/bin/e344f1f539a80912a0e9357cec841f36ce97a4e2/extensions/git/dist/askpass.sh

SSH\_CONNECTION=194.156.136.4 51378 129.237.87.112 22

foo=abc

VSCODE\_GIT\_ASKPASS\_EXTRA\_ARGS=

XDG\_SESSION\_CLASS=user

TERM=xterm-256color

USER=c206n885

VSCODE\_GIT\_IPC\_HANDLE=/run/user/100599096/vscode-git-de526c64c0.sock

SHLVL=2

UBUNTU\_MENUPROXY=1

XDG\_SESSION\_ID=148328

XDG\_RUNTIME\_DIR=/run/user/100599096

SSH\_CLIENT=194.156.136.4 51378 22

VSCODE\_GIT\_ASKPASS\_MAIN=/home/c206n885/.vscode-server/bin/e344f1f539a80912a0e9357cec841f36ce97a4e2/extensions/git/dist/askpass-main.js

XDG\_DATA\_DIRS=/usr/share/gnome:/home/c206n885/.local/share/flatpak/exports/share:/var/lib/flatpak/exports/share:/usr/local/share:/usr/share:/var/lib/snapd/desktop

BROWSER=/home/c206n885/.vscode-server/bin/e344f1f539a80912a0e9357cec841f36ce97a4e2/bin/helpers/browser.sh

PATH=/home/c206n885/.vscode-server/bin/e344f1f539a80912a0e9357cec841f36ce97a4e2/bin/remote-cli:/opt/node-v16.17.1-linux-x64/bin/:/opt/node-v16.17.1-linux-x64/bin/:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/home/c206n885/bin:/home/c206n885/bin

DBUS\_SESSION\_BUS\_ADDRESS=unix:path=/run/user/100599096/bus

OLDPWD=/home/c206n885/Desktop

TERM\_PROGRAM=vscode

VSCODE\_IPC\_HOOK\_CLI=/run/user/100599096/vscode-ipc-ad6651a4-8018-4077-9f73-77db2ed06102.sock

\_=./myenv

**2.4 Environment Variables and system()**

**Question 5.** How does the new program /bin/sh get its environment variables? Please explain

based on your observations.

A picture containing text

Description automatically generated

* The new program /**bin/sh** gets its environment variables from the parent process that called **system().** When **system()** is called, it executes **/bin/sh** and ask the shell to execute the command. The shell process, in turn inherits the environment variables of the parent process, which is the program that called system. Since the shell process can modify or add new environment variables which will be inherited by any subsequent processes it executes. So, after **export foo=abc** being executes, it will be available and be inherited by the shell command.

**Task 3: Environment Variables and Set-UID Programs**

When a Set-UID program runs, it assumes the owner’s privileges. Therefore, Set-UID programs could result in privilege escalation. It is quite risky despite being useful in many tasks.

**3.1 Use Environment Variables to Affect Set-UID Programs**

Text

Description automatically generated

Graphical user interface, text

Description automatically generated

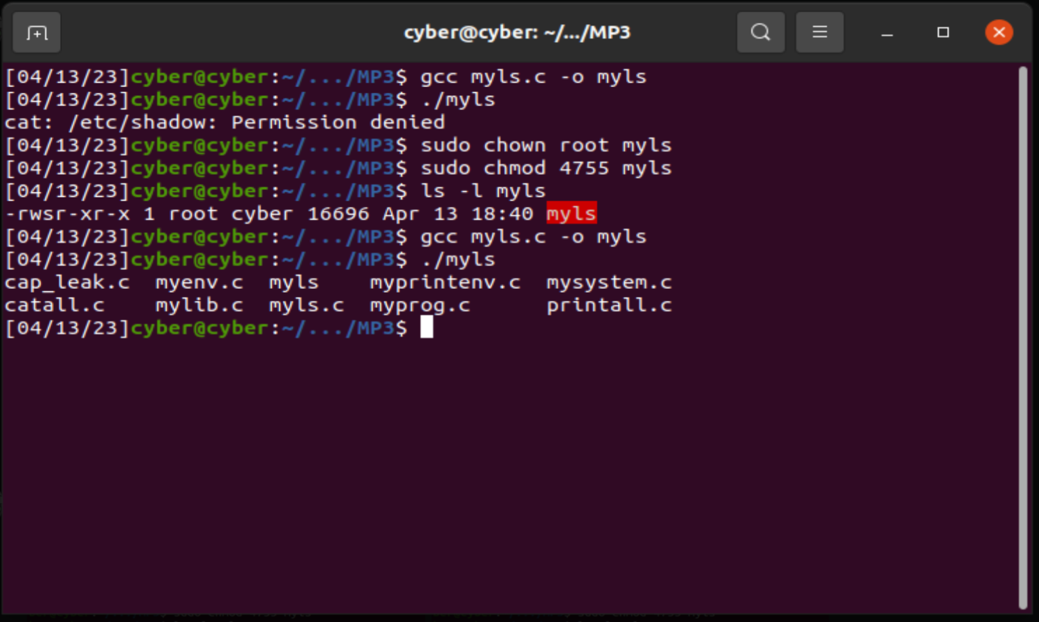
**Question 6.** Please check whether all the environment variables you set in the shell process (par- ent) get into the Set-UID child process. Describe your observation. If there are surprises to you, describe them.

* Yes, the variables that I set in the shell process (parent process) do get into the Set-UID child process. That why the output shows the same environment variables in the child process. It would look the same if I run the program as the root.

**3.2 The PATH Environment Variable**

**Question 7.** In step 3, can you get the Set-UID program to run a malicious command (such as system(”cat /etc/shadow”) or other commands of your choice)? Please report your observations (with screenshots). Are the programs running with the root privilege?

* Yes, it is possible to get the Set-UID program to run a malicious command. But it is impossible to display the shadow file. The content of what can be done is show in. the screenshot below.



* By using the command **sudo In -sf /bin/zsh/bin/sh** we can view the content of the **shadow** file.

Graphical user interface, text

Description automatically generated