The Laboratory of Information Security (UE19CS346)

Documented by Anurag.R.Simha

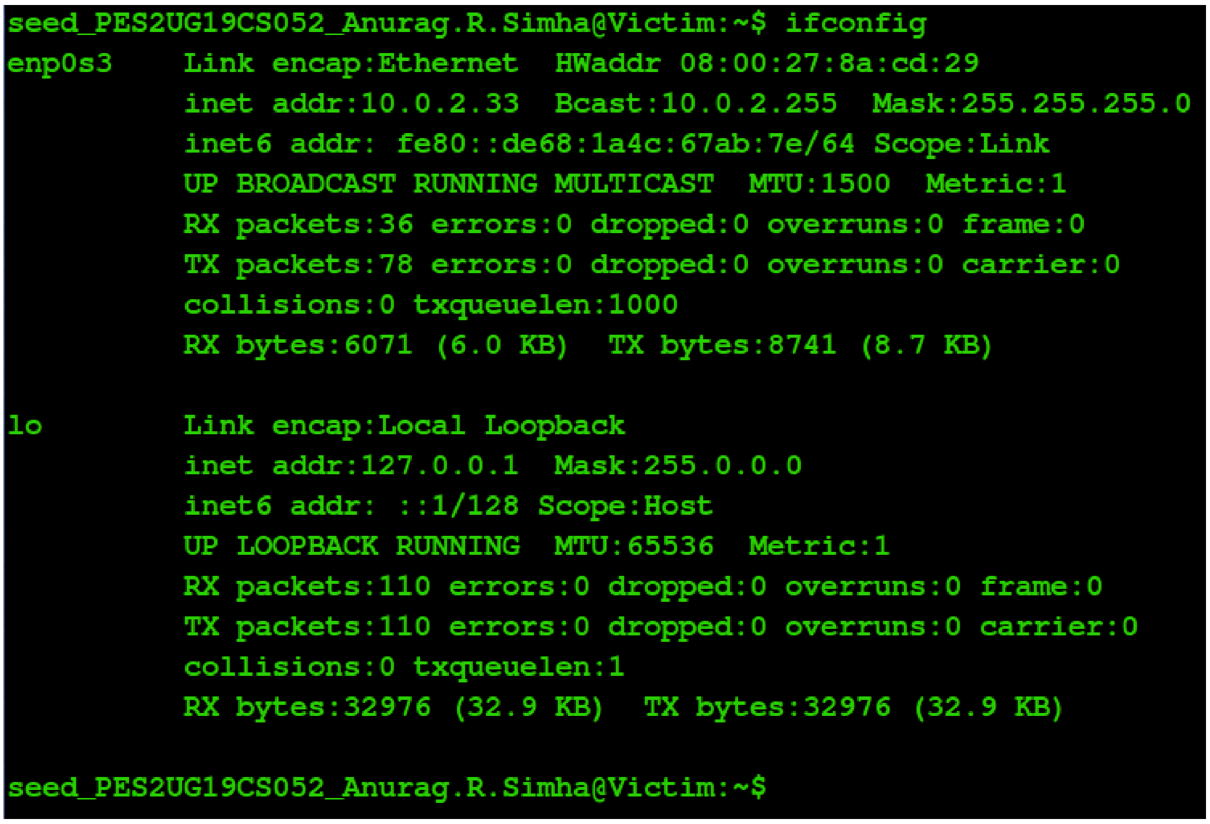
SRN	:	PES2UG19CS052
Name	:	Anurag.R.Simha
Date	:	24/01/2022
Section	:	A
Week	:	1

The Table of Contents

The Setup

For the experimentation of various attacks, a single virtual machine was employed.

1. The Victim/Client machine (10.0.2.13)



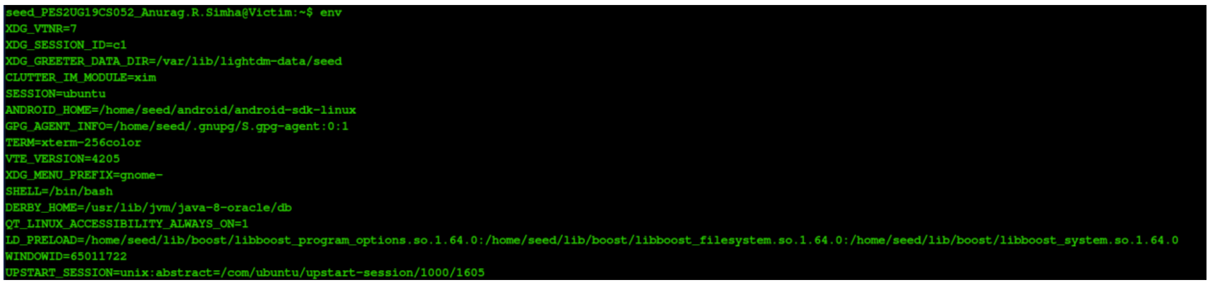
```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:8a:cd:29
          inet addr:10.0.2.33  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::de68:1a4c:67ab:7e/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:36 errors:0 dropped:0 overruns:0 frame:0
          TX packets:78 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:6071 (6.0 KB)  TX bytes:8741 (8.7 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:110 errors:0 dropped:0 overruns:0 frame:0
          TX packets:110 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:32976 (32.9 KB)  TX bytes:32976 (32.9 KB)

seed_PES2UG19CS052_Anurag.R.Simha@Victim:~$
```

Task 1: Manipulating environment variables

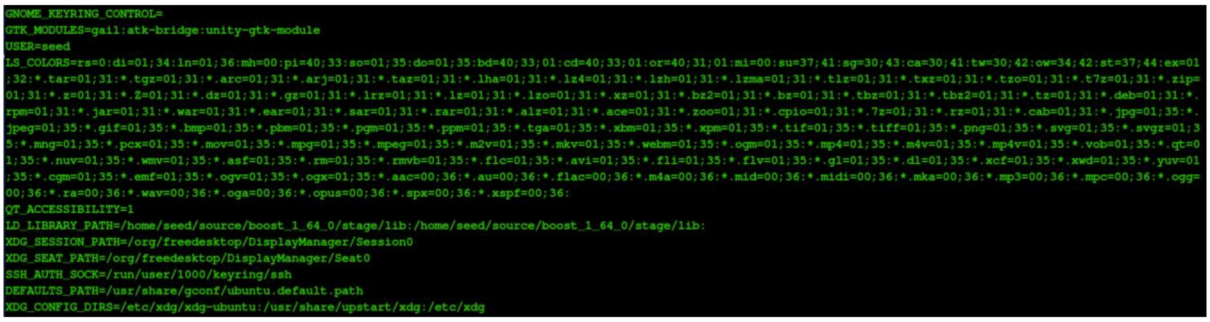
In this task, the commands that can be used to set and unset environment variables are studied. Here, it's done using Bash in the seed account. The default shell that a user makes use of is set in the `/etc/passwd` file (the last field of each entry). It can be changed to another shell program using the command `chsh`.

a) To inspect the environment variables, the command, `env` is run.

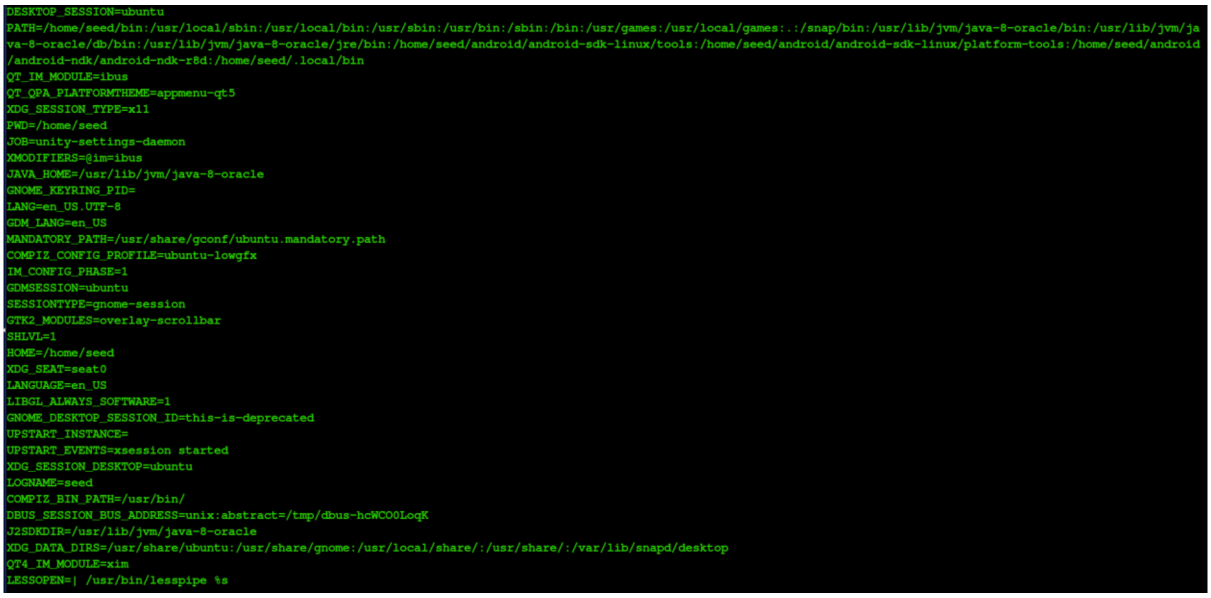
The command: `env`



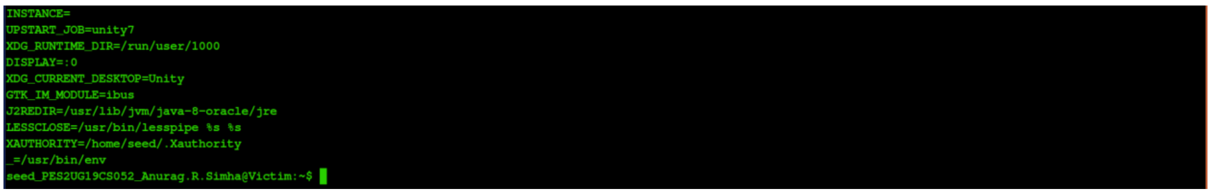
```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~$ env
XDG_VTNR=7
XDG_SESSION_ID=c1
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed
CLUTTER_IM_MODULE=xim
SESSION=ubuntu
ANDROID_HOME=/home/seed/android/android-sdk-linux
GPG_AGENT_INFO=/home/seed/.gnupg/S.gpg-agent:0:1
TERM=xterm-256color
VTE_VERSION=4205
XDG_MENU_PREFIX=gnome-
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_options.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=65011722
UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session/1000/1605
```



```
GNOME_KEYRING_CONTROL=
GTK_MODULES=gail:atk-bridge:unity-gtk-module
USER=seed
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:*.Z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=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:*.jpg=01;35:*.
jpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;3
5:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=0
1;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;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:*.opus=00;36:*.spx=00;36:*.xspf=00;36:
QT_ACCESSIBILITY=1
LD_LIBRARY_PATH=/home/seed/source/boost_1_64_0/stage/lib:/home/seed/source/boost_1_64_0/stage/lib:
XDG_SESSION_PATH=/org/freedesktop/DisplayManager/Session0
XDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat0
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
DEFAULTS_PATH=/usr/share/gconf/ubuntu.default.path
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/usr/share/upstart/xdg:/etc/xdg
```



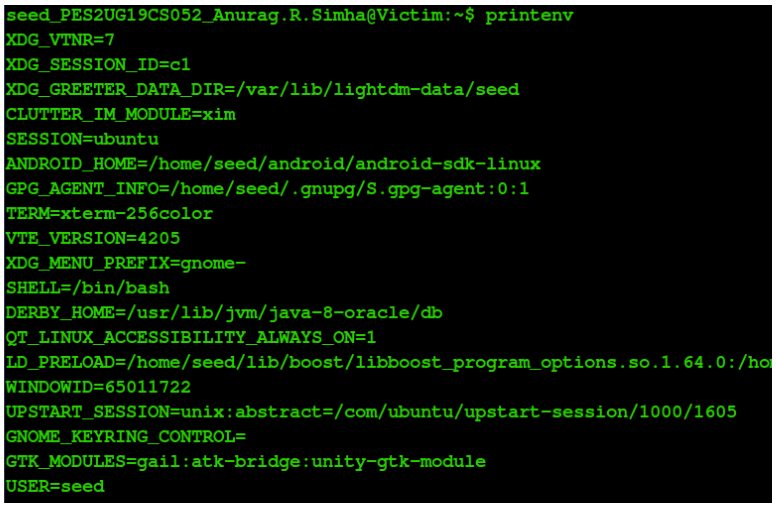
```
DESKTOP_SESSION=ubuntu
PATH=/home/seed/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:.:/snap/bin:/usr/lib/jvm/java-8-oracle/bin:/usr/lib/jvm/ja
va-8-oracle/db/bin:/usr/lib/jvm/java-8-oracle/jre/bin:/home/seed/android/android-sdk-linux/tools:/home/seed/android/android-sdk-linux/platform-tools:/home/seed/android
/android-ndk/android-ndk-r8d:/home/seed/.local/bin
QT_IM_MODULE=ibus
QT_QPA_PLATFORMTHEME=appmenu-qt5
XDG_SESSION_TYPE=x11
PWD=/home/seed
JOB=unity-settings-daemon
XMODIFIERS=@im=ibus
JAVA_HOME=/usr/lib/jvm/java-8-oracle
GNOME_KEYRING_PID=
LANG=en_US.UTF-8
GDM_LANG=en_US
MANDATORY_PATH=/usr/share/gconf/ubuntu.mandatory.path
COMPIZ_CONFIG_PROFILE=ubuntu-lowgfx
IM_CONFIG_PHASE=1
GDMSESSION=ubuntu
SESSIONTYPE=gnome-session
GTK2_MODULES=overlay-scrollbar
SHLVL=1
HOME=/home/seed
XDG_SEAT=seat0
LANGUAGE=en_US
LIBGL_ALWAYS_SOFTWARE=1
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
UPSTART_INSTANCE=
UPSTART_EVENTS=xsession started
XDG_SESSION_DESKTOP=ubuntu
LOGNAME=seed
COMPIZ_BIN_PATH=/usr/bin/
DBUS_SESSION_BUS_ADDRESS=unix:abstract=/tmp/dbus-hcWCOOLoqK
J2SDKDIR=/usr/lib/jvm/java-8-oracle
XDG_DATA_DIRS=/usr/share/ubuntu:/usr/share/gnome:/usr/local/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_PES2UG19CS052_Anurag.R.Simha@Victim:~$ 
```

Fig. 1(a): All the environment variables.

`printenv` can also be used.

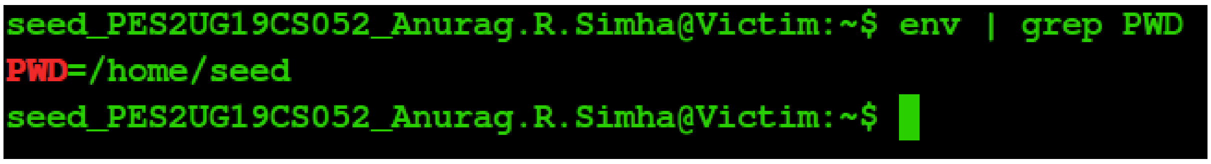


```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~$ printenv
XDG_VTNR=7
XDG_SESSION_ID=c1
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed
CLUTTER_IM_MODULE=xim
SESSION=ubuntu
ANDROID_HOME=/home/seed/android/android-sdk-linux
GPG_AGENT_INFO=/home/seed/.gnupg/S.gpg-agent:0:1
TERM=xterm-256color
VTE_VERSION=4205
XDG_MENU_PREFIX=gnome-
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_options.so.1.64.0:/ho
WINDOWID=65011722
UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session/1000/1605
GNOME_KEYRING_CONTROL=
GTK_MODULES=gail:atk-bridge:unity-gtk-module
USER=seed
```

Fig. 1(b): Using `printenv` to display environment variables.

To fetch the information of a particular command, grep is used.

The command: `env | grep PWD`



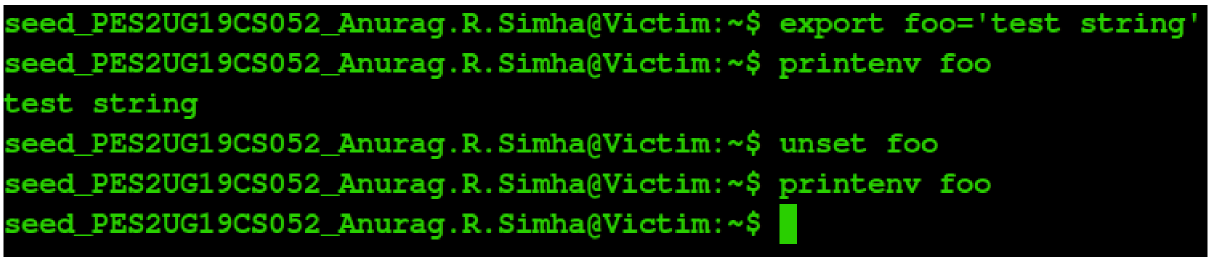
```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~$ env | grep PWD
PWD=/home/seed
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~$ 
```

Fig. 1(c): Details of PWD environment variable.

Here, it’s observed that the program executing this command sits in the location, `/home/seed`.

b) Using export and unset to set environment variables is performed. It should be noted that these two commands are not separate programs; they are two of Bash's internal commands (it’s not discoverable outside of Bash). Unset is used to unset the variable.

The command: `unset foo`



```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~$ export foo='test string'
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~$ printenv foo
test string
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~$ unset foo
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~$ printenv foo
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~$ 
```

Fig. 1(d): Set and unset environment variable.

The shell variable is initially set and stored in the memory. When the unset command is fired, the variable gets erased.

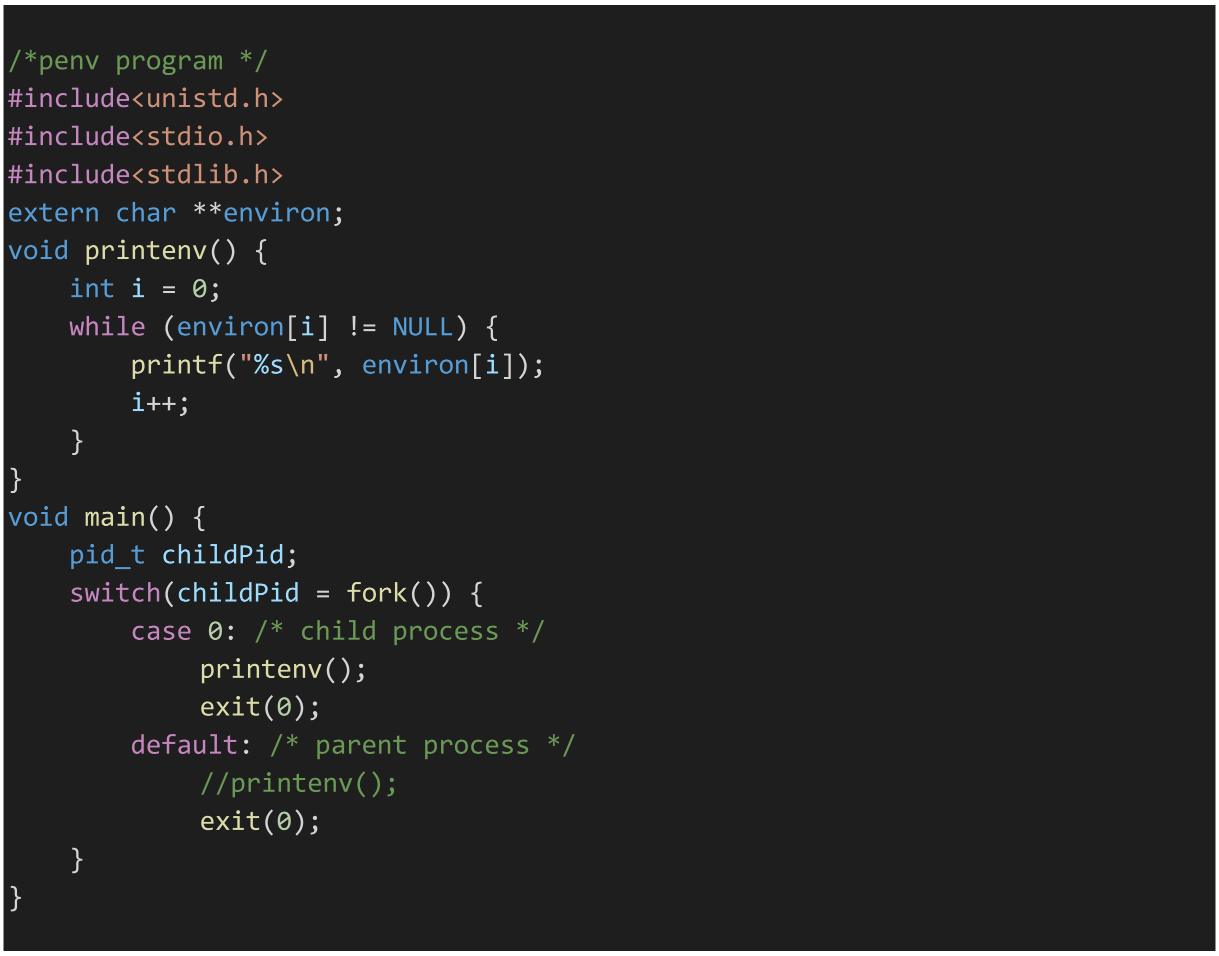
Task 2: Inheriting environment variables from parents

In this task, it’s studied how the environment variables are inherited by child processes from their parents. In Unix, the `fork()` method 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.

Stage 1

The program:

Name: `PENV.c`



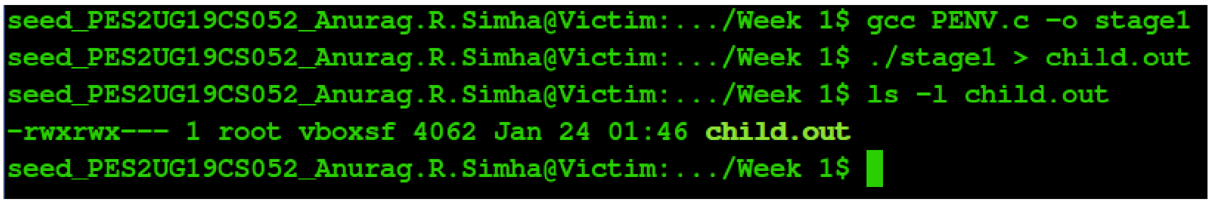
```c
/*penv program */
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>
extern char **environ;
void printenv() {
    int i = 0;
    while (environ[i] != NULL) {
        printf("%s\n", environ[i]);
        i++;
    }
}
void main() {
    pid_t childPid;
    switch(childPid = fork()) {
        case 0: /* child process */
            printenv();
            exit(0);
        default: /* parent process */
            //printenv();
            exit(0);
    }
}
```

In this program, a child process gets created by the `fork()` method. Then, the loop for this child process is called.

Now, this program is compiled and run.

The commands:

```
$ gcc penv.c -o stage1
$ ./stage1 > child.out
$ ls -l child.out
```



```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ gcc PENV.c -o stage1
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ ./stage1 > child.out
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ ls -l child.out
-rwxrwx--- 1 root vboxsf 4062 Jan 24 01:46 child.out
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ 
```

Fig. 2(a): Creating the child process.

Here the environment variables for the child process get displayed.

Stage 2

Now, the `printenv()` method under the zeroth case [line 17] is dwindled into a comment; and the same method that's a comment under default is made a process [line 20].

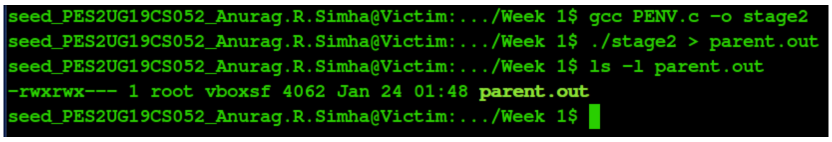
The program:

Name: `PENV.c`

```c
/*penv program */
#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);
    }
}
```

The commands:

```
$ gcc penv.c -o stage2

$ ./stage2 > parent.out

$ ls -l parent.out
```



```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ gcc PENV.c -o stage2
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ ./stage2 > parent.out
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ ls -l parent.out
-rwxrwx--- 1 root vboxsf 4062 Jan 24 01:48 parent.out
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$
```

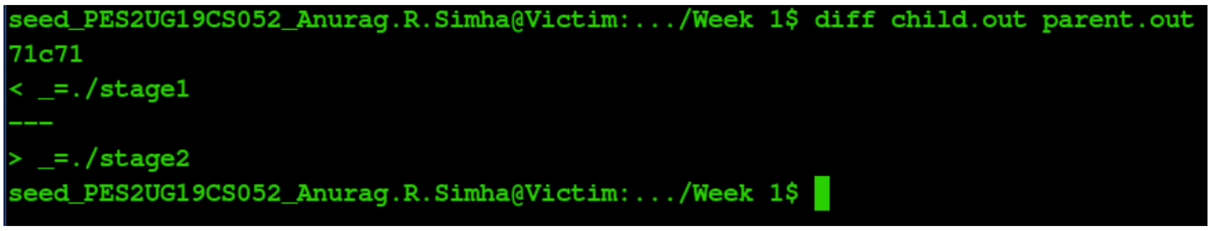
Fig. 2(b): Creating the parent process.

The second stage is involved with displaying the environment variables for the parent process.

Stage 3

The difference(s) between the two output files is/are now inspected.

The command: `diff child.out parent.out`



```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ diff child.out parent.out
71c71
< _=./stage1
---
> _=./stage2
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ 
```

Fig. 2(c): Inspecting the difference between the two files.

It's noticed that there are special symbols between the two files.

Task 3: Environment variables and execve()

In this task, it's studied how the 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 the spotlight is upon what happens to the environment variables? Are they automatically inherited by the new program?

Stage 1

The following program is executed. This program simply executes a program called `/usr/bin/env`, which prints out the environment variables of the current process.

The program:

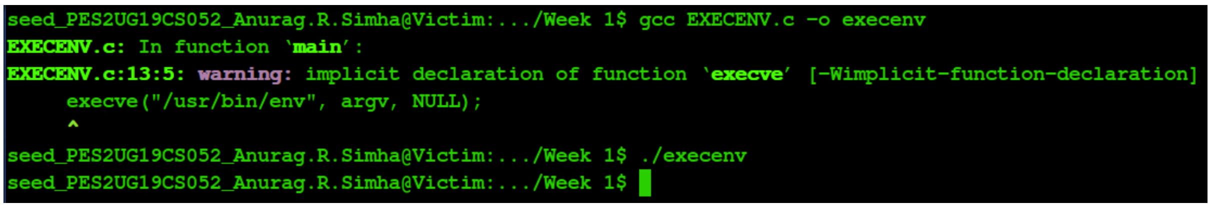
Name: `EXECENV.c`

```
#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;
}
```

In the program, the method `execve()` displays all the environment variables. But, the third parameter is set to NULL.

The commands:

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



```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ gcc EXECENV.c -o execenv
EXECENV.c: In function 'main':
EXECENV.c:13:5: warning: implicit declaration of function 'execve' [-Wimplicit-function-declaration]
     execve("/usr/bin/env", argv, NULL);
     ^
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ ./execenv
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ 
```

Fig. 3(a): Compiling and running the program.

It's noticed that there's no output to the program that got compiled. For, the third parameter is NULL.

Stage 2

The third parameter of `execve()` method is altered to `environ`.

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

Below is the program.

Name: `EXECENV.c`

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

extern char **environ;
```

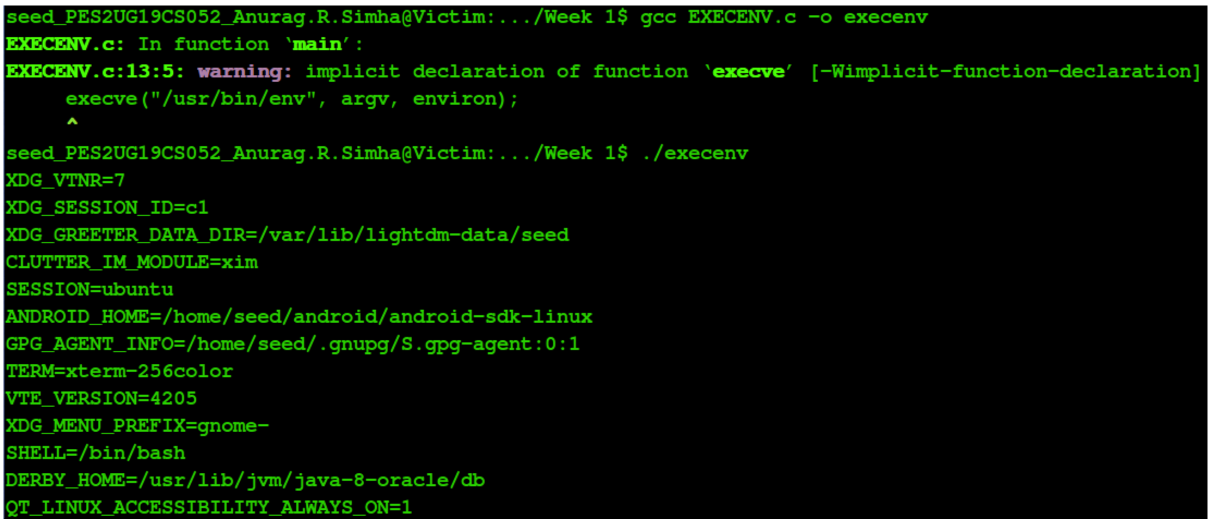
```c
int main()
{
    char *argv[2];

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

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

    return 0;
}
```

Now, in the program, the method `execve()` displays all the environment variables. But, setting the third parameter to environ is bound to display the environment variables.

Now, the program is compiled and run.

`$ gcc EXECENV.c -o execenv`

`$./execenv`



```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ gcc EXECENV.c -o execenv
EXECENV.c: In function 'main':
EXECENV.c:13:5: warning: implicit declaration of function 'execve' [-Wimplicit-function-declaration]
     execve("/usr/bin/env", argv, environ);
     ^
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ ./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
XDG_MENU_PREFIX=gnome-
SHELL=/bin/bash
DERBY_HOME=/usr/lib/jvm/java-8-oracle/db
QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
```

Fig. 3(b): Compiling and running the altered program.

Finally, after the alteration, it's lucid that the environment variables are displayed by the program.

Q. What happens to the environment variables? Are they automatically inherited by the new program?

The environment variables are successfully displayed only in the second stage. No, they are not automatically inherited by the new program until calling environ. After reading man execve, there are 3 parameters in execve, namely:

```
int execve(const char *filename, char *const argv[],
char *const envp[]);
```

when `envp[]` is null, nothing is displayed. but when `envp[]` is filled with environ, it shows all envrionment. `extern char **environ;` means that the code above loads the environment variable and executes it with execve , and the env cannot be called directly without loading the environment variable.

Task 4: Environment variables and system()

In this task, it's studied how the environment variables get 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 the implementation of the `system()` function is noticed, it's seen that `execl()` is used 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.

Below is the program to verify this.

Name: `SYSENV.c`

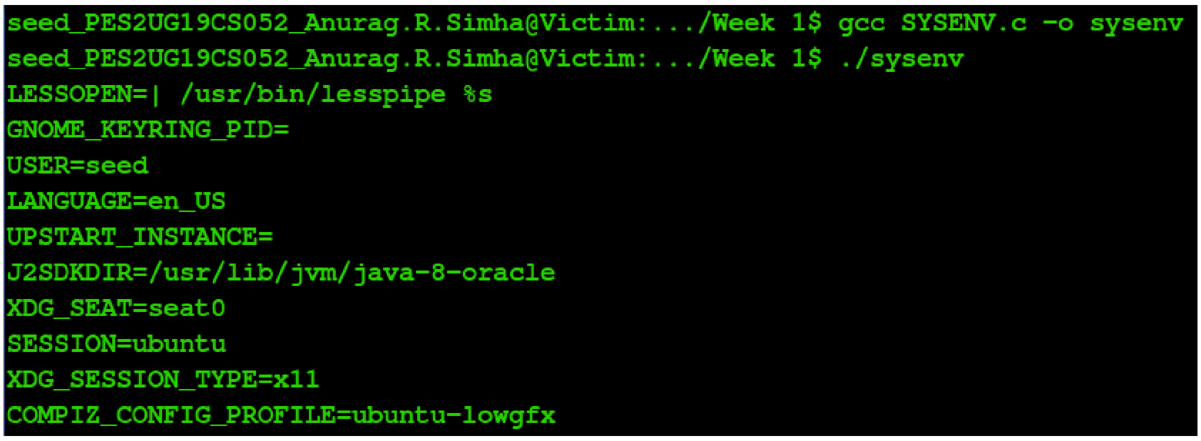
```
#include<stdio.h>
#include<stdlib.h>

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

    return 0;
}
```

In this program, the environment variables are displayed by the `system()` function.

The commands:

```
$ gcc SYSENV.c -o sysenv
```

```
$ ./sysenv
```



```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ gcc SYSENV.c -o sysenv
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ ./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
```

Fig. 4(a): Executing the sysenv program.

It's noticed that the environment variables are triumphantly displayed by the function that executed the program.

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 many interesting things, but it escalates the user's privilege when executed, making it quite risky. Although the behaviours of Set-UID programs are decided by their program logic, and not by the users, they can indeed affect the behaviours via environment variables. To understand how Set-UID programs are affected, it's first examined whether environment variables are inherited by the Set-UID program's process from the user's process.

Stage 1

The program:

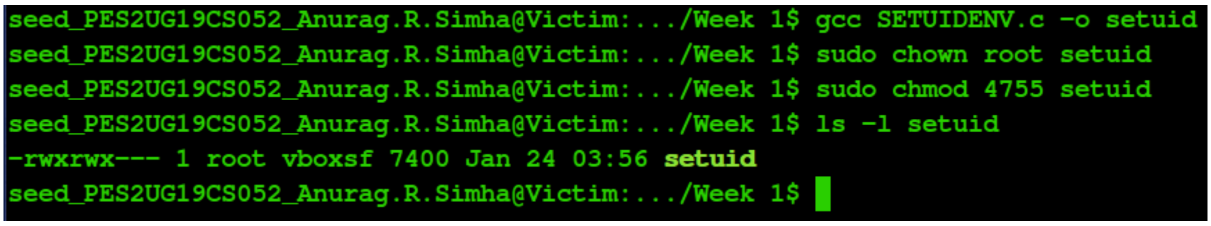
Name: `SETUIDENV.c`

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

extern char **environ;

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

Next, the program is compiled and run.

The commands:

```
$ gcc SETUIDENV.c -o setuid

$ sudo chown root setuid

$ sudo chmod 4755 setuid
```



```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ gcc SETUIDENV.c -o setuid
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ sudo chown root setuid
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ sudo chmod 4755 setuid
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ ls -l setuid
-rwxrwx--- 1 root vboxsf 7400 Jan 24 03:56 setuid
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ 
```

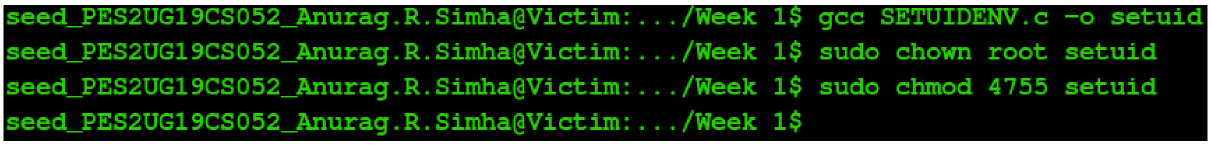
Fig. 5(a): Running the program.

Stage 2

The above program is compiled, with the ownership changed to root, and making it a Set-UID program.

The commands:

```
$ gcc SETUIDENV.c -o setuid

$ sudo chown root setuid

$ sudo chmod 4755 setuid
```



```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ gcc SETUIDENV.c -o setuid
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ sudo chown root setuid
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ sudo chmod 4755 setuid
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$
```

Fig. 5(b): Changing the ownership.

Stage 3

In the Bash shell (Normal user account must be used, not the root account), the export command is used to set the following environment variables (they may exist already):

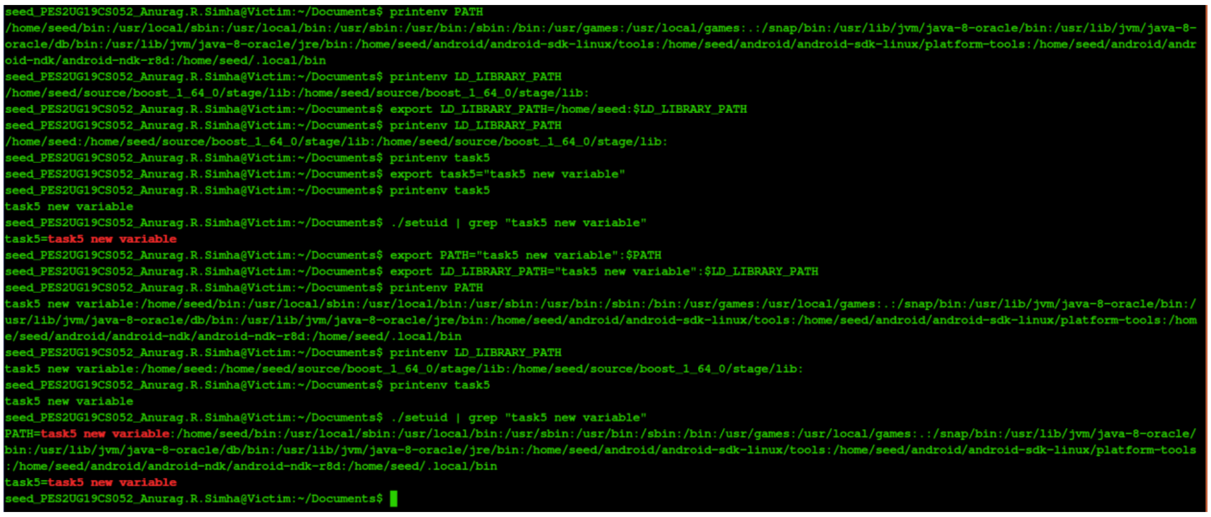
`PATH`

`LD LIBRARY PATH`

`ANY NAME` (this is an environment variable defined by the user).

The 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

$ ./setuid | grep "task5 new variable"

$ export PATH="task5 new variable":$PATH

$ export LD_LIBRARY_PATH="task5 new
variable":$LD_LIBRARY_PATH

$ printenv PATH

$ printenv LD_LIBRARY_PATH

$ printenv task5

$ ./setuid | grep "task5 new variable"
```



```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ printenv PATH
/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/android/android-sdk-linux/platform-tools:/home/seed/android/andr
oid-ndk/android-ndk-r8d:/home/seed/.local/bin
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ printenv LD_LIBRARY_PATH
/home/seed/source/boost_1_64_0/stage/lib:/home/seed/source/boost_1_64_0/stage/lib:
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ export LD_LIBRARY_PATH=/home/seed:$LD_LIBRARY_PATH
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ printenv LD_LIBRARY_PATH
/home/seed:/home/seed/source/boost_1_64_0/stage/lib:/home/seed/source/boost_1_64_0/stage/lib:
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ printenv task5
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ export task5="task5 new variable"
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ printenv task5
task5 new variable
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ ./setuid | grep "task5 new variable"
task5=task5 new variable
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ export PATH="task5 new variable":$PATH
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ export LD_LIBRARY_PATH="task5 new variable":$LD_LIBRARY_PATH
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ printenv 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-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/android/android-sdk-linux/platform-tools:/hom
e/seed/android/android-ndk/android-ndk-r8d:/home/seed/.local/bin
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ printenv LD_LIBRARY_PATH
task5 new variable:/home/seed:/home/seed/source/boost_1_64_0/stage/lib:/home/seed/source/boost_1_64_0/stage/lib:
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ printenv task5
task5 new variable
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ ./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-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/android/android-sdk-linux/platform-tools
:/home/seed/android/android-ndk/android-ndk-r8d:/home/seed/.local/bin
task5=task5 new variable
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ 
```

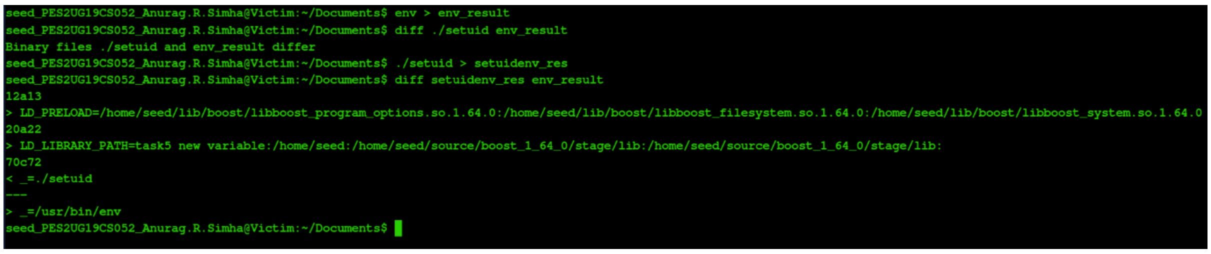
Fig. 5(c): Output of the commands.

These environment variables are set in the user’s shell process.

Now, the difference is examined.

The commands:

```
$ env > env_result

$ diff ./setuid env_result

$ ./setuid > setuidenv_res

$ diff setuidenv_res env_result
```

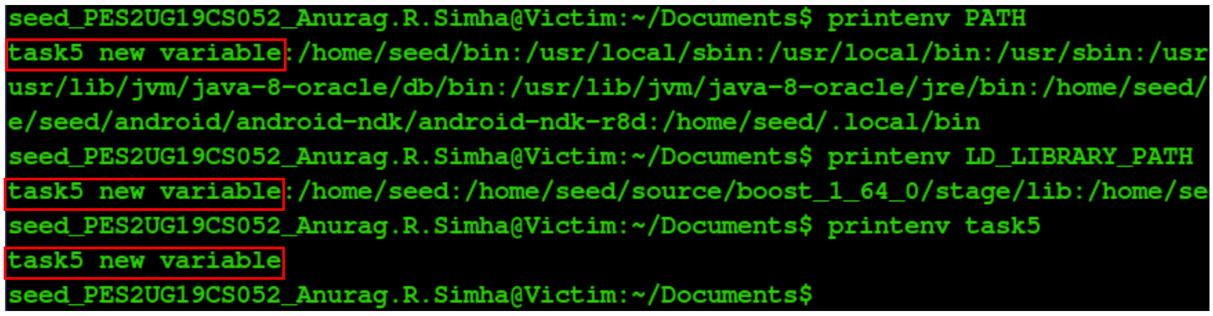


```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ env > env_result
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ diff ./setuid env_result
Binary files ./setuid and env_result differ
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ ./setuid > setuidenv_res
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ diff setuidenv_res env_result
12a13
> LD_PRELOAD=/home/seed/lib/boost/libboost_program_options.so.1.64.0:/home/seed/lib/boost/libboost_filesystem.so.1.64.0:/home/seed/lib/boost/libboost_system.so.1.64.0
20a22
> LD_LIBRARY_PATH=task5 new variable:/home/seed:/home/seed/source/boost_1_64_0/stage/lib:/home/seed/source/boost_1_64_0/stage/lib:
70c72
< _=./setuid
---
> _=/usr/bin/env
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$
```

Fig. 5(d): Examining the environment variables.

It's noticed that there is a difference between the output stored in both the files. This is due to the alterations performed on the environment variables using the Set-UID feature in UNIX.

Q. Check whether all the environment variables set in the shell process (parent) get into the Set-UID child process.



```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ printenv PATH
task5 new variable:/home/seed/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr
usr/lib/jvm/java-8-oracle/db/bin:/usr/lib/jvm/java-8-oracle/jre/bin:/home/seed/
e/seed/android/android-ndk/android-ndk-r8d:/home/seed/.local/bin
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ printenv LD_LIBRARY_PATH
task5 new variable:/home/seed:/home/seed/source/boost_1_64_0/stage/lib:/home/se
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ printenv task5
task5 new variable
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$
```

Fig. 5(e): Manipulating the environment variables.

Yes, the environment variables set in the shell process indeed get into the Set-UID child process. The sole reason for this is that the value, 'task5 new variable' is being injected into the legitimate environment variables. The shell command, 'export' is the reason to triumph this manipulation. The activation of export 'task5 new variable' results in the value getting into the child process.

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 due to the actual behaviour 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 behaviour of the Set-UID program. In Bash, the PATH environment variable can be changed 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.

The program:

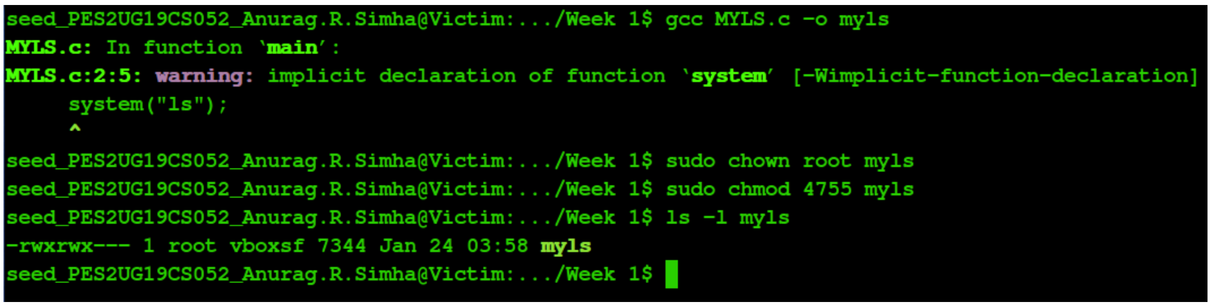
Name: `MYLS.c`

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

In this program, the system function executes the ‘ls’ command.

The commands:

```
$ gcc MYLS.c -o myls

$ sudo chown root myls

$ sudo chmod 4755 myls

$ ls -l myls
```



```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ 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_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ sudo chown root myls
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ sudo chmod 4755 myls
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ ls -l myls
-rwxrwx--- 1 root vboxsf 7344 Jan 24 03:58 myls
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ 
```

Fig. 6(a): Changing the permissions and compiling the program.

Therefore, the file is created.

The next program:

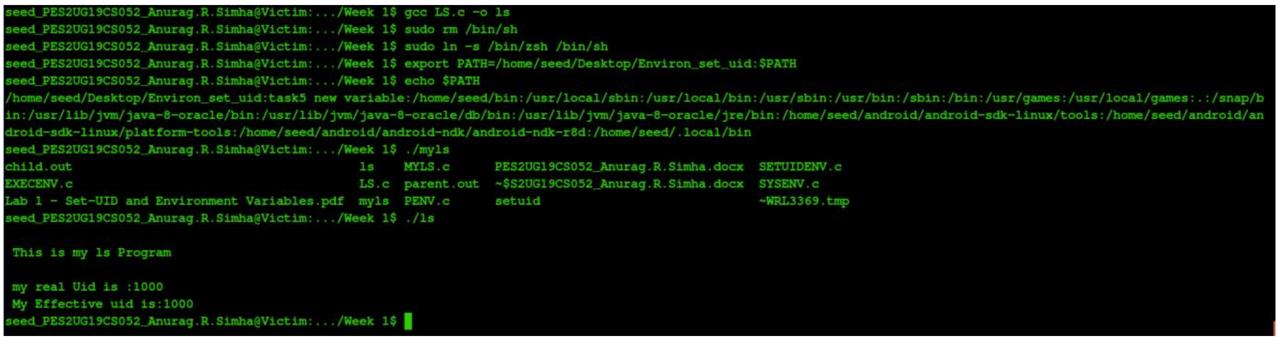
Name: `LS.c`

```c
#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);
}
```

In this program, the UID and EUID is displayed.

The 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_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ gcc LS.c -o ls
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ sudo rm /bin/sh
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ sudo ln -s /bin/zsh /bin/sh
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ export PATH=/home/seed/Desktop/Environ_set_uid:$PATH
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ echo $PATH
/home/seed/Desktop/Environ_set_uid:task5 new variable:/home/seed/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:.:/snap/b
in:/usr/lib/jvm/java-8-oracle/bin:/usr/lib/jvm/java-8-oracle/db/bin:/usr/lib/jvm/java-8-oracle/jre/bin:/home/seed/android/android-sdk-linux/tools:/home/seed/android/an
droid-sdk-linux/platform-tools:/home/seed/android/android-ndk/android-ndk-r8d:/home/seed/.local/bin
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ ./myls
child.out                                 ls    MYLS.c      PES2UG19CS052_Anurag.R.Simha.docx  SETUIDENV.c
EXECENV.c                                 LS.c  parent.out  ~$S2UG19CS052_Anurag.R.Simha.docx  SYSENV.c
Lab 1 - Set-UID and Environment Variables.pdf  myls  PENV.c      setuid                             ~WRL3369.tmp
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ ./ls

 This is my ls Program

 my real Uid is :1000
 My Effective uid is:1000
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ 
```

Fig. 6(b): Running the programs.

With zsh shell root shell is achieved. But with native shell of Ubuntu 16.04 root shell can’t be achieved. Also, the command is manipulated.

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

In this task, it’s studied 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 behaviour 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 behaviours, `LD LIBRARY PATH` and `LD PRELOAD` are the two concerned targets 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.

Stage 1

First, it's observed how these environment variables influence the behaviour of dynamic loader/linker when running a normal program.

The program:

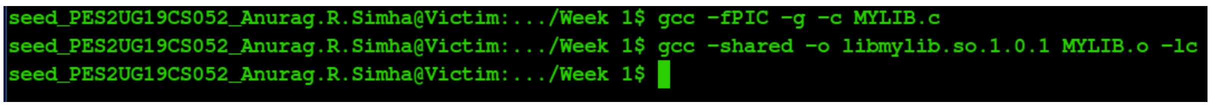
Name: `MYLIB.c`

```
#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");
}
```

Now, the program is compiled.

The commands:

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



```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ gcc -fPIC -g -c MYLIB.c
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ gcc -shared -o libmylib.so.1.0.1 MYLIB.o -lc
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$
```

Fig. 7.1(a): Compiling the program.

Next, the LD_PRELOAD environment variable is set.

The command:

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

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ export LD_PRELOAD=./libmylib.so.1.0.1
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$
```

Fig. 7.1(b): Setting the LD_PRELOAD environment variable.

Finally, the following program MYPROG.c is compiled, and is placed in the same directory as the above dynamic link library libmylib.so.1.0.1

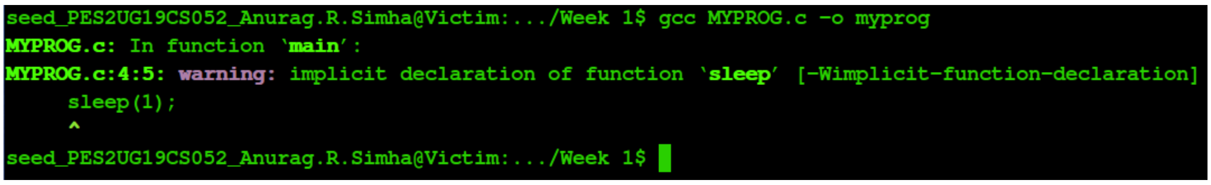
The program:

Name: `MYPROG.c`

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

The program is compiled.

The command: `gcc MYPROG.c -o myprog`



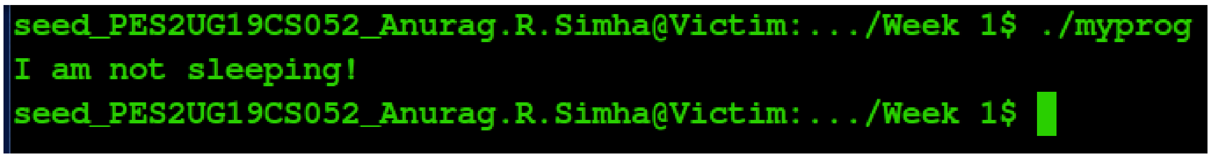
```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ gcc MYPROG.c -o myprog
MYPROG.c: In function ‘main’:
MYPROG.c:4:5: warning: implicit declaration of function ‘sleep’ [-Wimplicit-function-declaration]
     sleep(1);
     ^
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$
```

Fig. 7.1(c): Compiling the program.

Stage 2

Now, the program, `MYPROG.c` is executed under the following conditions.

- myprog is made a regular program, and ran as a normal user.



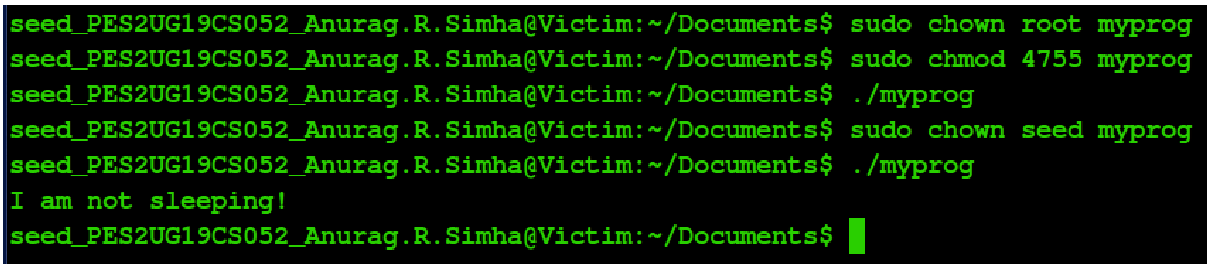
```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$ ./myprog
I am not sleeping!
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../Week 1$
```

Fig. 7.2(a): Running myprog normally

- myprog is made a Set-UID root program, and ran as a normal user.

The commands:

```
$ sudo chown root myprog
$ sudo chmod 4755 myprog
$ ./myprog
$ sudo chown seed myprog
$ ./myprog
```



```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ sudo chown root myprog
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ sudo chmod 4755 myprog
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ ./myprog
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ sudo chown seed myprog
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ ./myprog
I am not sleeping!
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$
```

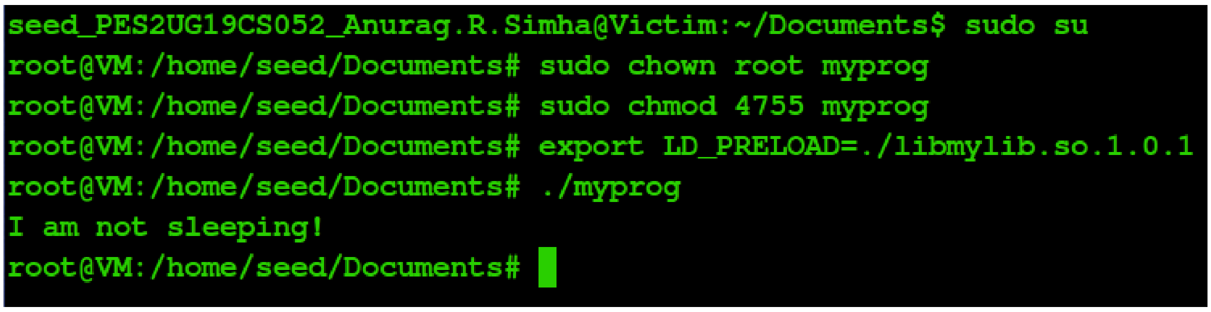
Fig. 7.2(b): Running myprog after making it a Set-UID root program.

If the program owner is changed to root, sleep works. But, if the owner is changed to seed, the sleep injection is successful, indicating a different env. The presence of the countermeasure foils the invoking of the shared program.

- myprog is made a Set-UID root program, the LD_PRELOAD environment variable is exported again in the root account and ran.

The commands:

```
$ sudo su
$ sudo chown root myprog
$ sudo chmod 4755 myprog
$ export LD_PRELOAD=./libmylib.so.1.0.1
$ ./myprog
```



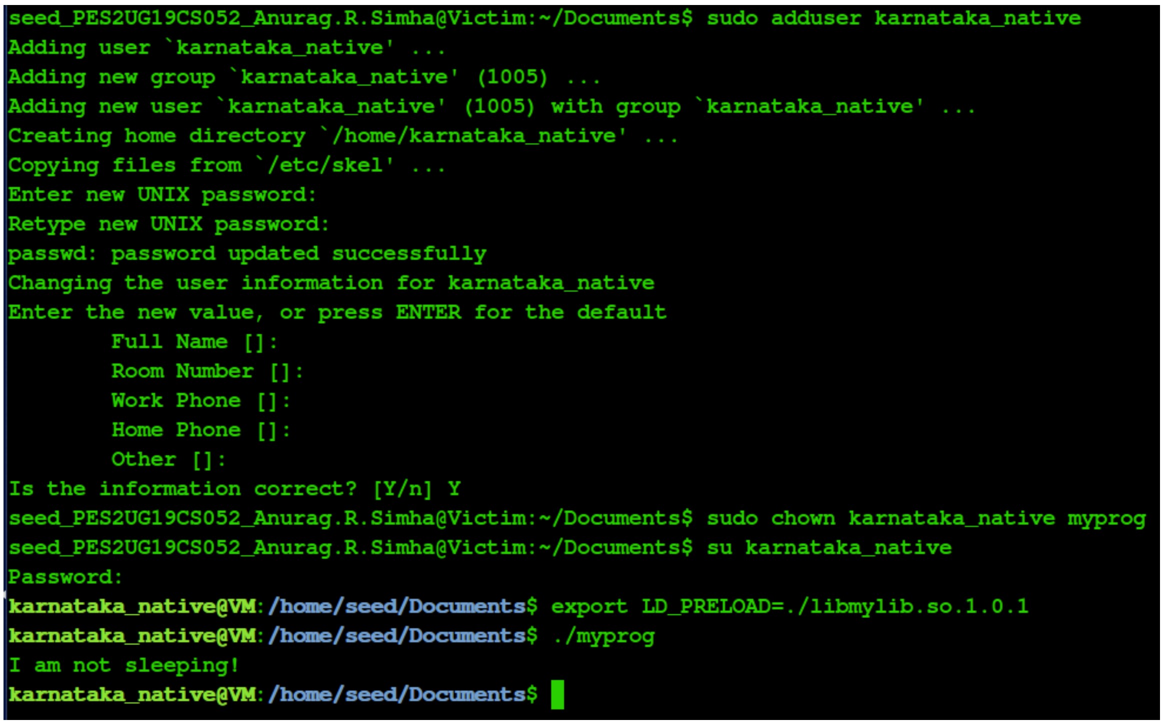
```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ sudo su
root@VM:/home/seed/Documents# sudo chown root myprog
root@VM:/home/seed/Documents# sudo chmod 4755 myprog
root@VM:/home/seed/Documents# export LD_PRELOAD=./libmylib.so.1.0.1
root@VM:/home/seed/Documents# ./myprog
I am not sleeping!
root@VM:/home/seed/Documents#
```

Fig. 7.2(c): Performing root operations on myprog.

- myprog is made a Set-UID user1 program (i.e., the owner is user1, which is another user account), the LD_PRELOAD environment variable is exported again in a different user’s account (not-root user) and ran.

The commands:

```
$ sudo adduser karnataka_native
$ sudo chown karnataka_native myprog
$ su karnataka_native
$ export LD_PRELOAD=./libmylib.so.1.0.1
$ ./myprog
```



```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ sudo adduser karnataka_native
Adding user `karnataka_native' ...
Adding new group `karnataka_native' (1005) ...
Adding new user `karnataka_native' (1005) with group `karnataka_native' ...
Creating home directory `/home/karnataka_native' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for karnataka_native
Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] Y
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ sudo chown karnataka_native myprog
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ su karnataka_native
Password:
karnataka_native@VM:/home/seed/Documents$ export LD_PRELOAD=./libmylib.so.1.0.1
karnataka_native@VM:/home/seed/Documents$ ./myprog
I am not sleeping!
karnataka_native@VM:/home/seed/Documents$
```

Fig. 7.2(d): Another user performing operations on myprog.

Stage 3

Q. What causes the differences?

The differences caused here are due to the access control manipulation performed and the behaviour of environment variables. Till fig. 7.2(c) root had at least some access to the file. When seed also got the ownership, the injection triumphed. Here, the shell variable `LD_PRELOAD` is used effectively after manipulating the access privileges. There is a shared library between the two programs. This is done with the aid of that environment variable (`LD_PRELOAD`). In the end, after karnataka_native gains access to the file, and seed already had access to the file, myprog slept for a second and then ran mylib. Here, the play is with access control mechanism and `LD_PRELOAD` environment variable.

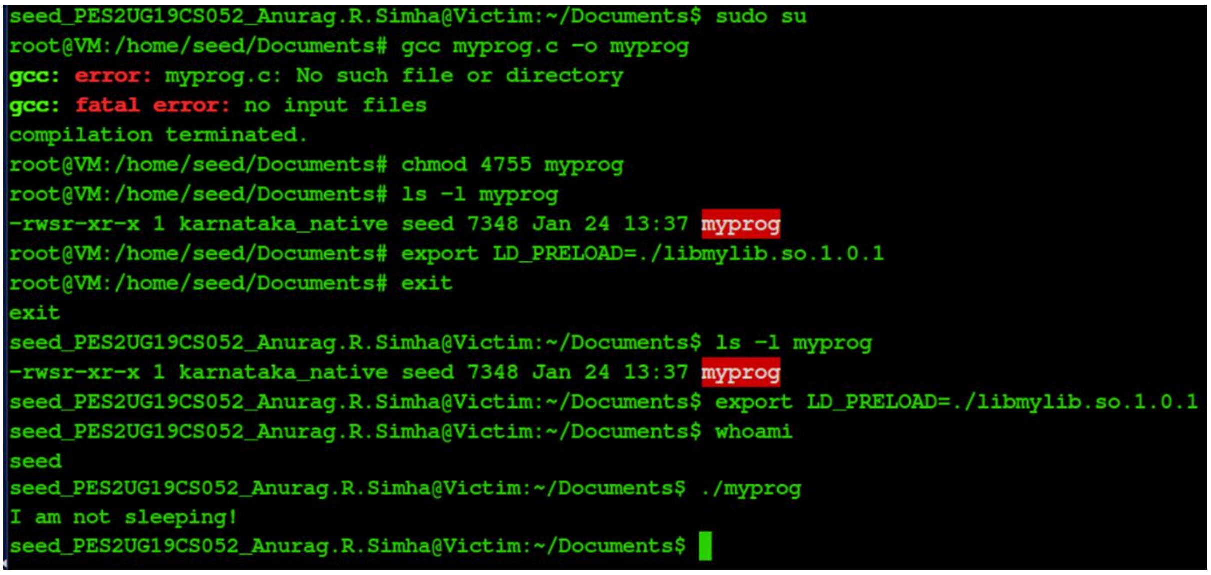
An experimental setup was made for this. The commands below were executed.

In the root environment:

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

Out of the root environment:

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



```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ sudo su
root@VM:/home/seed/Documents# gcc myprog.c -o myprog
gcc: error: myprog.c: No such file or directory
gcc: fatal error: no input files
compilation terminated.
root@VM:/home/seed/Documents# chmod 4755 myprog
root@VM:/home/seed/Documents# ls -l myprog
-rwsr-xr-x 1 karnataka_native seed 7348 Jan 24 13:37 myprog
root@VM:/home/seed/Documents# export LD_PRELOAD=./libmylib.so.1.0.1
root@VM:/home/seed/Documents# exit
exit
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ ls -l myprog
-rwsr-xr-x 1 karnataka_native seed 7348 Jan 24 13:37 myprog
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ export LD_PRELOAD=./libmylib.so.1.0.1
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ whoami
seed
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ ./myprog
I am not sleeping!
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$
```

Fig. 7.3(a): Noting the behaviour.

When root attempts to compile the file, it deliberately fails. On exiting the root terminal, when the program is run after exporting `LD_PRELOAD`, there is a triumphant injection observed.

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. It's observed how the PATH environment variable affects the behaviour of `system()`, since the variable affects how the shell works. `execve()` does not have the problem, for it does not invoke shell. Invoking a shell has another dangerous consequence, and this time, it has nothing to do with environment variables.

A bijou 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, 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.

The program:

Name: `SYSEXECENV.c`

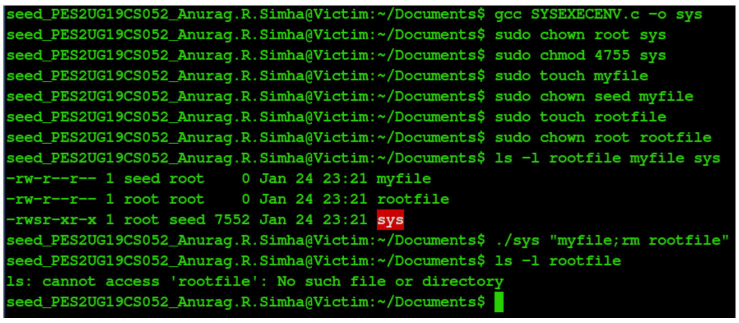
```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

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;
}
```

In this program, to invoke the command, system() is used.

Stage 1

The program is compiled and run. Root is made as the owner. Then, the program is changed to a Set-UID program. Two files named `myfile` and `rootfile` are created with seed and root taking the ownership for the respective files.

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



```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ gcc SYSEXECENV.c -o sys
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ sudo chown root sys
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ sudo chmod 4755 sys
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ sudo touch myfile
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ sudo chown seed myfile
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ sudo touch rootfile
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ sudo chown root rootfile
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ ls -l rootfile myfile sys
-rw-r--r-- 1 seed root    0 Jan 24 23:21 myfile
-rw-r--r-- 1 root root    0 Jan 24 23:21 rootfile
-rwsr-xr-x 1 root seed 7552 Jan 24 23:21 sys
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ ./sys "myfile;rm rootfile"
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ ls -l rootfile
ls: cannot access 'rootfile': No such file or directory
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$
```

Fig. 8.1(a): Compiling the file and executing the manipulation commands.

Q. 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?

Yes, the operation is possible. On closely observing the output (as displayed in fig. 8.1(a)), although seed owns no ownership to the `rootfile`, yet it's removed by the sys program through a mere single-line command.

Stage 2

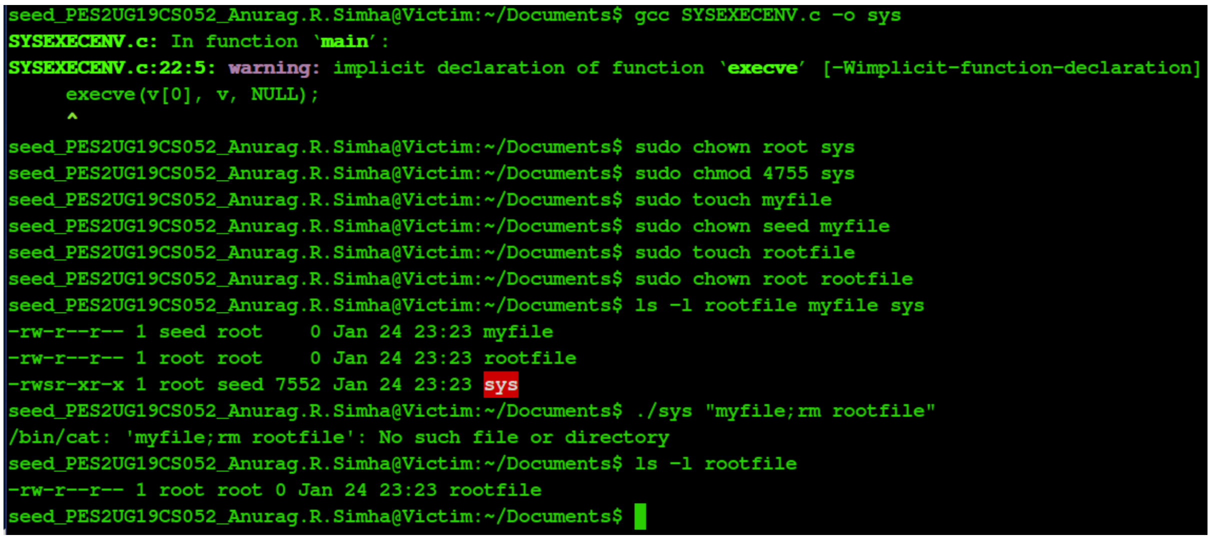
Next, line number 21 where the `system()` function gets called is made a comment. Then, the `execve()` function is released from being a comment. This persuades a different observation leading the program to use `execve()` for invoking the command.

The program:

Name: `SYSEXECENV.c`

```c
#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;
}
```

Once again, the command set in stage 1 is fired.



```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ gcc SYSEXECENV.c -o sys
SYSEXECENV.c: In function `main':
SYSEXECENV.c:22:5: warning: implicit declaration of function `execve' [-Wimplicit-function-declaration]
     execve(v[0], v, NULL);
     ^
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ sudo chown root sys
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ sudo chmod 4755 sys
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ sudo touch myfile
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ sudo chown seed myfile
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ sudo touch rootfile
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ sudo chown root rootfile
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ ls -l rootfile myfile sys
-rw-r--r-- 1 seed root    0 Jan 24 23:23 myfile
-rw-r--r-- 1 root root    0 Jan 24 23:23 rootfile
-rwsr-xr-x 1 root seed 7552 Jan 24 23:23 sys
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ ./sys "myfile;rm rootfile"
/bin/cat: 'myfile;rm rootfile': No such file or directory
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ ls -l rootfile
-rw-r--r-- 1 root root 0 Jan 24 23:23 rootfile
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ 
```

Fig. 8.2(a): Attempting the manipulation.

Q. Do your attacks in Step 1 still work? Please describe and explain your observations.

Henceforth, the manipulation is no more possible. For, the manipulation command is not executed. The sys program can't create the command to perform the desired operation. This is due to the procedure call to `execve()` function.

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 since it possesses privileged capabilities.

The program below is compiled and run.

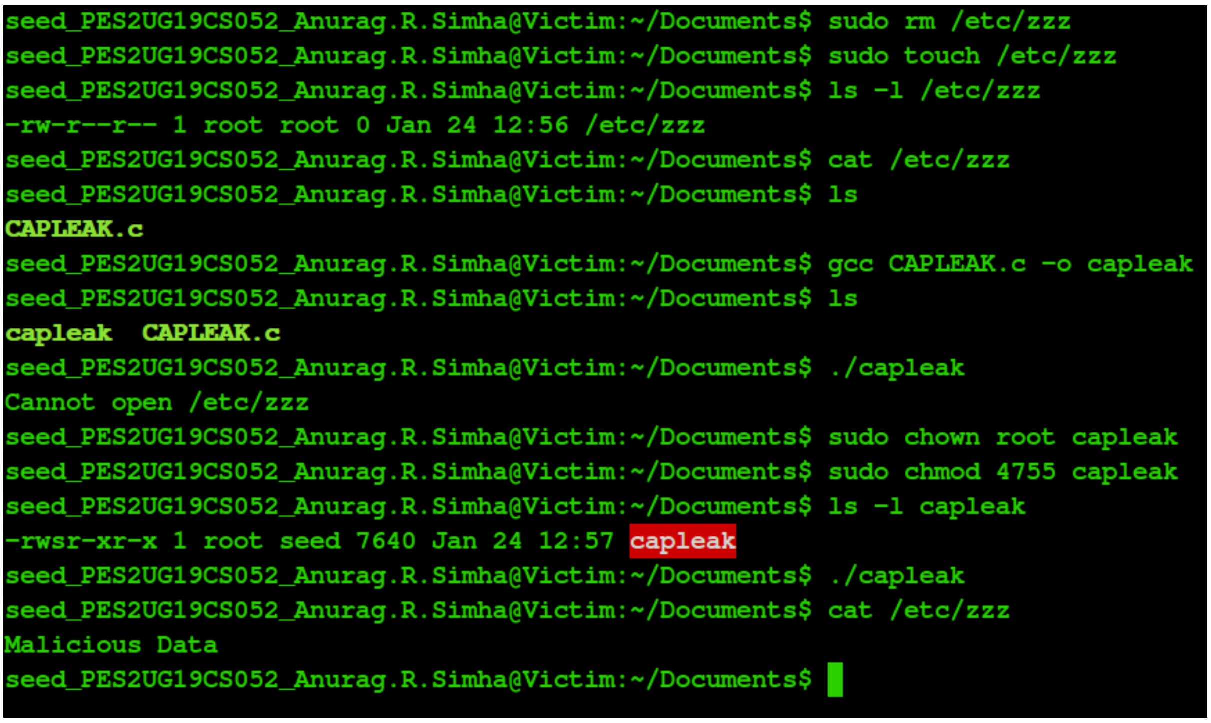
Name: `CAPLEAK.c`

```
#include<stdio.h>
#include<stdlib.h>
#include<fcntl.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
        // have injected the following statements into this process

        write(fd, "Malicious Data\n", 15);
        close(fd);
    }
}
```

In this program, it’s assumed that an important file named zzz is present under the etc folder. To this file, certain data get injected with a child process getting compromised.

The commands:

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



```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ sudo rm /etc/zzz
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ sudo touch /etc/zzz
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ ls -l /etc/zzz
-rw-r--r-- 1 root root 0 Jan 24 12:56 /etc/zzz
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ cat /etc/zzz
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ ls
CAPLEAK.c
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ gcc CAPLEAK.c -o capleak
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ ls
capleak  CAPLEAK.c
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ ./capleak
Cannot open /etc/zzz
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ sudo chown root capleak
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ sudo chmod 4755 capleak
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ ls -l capleak
-rwsr-xr-x 1 root seed 7640 Jan 24 12:57 capleak
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ ./capleak
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ cat /etc/zzz
Malicious Data
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~/Documents$ 
```

Fig. 9(a): Injecting data into the restricted file.

Q. Run the program as a normal user, and describe what you have observed. Will the file `/etc/zzz` be modified? Please explain your observation.

As seen in fig. 9(a), after running the program as a normal user, modification to the file sadly fails. But, on illegally gaining the permissions to that file, it's made possible to alter the file with the desired injectable data. Initially, the access to zzz is restricted. With the aid of `chown` and `chmod` commands the access is granted and manipulation of the file becomes a triumphant operation. This is the manner capability leaking functions.

****