National Taiwan Normal University
Computer Science and Information Engineering
Information Security: A Hands-on Approach

Instructor: Po-Wen Chi
Due Date: 09 24, 2023, PM 11:59

Assignment 1

Policies:

- Zero tolerance for late submission.
- Please pack all your submissions in one zip file. **RAR is not allowed!!**
- I only accept **PDF**. MS Word is not allowed.
- Hand-writing is not allowed.

1.1 SEED Lab (30 pts)

Environment Variable and Set-UID Lab

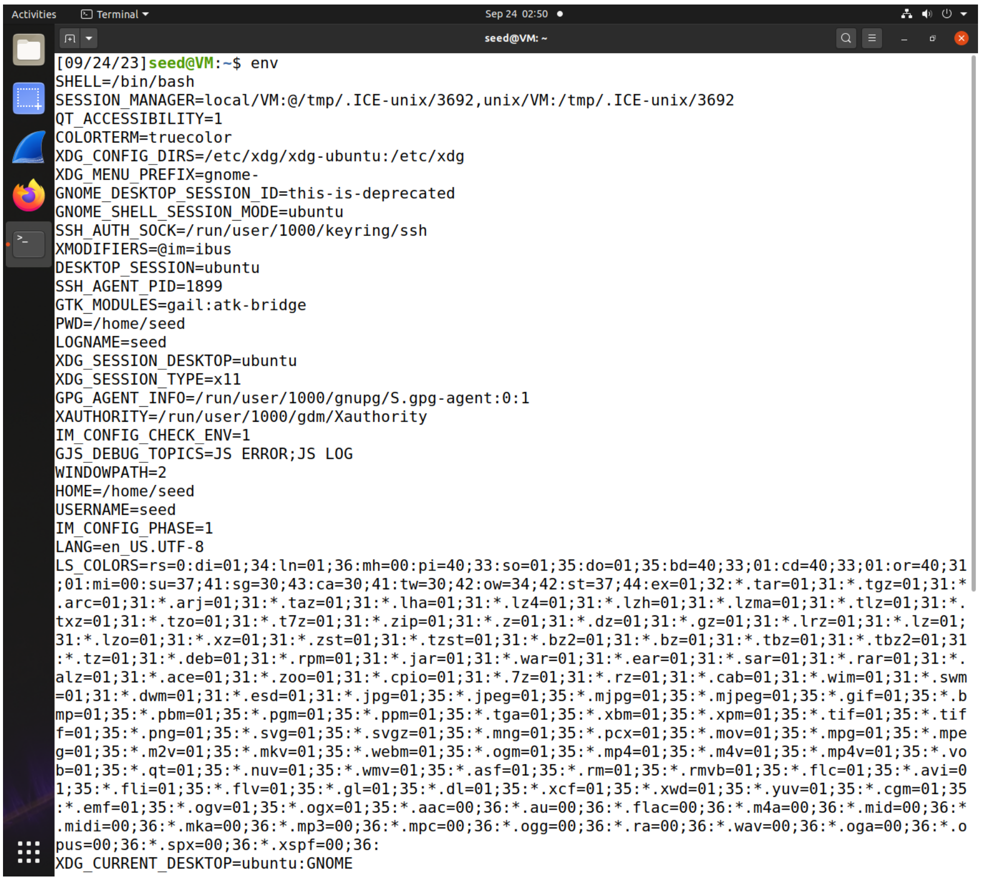
https://seedsecuritylabs.org/Labs_20.04/Software/Environment_Variable_and_SetUID/

Please record all your steps with **screen captures** and **answer all questions**.

Task 1: Manipulating Environment Variables

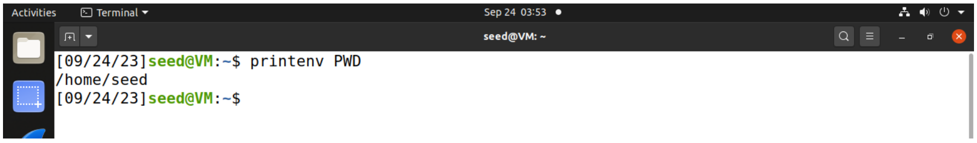
In this task, we study the commands that can be used to set and unset environment variables. We are using Bash in the seed account. The default shell that a user uses is set in the `/etc/passwd` file (the last field of each entry). You can change this to another shell program using the command `chsh` (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 "`printenv PWD`" or "`env | grep PWD`".
- Use `export` and `unset` to set or unset environment variables. It should be noted that these two commands are not separate programs; they are two of the Bash's internal commands (you will not be able to find them outside of Bash).



```
[09/24/23]seed@VM:~$ env
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/3692,unix/VM:/tmp/.ICE-unix/3692
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
XDG_MENU_PREFIX=gnome-
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
GNOME_SHELL_SESSION_MODE=ubuntu
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
XMODIFIERS=@im=ibus
DESKTOP_SESSION=ubuntu
SSH_AGENT_PID=1899
GTK_MODULES=gail:atk-bridge
PWD=/home/seed
LOGNAME=seed
XDG_SESSION_DESKTOP=ubuntu
XDG_SESSION_TYPE=x11
GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1
XAUTHORITY=/run/user/1000/gdm/Xauthority
IM_CONFIG_CHECK_ENV=1
GJS_DEBUG_TOPICS=JS ERROR;JS LOG
WINDOWPATH=2
HOME=/home/seed
USERNAME=seed
IM_CONFIG_PHASE=1
LANG=en_US.UTF-8
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31
;01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*
.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.
txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;
31:*.lzo=01;31:*.xz=01;31:*.zst=01;31:*.tzst=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31
:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.
alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.wim=01;31:*.swm
=01;31:*.dwm=01;31:*.esd=01;31:*.jpg=01;35:*.jpeg=01;35:*.mjpg=01;35:*.mjpeg=01;35:*.gif=01;35:*.b
mp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tif
f=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpe
g=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vo
b=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=0
1;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35
:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m4a=00;36:*.mid=00;36:*
.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.wav=00;36:*.oga=00;36:*.o
pus=00;36:*.spx=00;36:*.xspf=00;36:
XDG_CURRENT_DESKTOP=ubuntu:GNOME
```

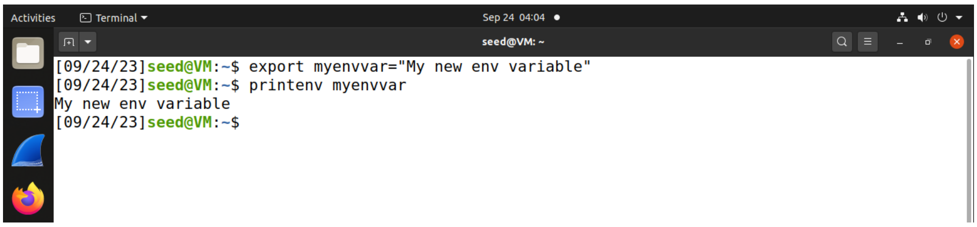
Use “env” command to print out the environment variables



```
[09/24/23]seed@VM:~$ printenv PWD
/home/seed
[09/24/23]seed@VM:~$
```

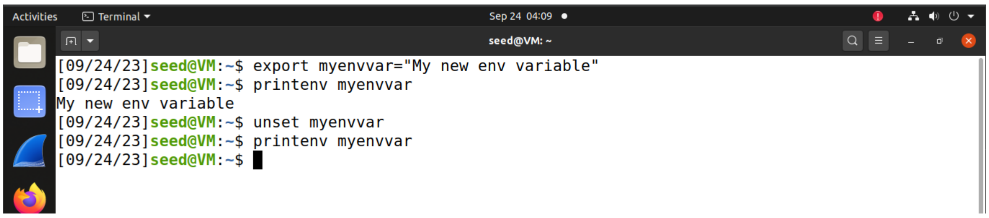
Use “printenv PWD” command, which only show the value of PWD variable

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



```
[09/24/23]seed@VM:~$ export myenvvar="My new env variable"
[09/24/23]seed@VM:~$ printenv myenvvar
My new env variable
[09/24/23]seed@VM:~$
```

Use “export” command to create an environment variable named “myenvvar”, with the value “My new env variable”. Then use “printenv” command to check the value of “myenvvar”



```
[09/24/23]seed@VM:~$ export myenvvar="My new env variable"
[09/24/23]seed@VM:~$ printenv myenvvar
My new env variable
[09/24/23]seed@VM:~$ unset myenvvar
[09/24/23]seed@VM:~$ printenv myenvvar
[09/24/23]seed@VM:~$ 
```

Use “unset” command to remove the “myenvvar” variable, and then use “printenv” command to check if ”myenvvaar” still exists. (Bash shows nothing after the unset command.)

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

In this task, we study how a child process gets its environment variables from its parent. 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`). In this task, we would like to know whether the parent's environment variables are inherited by the child process or not.

Step 1. Please compile and run the following program, and describe your observation. The program can be found in the `Labsetup` folder; it can be compiled using "`gcc myprintenv.c`", which will generate a binary called `a.out`. Let's run it and save the output into a file using "`a.out > file`".

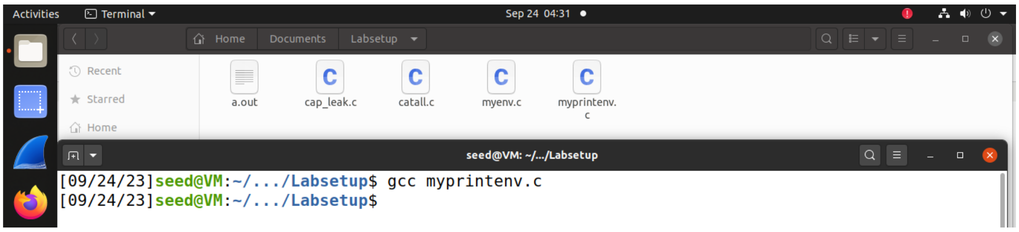
Listing 1: `myprintenv.c`

```
#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();          ①
      exit(0);
    default:  /* parent process */
      //printenv();        ②
      exit(0);
  }
}
```

Step 2. Now comment out the `printenv()` statement in the child process case (Line ①), and uncomment the `printenv()` statement in the parent process case (Line ②). Compile and run the code again, and describe your observation. Save the output in another file.

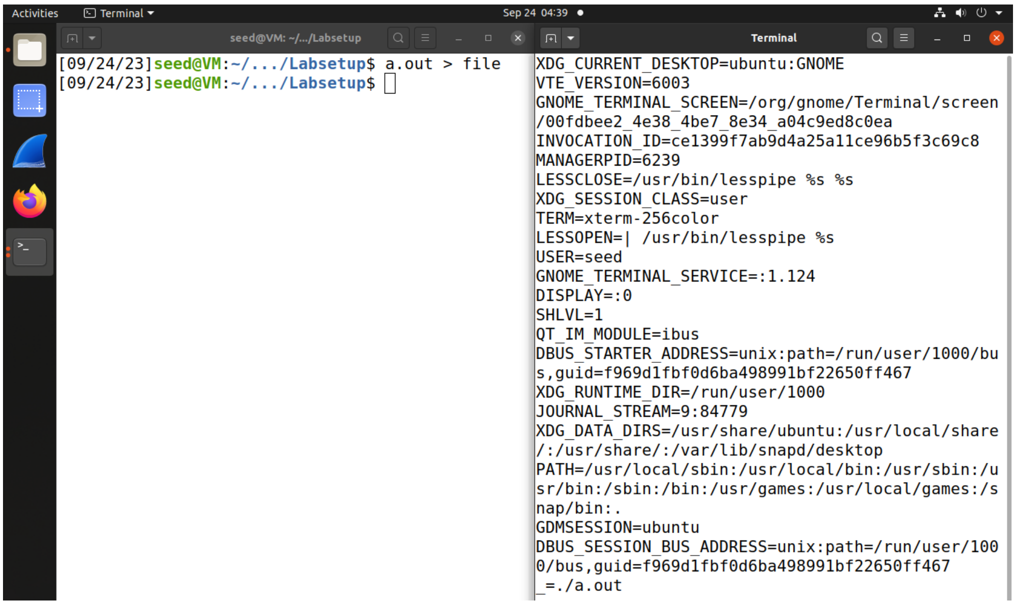
Step 3. Compare the difference of these two files using the `diff` command. Please draw your conclusion.

Step1:



```
[09/24/23]seed@VM:~/.../Labsetup$ gcc myprintenv.c
[09/24/23]seed@VM:~/.../Labsetup$
```

Use gcc compile “myprintenv.c” and generate a file called “a.out” in the Labsetup folder.

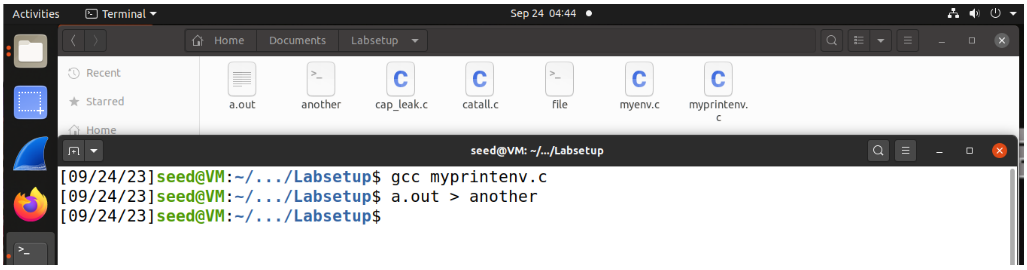


```
[09/24/23]seed@VM:~/.../Labsetup$ a.out > file
[09/24/23]seed@VM:~/.../Labsetup$
```

```
XDG_CURRENT_DESKTOP=ubuntu:GNOME
VTE_VERSION=6003
GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen
/00fdbee2_4e38_4be7_8e34_a04c9ed8c0ea
INVOCATION_ID=ce1399f7ab9d4a25a11ce96b5f3c69c8
MANAGERPID=6239
LESSCLOSE=/usr/bin/lesspipe %s %s
XDG_SESSION_CLASS=user
TERM=xterm-256color
LESSOPEN=| /usr/bin/lesspipe %s
USER=seed
GNOME_TERMINAL_SERVICE=:1.124
DISPLAY=:0
SHLVL=1
QT_IM_MODULE=ibus
DBUS_STARTER_ADDRESS=unix:path=/run/user/1000/bu
s,guid=f969d1fbf0d6ba498991bf22650ff467
XDG_RUNTIME_DIR=/run/user/1000
JOURNAL_STREAM=9:84779
XDG_DATA_DIRS=/usr/share/ubuntu:/usr/local/share
/:/usr/share/:/var/lib/snapd/desktop
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/u
sr/bin:/sbin:/bin:/usr/games:/usr/local/games:/s
nap/bin:.
GDMSESSION=ubuntu
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/100
0/bus,guid=f969d1fbf0d6ba498991bf22650ff467
_=./a.out
```

Use “a.out > file” to save the output to “file” file. The right side is the content of “file”

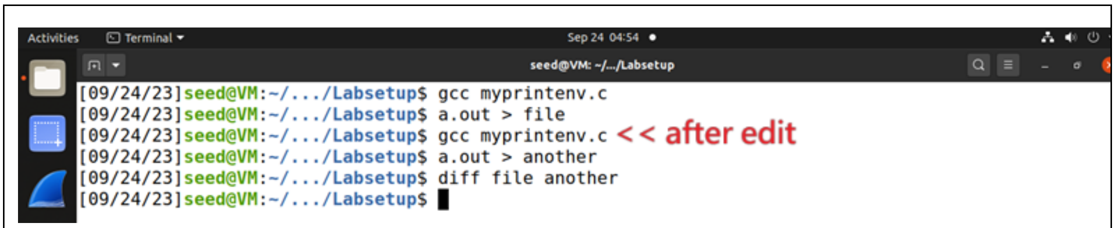
Step2



```
[09/24/23]seed@VM:~/.../Labsetup$ gcc myprintenv.c
[09/24/23]seed@VM:~/.../Labsetup$ a.out > another
[09/24/23]seed@VM:~/.../Labsetup$
```

Use gcc compile edited “myprintenv.c”. Run “a.out” and save the output to “another” file.

Step3



```
[09/24/23]seed@VM:~/.../Labsetup$ gcc myprintenv.c
[09/24/23]seed@VM:~/.../Labsetup$ a.out > file
[09/24/23]seed@VM:~/.../Labsetup$ gcc myprintenv.c
[09/24/23]seed@VM:~/.../Labsetup$ a.out > another
[09/24/23]seed@VM:~/.../Labsetup$ diff file another
[09/24/23]seed@VM:~/.../Labsetup$ 
```

<< after edit

Use “diff” command to compare the “file” and “another”.

The terminal show no different between the two files.

Since the child process inherit a copy of parent environment variables.

Task 3: Environment Variables and execve() In this task, we study

In this task, we 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. We are interested in 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.

Listing 2: `myenv.c`

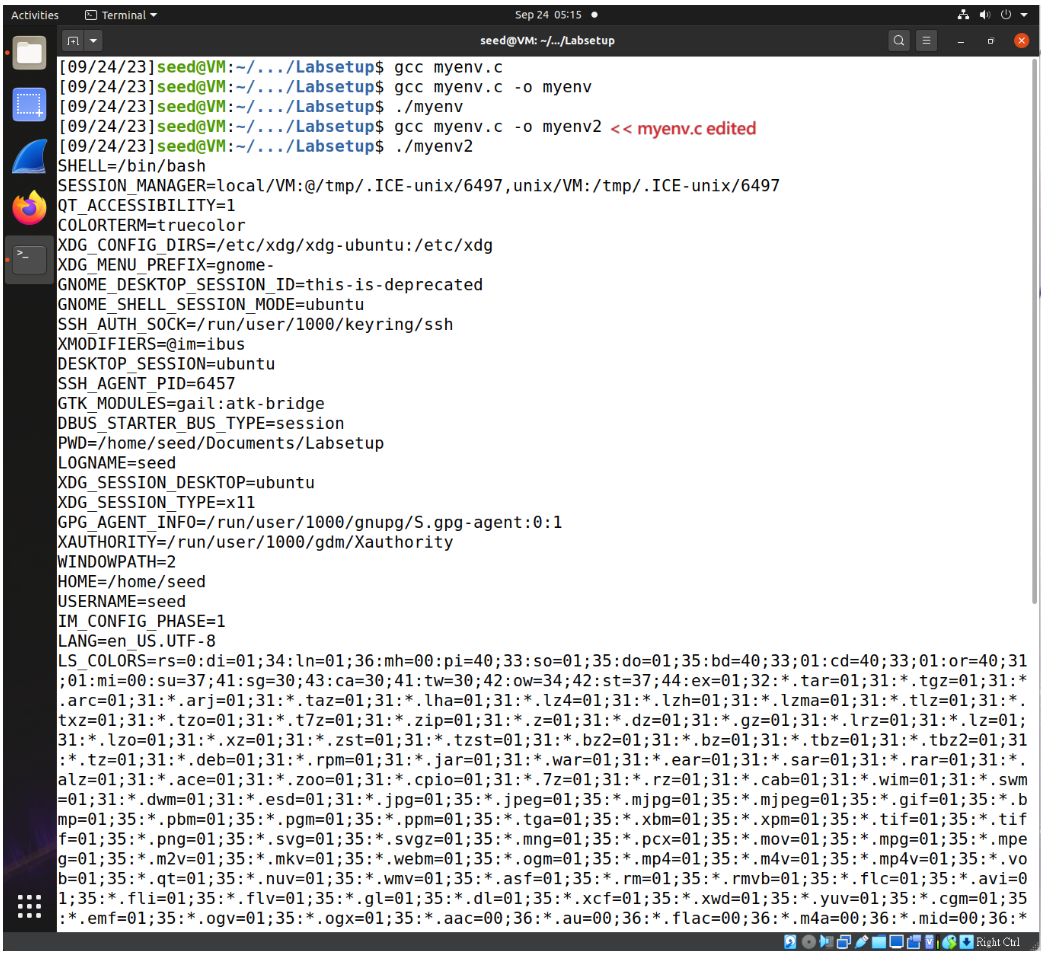
```c
#include <unistd.h>

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

  argv[0] = "/usr/bin/env";
  argv[1] = NULL;
  execve("/usr/bin/env", argv, NULL);     ①

  return 0 ;
}
```

Step 2. Change the invocation of `execve()` in Line ① to the following; describe your observation.

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

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

Step1 & Step2



```
[09/24/23]seed@VM:~/.../Labsetup$ gcc myenv.c
[09/24/23]seed@VM:~/.../Labsetup$ gcc myenv.c -o myenv
[09/24/23]seed@VM:~/.../Labsetup$ ./myenv
[09/24/23]seed@VM:~/.../Labsetup$ gcc myenv.c -o myenv2 << myenv.c edited
[09/24/23]seed@VM:~/.../Labsetup$ ./myenv2
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/6497,unix/VM:/tmp/.ICE-unix/6497
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
XDG_MENU_PREFIX=gnome-
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
GNOME_SHELL_SESSION_MODE=ubuntu
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
XMODIFIERS=@im=ibus
DESKTOP_SESSION=ubuntu
SSH_AGENT_PID=6457
GTK_MODULES=gail:atk-bridge
DBUS_STARTER_BUS_TYPE=session
PWD=/home/seed/Documents/Labsetup
LOGNAME=seed
XDG_SESSION_DESKTOP=ubuntu
XDG_SESSION_TYPE=x11
GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1
XAUTHORITY=/run/user/1000/gdm/Xauthority
WINDOWPATH=2
HOME=/home/seed
USERNAME=seed
IM_CONFIG_PHASE=1
LANG=en_US.UTF-8
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31
;01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*
.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.
txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;
31:*.lzo=01;31:*.xz=01;31:*.zst=01;31:*.tzst=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31
:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.
alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.wim=01;31:*.swm
=01;31:*.dwm=01;31:*.esd=01;31:*.jpg=01;35:*.jpeg=01;35:*.mjpg=01;35:*.mjpeg=01;35:*.gif=01;35:*.b
mp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tif
f=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpe
g=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vo
b=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=0
1;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35
:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m4a=00;36:*.mid=00;36:*
```

Use gcc compile myenv.c and name output file as “myenv”

Use gcc compile edited myenv.c and name output file as “myenv2”

myenv no output, and myenv2 show the environment variables.

Step3

The unedited mynev.c (line12: `execve("/usr/bin/env", argv, NULL);`)

NULL is used as the third argument of “execve”, resulting in no environment variables being passed to the new process.

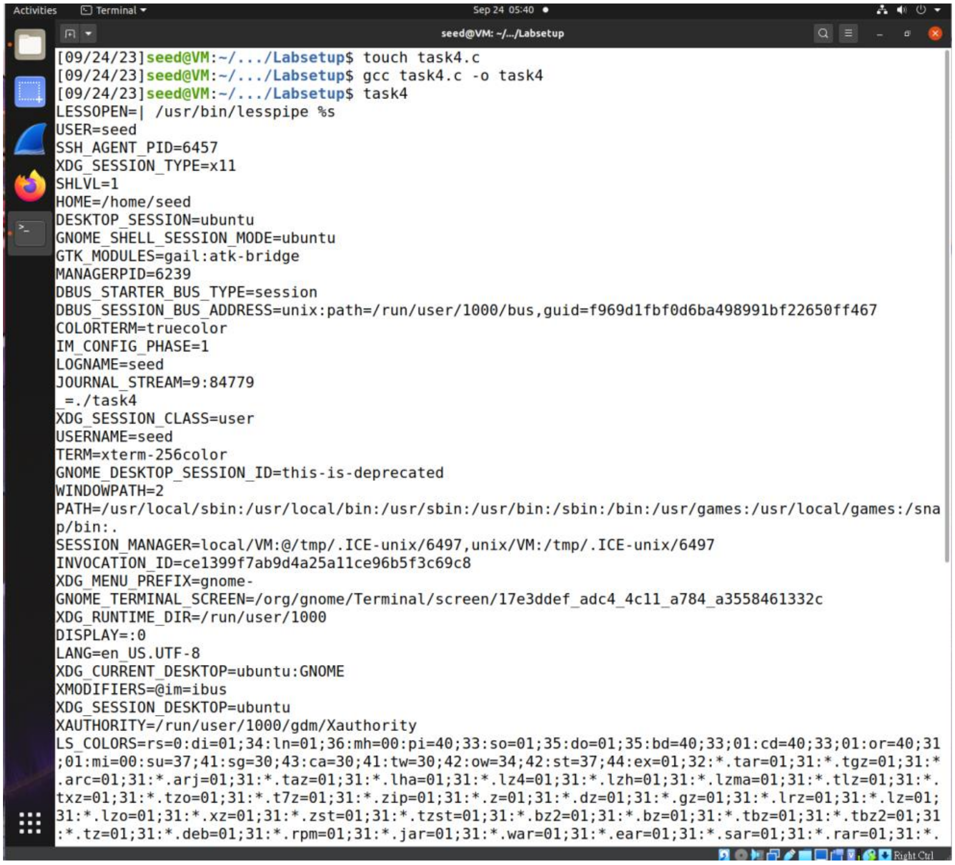
After edited (line12: `execve("/usr/bin/env", argv, eviron);`) “eviron” is used as the third argument of execve, so can the new process get the environment variable which the current process keep.

Task 4: Environment Variables and system()

In this task, we 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`; `execl()` calls `execve()`, passing to it the environment variables array. Therefore, using `system()`, the environment variables of the calling process is 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 ;
}
```



```
[09/24/23]seed@VM:~/.../Labsetup$ touch task4.c
[09/24/23]seed@VM:~/.../Labsetup$ gcc task4.c -o task4
[09/24/23]seed@VM:~/.../Labsetup$ task4
LESSOPEN=| /usr/bin/lesspipe %s
USER=seed
SSH_AGENT_PID=6457
XDG_SESSION_TYPE=x11
SHLVL=1
HOME=/home/seed
DESKTOP_SESSION=ubuntu
GNOME_SHELL_SESSION_MODE=ubuntu
GTK_MODULES=gail:atk-bridge
MANAGERPID=6239
DBUS_STARTER_BUS_TYPE=session
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus,guid=f969d1fbf0d6ba498991bf22650ff467
COLORTERM=truecolor
IM_CONFIG_PHASE=1
LOGNAME=seed
JOURNAL_STREAM=9:84779
_=./task4
XDG_SESSION_CLASS=user
USERNAME=seed
TERM=xterm-256color
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
WINDOWPATH=2
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/sna
p/bin:.
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/6497,unix/VM:/tmp/.ICE-unix/6497
INVOCATION_ID=ce1399f7ab9d4a25a11ce96b5f3c69c8
XDG_MENU_PREFIX=gnome-
GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/17e3ddef_adc4_4c11_a784_a3558461332c
XDG_RUNTIME_DIR=/run/user/1000
DISPLAY=:0
LANG=en_US.UTF-8
XDG_CURRENT_DESKTOP=ubuntu:GNOME
XMODIFIERS=@im=ibus
XDG_SESSION_DESKTOP=ubuntu
XAUTHORITY=/run/user/1000/gdm/Xauthority
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31
;01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*
.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.
txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;
31:*.lzo=01;31:*.xz=01;31:*.zst=01;31:*.tzst=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31
:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.
```

Use touch command to create task4.c and write the given code in it.

Use gcc compile task4.c and run.

The output show the environment variables of the current process.

Since the system function pass the environment variables to /bin/sh implicitly

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, 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 since it escalates the user's privilege, it is 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. Write the following program that can print out all the environment variables in the current process.

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

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

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

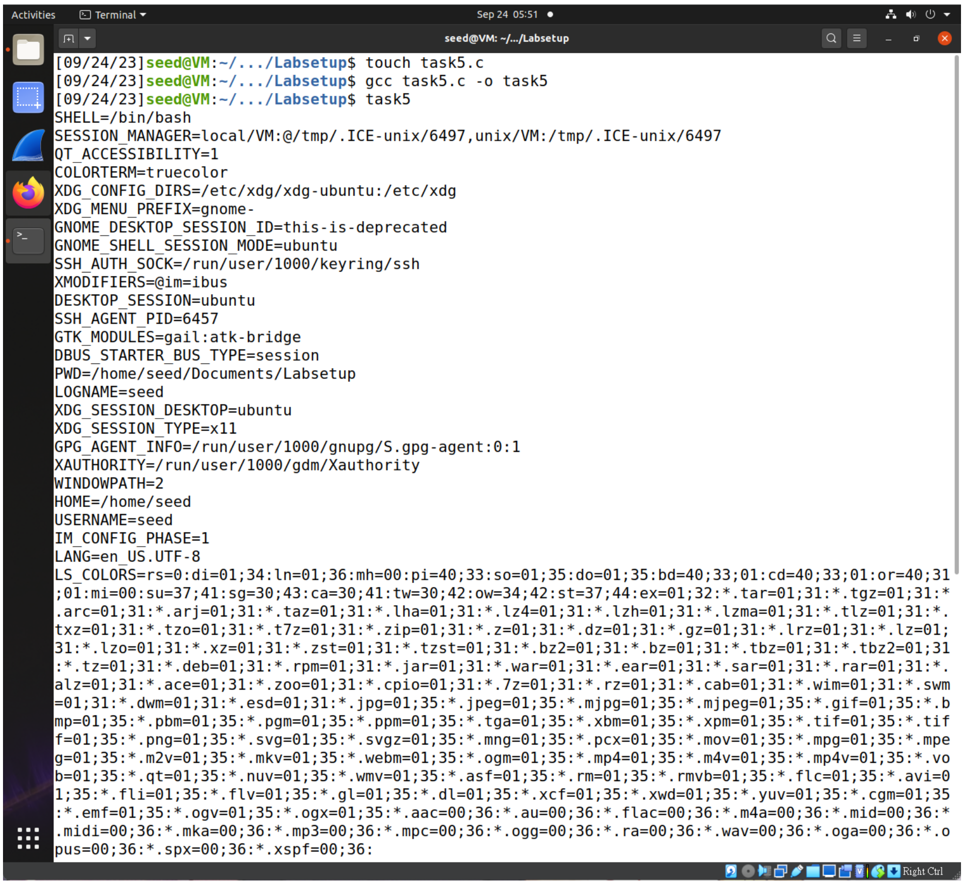
```
// Asssume the program's name is foo
$ sudo chown root foo
$ sudo chmod 4755 foo
```

Step 3. In your 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).

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.

Step1



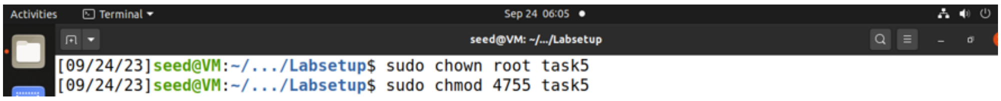
```
[09/24/23]seed@VM:~/.../Labsetup$ touch task5.c
[09/24/23]seed@VM:~/.../Labsetup$ gcc task5.c -o task5
[09/24/23]seed@VM:~/.../Labsetup$ task5
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/6497,unix/VM:/tmp/.ICE-unix/6497
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
XDG_MENU_PREFIX=gnome-
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
GNOME_SHELL_SESSION_MODE=ubuntu
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
XMODIFIERS=@im=ibus
DESKTOP_SESSION=ubuntu
SSH_AGENT_PID=6457
GTK_MODULES=gail:atk-bridge
DBUS_STARTER_BUS_TYPE=session
PWD=/home/seed/Documents/Labsetup
LOGNAME=seed
XDG_SESSION_DESKTOP=ubuntu
XDG_SESSION_TYPE=x11
GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1
XAUTHORITY=/run/user/1000/gdm/Xauthority
WINDOWPATH=2
HOME=/home/seed
USERNAME=seed
IM_CONFIG_PHASE=1
LANG=en_US.UTF-8
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31
;01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*
.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.
txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;
31:*.lzo=01;31:*.xz=01;31:*.zst=01;31:*.tzst=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31
:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.
alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.wim=01;31:*.swm
=01;31:*.dwm=01;31:*.esd=01;31:*.jpg=01;35:*.jpeg=01;35:*.mjpg=01;35:*.mjpeg=01;35:*.gif=01;35:*.b
mp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tif
f=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpe
g=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vo
b=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=0
1;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35
:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m4a=00;36:*.mid=00;36:*
.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.wav=00;36:*.oga=00;36:*.o
pus=00;36:*.spx=00;36:*.xspf=00;36:
```

Use touch command to create task4.c and write the given code in it.

Use gcc compile task4.c and run.

The output show the environment variables of the current process.

Step 2

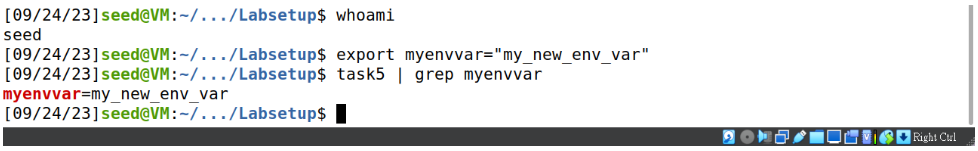


```
[09/24/23]seed@VM:~/.../Labsetup$ sudo chown root task5
[09/24/23]seed@VM:~/.../Labsetup$ sudo chmod 4755 task5
```

"sudo chown root task5" making the owner of task5 is root

"sudo chmod 4755 task5" making the program a SET-UID program

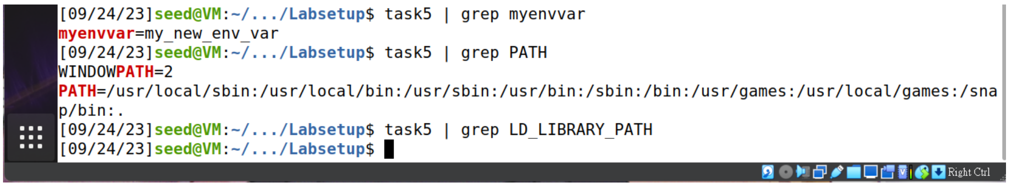
Step3



```
[09/24/23]seed@VM:~/.../Labsetup$ whoami
seed
[09/24/23]seed@VM:~/.../Labsetup$ export myenvvar="my_new_env_var"
[09/24/23]seed@VM:~/.../Labsetup$ task5 | grep myenvvar
myenvvar=my_new_env_var
[09/24/23]seed@VM:~/.../Labsetup$
```

Use whoami to check current user

Use “export” command to create an environment variable named “myenvvar”.



```
[09/24/23]seed@VM:~/.../Labsetup$ task5 | grep myenvvar
myenvvar=my_new_env_var
[09/24/23]seed@VM:~/.../Labsetup$ task5 | grep PATH
WINDOWPATH=2
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/sna
p/bin:.
[09/24/23]seed@VM:~/.../Labsetup$ task5 | grep LD_LIBRARY_PATH
[09/24/23]seed@VM:~/.../Labsetup$
```

task5 | grep myenvar, check “myenvvar” if exist -> Yes

task5 | grep PATH, check "PATH" if exist -> Yes

task5 | grep LD_LIBRARY_PATH, check "LD_LIBRARY_PATH" if exist -> No

LD_LIBRARY_PATH is missing, since real user id and effective user id are different.

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:

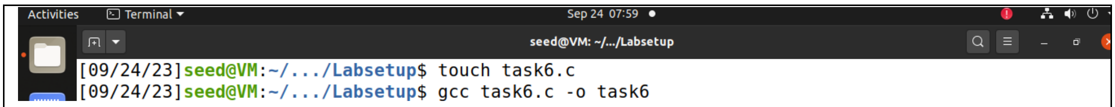
```
int main()
{
  system("ls");
  return 0;
}
```

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

Note: The `system(cmd)` function executes the `/bin/sh` program first, and then asks this shell program to run the `cmd` command. In Ubuntu 20.04 (and several versions before), `/bin/sh` is actually a symbolic link pointing to `/bin/dash`. This shell program has a countermeasure that prevents itself from being executed in a `Set-UID` process. Basically, if `dash` detects that it is executed in a `Set-UID` process, it immediately changes the effective user ID to the process's real user ID, essentially dropping the privilege.

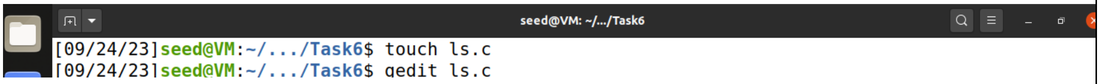
Since our victim program is a `Set-UID` program, the countermeasure in `/bin/dash` can prevent our attack. To see how our attack works without such a countermeasure, we will link `/bin/sh` to another shell that does not have such a countermeasure. We have installed a shell program called `zsh` in our Ubuntu 20.04 VM. We use the following commands to link `/bin/sh` to `/bin/zsh`:

```
$ sudo ln -sf /bin/zsh /bin/sh
```



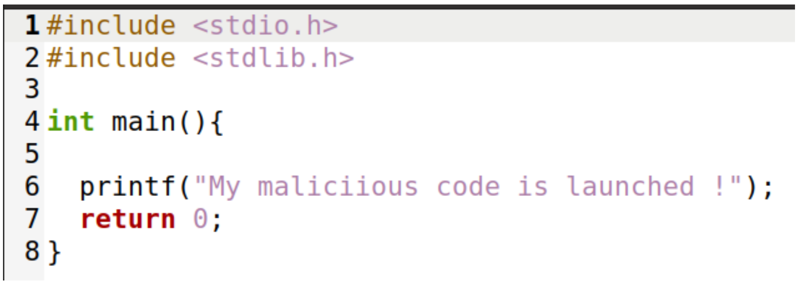
```
[09/24/23]seed@VM:~/.../Labsetup$ touch task6.c
[09/24/23]seed@VM:~/.../Labsetup$ gcc task6.c -o task6
```

Create a file "task6.c" (compile the code above), compiled by gcc)



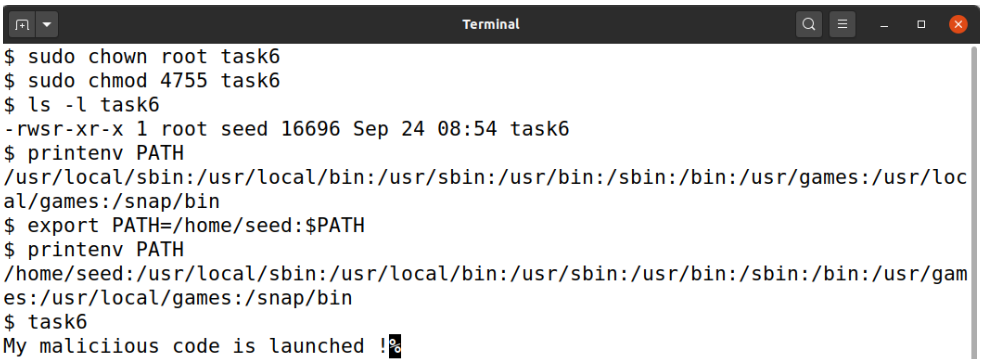
```
[09/24/23]seed@VM:~/.../Task6$ touch ls.c
[09/24/23]seed@VM:~/.../Task6$ gedit ls.c
```

Create a program used as "malicious code"



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

int main(){

  printf("My maliciious code is launched !");
  return 0;
}
```

(I have moved all the needed file to “/home/seed”)



```
$ sudo chown root task6
$ sudo chmod 4755 task6
$ ls -l task6
-rwsr-xr-x 1 root seed 16696 Sep 24 08:54 task6
$ printenv PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/loc
al/games:/snap/bin
$ export PATH=/home/seed:$PATH
$ printenv PATH
/home/seed:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/gam
es:/usr/local/games:/snap/bin
$ task6
My maliciious code is launched !%
```

Use sudo ln -sf /bin/zsh /bin/sh link sh to zsh

Change its owner to root, make it a Set-UID program and check whether it is a Set-UID program.

Try to run my code instead of “ls” program, I changed the value of environment. Variable PATH. (making the program to search for the file in my folder first)

After run task 6, the output is “My malicious code is launched !”

It run my code instead of the “/bin/ls” program !!

Conclusion:

Changing the PATH to a specific folder may execute hacker’s code.

Using system(“…”) with relative path in a Set-UID program is dangerous, allowing attacker perform his code.

Task 7 The LD_PRELOAD Environment Variable and Set-UID program

In this task, we 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 run time.

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 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, we will only study `LD_PRELOAD`.

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

1. Let us 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. We can compile the above program using the following commands (in the `-lc` argument, the second character is ℓ):

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

3. Now, set the `LD_PRELOAD` environment variable:

```
$ export LD_PRELOAD=./libmylib.so.1.0.1
```

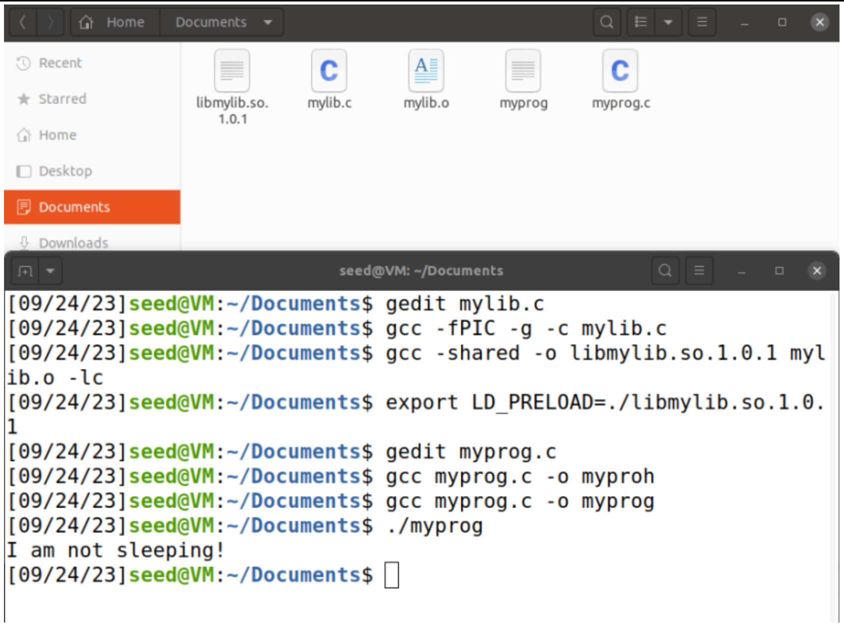
4. Finally, compile the following program `myprog`, and in the same directory as the above dynamic link library `libmylib.so.1.0.1`:

```
/* myprog.c */
#include <unistd.h>
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.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.

Step 1



```
[09/24/23]seed@VM:~/Documents$ gedit mylib.c
[09/24/23]seed@VM:~/Documents$ gcc -fPIC -g -c mylib.c
[09/24/23]seed@VM:~/Documents$ gcc -shared -o libmylib.so.1.0.1 myl
ib.o -lc
[09/24/23]seed@VM:~/Documents$ export LD_PRELOAD=./libmylib.so.1.0.
1
[09/24/23]seed@VM:~/Documents$ gedit myprog.c
[09/24/23]seed@VM:~/Documents$ gcc myprog.c -o myproh
[09/24/23]seed@VM:~/Documents$ gcc myprog.c -o myprog
[09/24/23]seed@VM:~/Documents$ ./myprog
I am not sleeping!
[09/24/23]seed@VM:~/Documents$ 
```

Create a file mylib.c and compiled by gcc

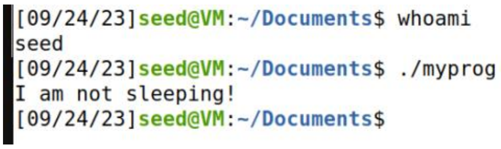
-fIPC means position independent code

-g means output debugging information

-c means compile the file but not link it

-shared means a shared object which can be linked to other object

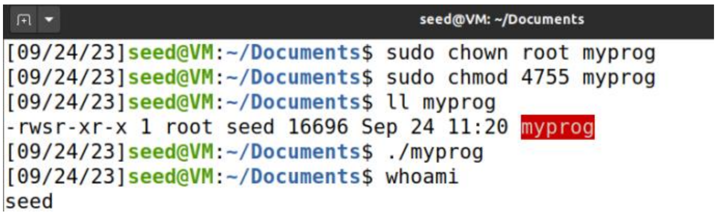
Step2



```
[09/24/23]seed@VM:~/Documents$ whoami
seed
[09/24/23]seed@VM:~/Documents$ ./myprog
I am not sleeping!
[09/24/23]seed@VM:~/Documents$
```

A regular program “myprog”, run it as a normal user

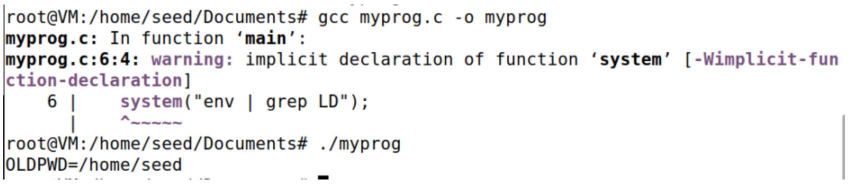
Output “I am not sleeping”



```
[09/24/23]seed@VM:~/Documents$ sudo chown root myprog
[09/24/23]seed@VM:~/Documents$ sudo chmod 4755 myprog
[09/24/23]seed@VM:~/Documents$ ll myprog
-rwsr-xr-x 1 root seed 16696 Sep 24 11:20 myprog
[09/24/23]seed@VM:~/Documents$ ./myprog
[09/24/23]seed@VM:~/Documents$ whoami
seed
```

Make it a Set-UID program, and run it as a normal user

The terminal stuck 1 second then output nothing.



```
root@VM:/home/seed/Documents# gcc myprog.c -o myprog
myprog.c: In function ‘main’:
myprog.c:6:4: warning: implicit declaration of function ‘system’ [-Wimplicit-fun
ction-declaration]
    6 |    system("env | grep LD");
      |    ^~~~~~
root@VM:/home/seed/Documents# ./myprog
OLDPWD=/home/seed
```

Make it a Set-UID program, and run it as root

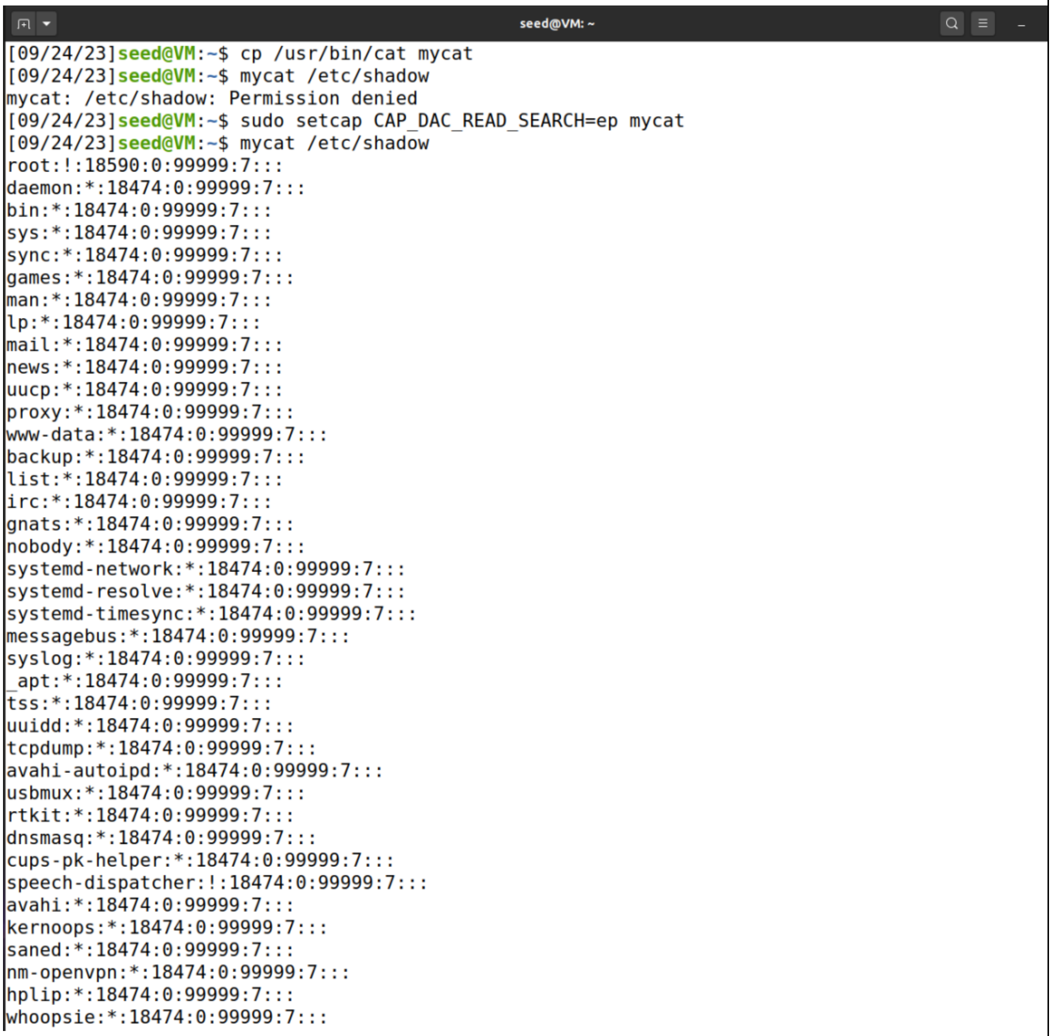
1.2 Capabilities (15 pts)

Please run the following sequence of commands.

```
$ cp /usr/bin/cat mycat
$ mycat /etc/shadow
$ sudo setcap CAP_DAC_READ_SEARCH=ep mycat
$ mycat /etc/shadow
```

1. What are results?
2. Please explain the meaning of the third line.
3. How to check the capability of a file? Please use **/usr/bin/ping** as an example and describe its capabilities.

1. What are results



```
[09/24/23]seed@VM:~$ cp /usr/bin/cat mycat
[09/24/23]seed@VM:~$ mycat /etc/shadow
mycat: /etc/shadow: Permission denied
[09/24/23]seed@VM:~$ sudo setcap CAP_DAC_READ_SEARCH=ep mycat
[09/24/23]seed@VM:~$ mycat /etc/shadow
root:!:18590:0:99999:7:::
daemon:*:18474:0:99999:7:::
bin:*:18474:0:99999:7:::
sys:*:18474:0:99999:7:::
sync:*:18474:0:99999:7:::
games:*:18474:0:99999:7:::
man:*:18474:0:99999:7:::
lp:*:18474:0:99999:7:::
mail:*:18474:0:99999:7:::
news:*:18474:0:99999:7:::
uucp:*:18474:0:99999:7:::
proxy:*:18474:0:99999:7:::
www-data:*:18474:0:99999:7:::
backup:*:18474:0:99999:7:::
list:*:18474:0:99999:7:::
irc:*:18474:0:99999:7:::
gnats:*:18474:0:99999:7:::
nobody:*:18474:0:99999:7:::
systemd-network:*:18474:0:99999:7:::
systemd-resolve:*:18474:0:99999:7:::
systemd-timesync:*:18474:0:99999:7:::
messagebus:*:18474:0:99999:7:::
syslog:*:18474:0:99999:7:::
_apt:*:18474:0:99999:7:::
tss:*:18474:0:99999:7:::
uuidd:*:18474:0:99999:7:::
tcpdump:*:18474:0:99999:7:::
avahi-autoipd:*:18474:0:99999:7:::
usbmux:*:18474:0:99999:7:::
rtkit:*:18474:0:99999:7:::
dnsmasq:*:18474:0:99999:7:::
cups-pk-helper:*:18474:0:99999:7:::
speech-dispatcher:!:18474:0:99999:7:::
avahi:*:18474:0:99999:7:::
kernoops:*:18474:0:99999:7:::
saned:*:18474:0:99999:7:::
nm-openvpn:*:18474:0:99999:7:::
hplip:*:18474:0:99999:7:::
whoopsie:*:18474:0:99999:7:::
```

2. Please explain the meaning of the third line.

Set capability on the “mycat” with superuser permission.

Set ‘CAP_DAC_READ_SEARCH’ to ep (effective & permit)

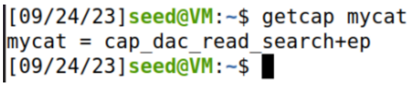
CAP_DAC_READ_SEARCH : Bypass file read-only operation checks

“mycat” now can read the file although he/she is not root

3. How to check the capability of a file?

Use ‘getcap’ command

Ex:



```
[09/24/23]seed@VM:~$ getcap mycat
mycat = cap_dac_read_search+ep
[09/24/23]seed@VM:~$ 
```