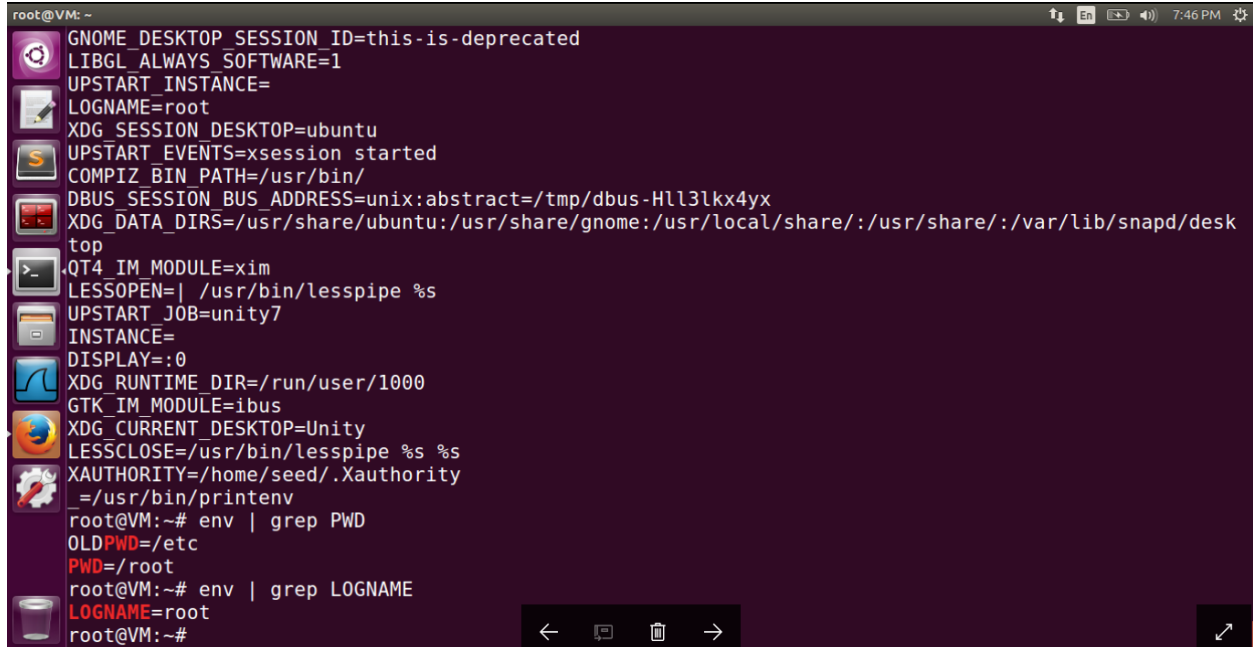


Computer Security Lab 1 Task

Aastha Yadav (ayadav02@syr.edu)

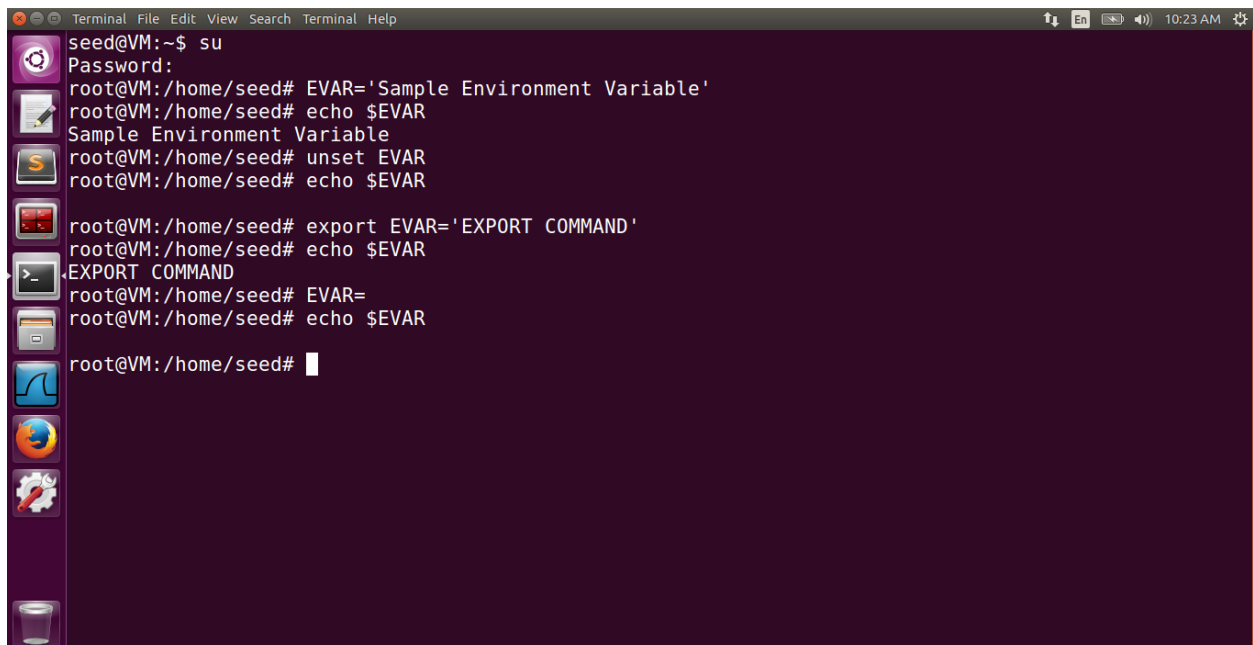
2.1 Task 1: Manipulating Environment Variables



```
root@VM: ~  
GNOME_DESKTOP_SESSION_ID=this-is-deprecated  
LIBGL_ALWAYS_SOFTWARE=1  
UPSTART_INSTANCE=  
LOGNAME=root  
XDG_SESSION_DESKTOP=ubuntu  
UPSTART_EVENTS=xsession started  
COMPIZ_BIN_PATH=/usr/bin/  
DBUS_SESSION_BUS_ADDRESS=unix:abstract=/tmp/dbus-H1l3lkx4yx  
XDG_DATA_DIRS=/usr/share/ubuntu:/usr/share/gnome:/usr/local/share/:/usr/share/:/var/lib/snapd/desk  
top  
QT4_IM_MODULE=xim  
LESSOPEN=| /usr/bin/lesspipe %s  
UPSTART_JOB=unity7  
INSTANCE=  
DISPLAY=:0  
XDG_RUNTIME_DIR=/run/user/1000  
GTK_IM_MODULE=ibus  
XDG_CURRENT_DESKTOP=Unity  
LESSCLOSE=/usr/bin/lesspipe %s %s  
XAUTHORITY=/home/seed/.Xauthority  
_=usr/bin/printenv  
root@VM:~# env | grep PWD  
OLDPWD=/etc  
PWD=/root  
root@VM:~# env | grep LOGNAME  
LOGNAME=root  
root@VM:~#
```

Figure 1

Observations: printenv or env command is used to print all environment variables. Grep command can be used to print particular environment variables.



```
Terminal File Edit View Search Terminal Help  
seed@VM:~$ su  
Password:  
root@VM:/home/seed# EVAR='Sample Environment Variable'  
root@VM:/home/seed# echo $EVAR  
Sample Environment Variable  
root@VM:/home/seed# unset EVAR  
root@VM:/home/seed# echo $EVAR  
  
root@VM:/home/seed# export EVAR='EXPORT COMMAND'  
root@VM:/home/seed# echo $EVAR  
EXPORT COMMAND  
root@VM:/home/seed# EVAR=  
root@VM:/home/seed# echo $EVAR  
  
root@VM:/home/seed#
```

Figure 2

Observations: We can see from the screenshot that export command and Variable='' can be used to set values to environment variables. Unset command can be used to unset environment variables.

Additional Findings:

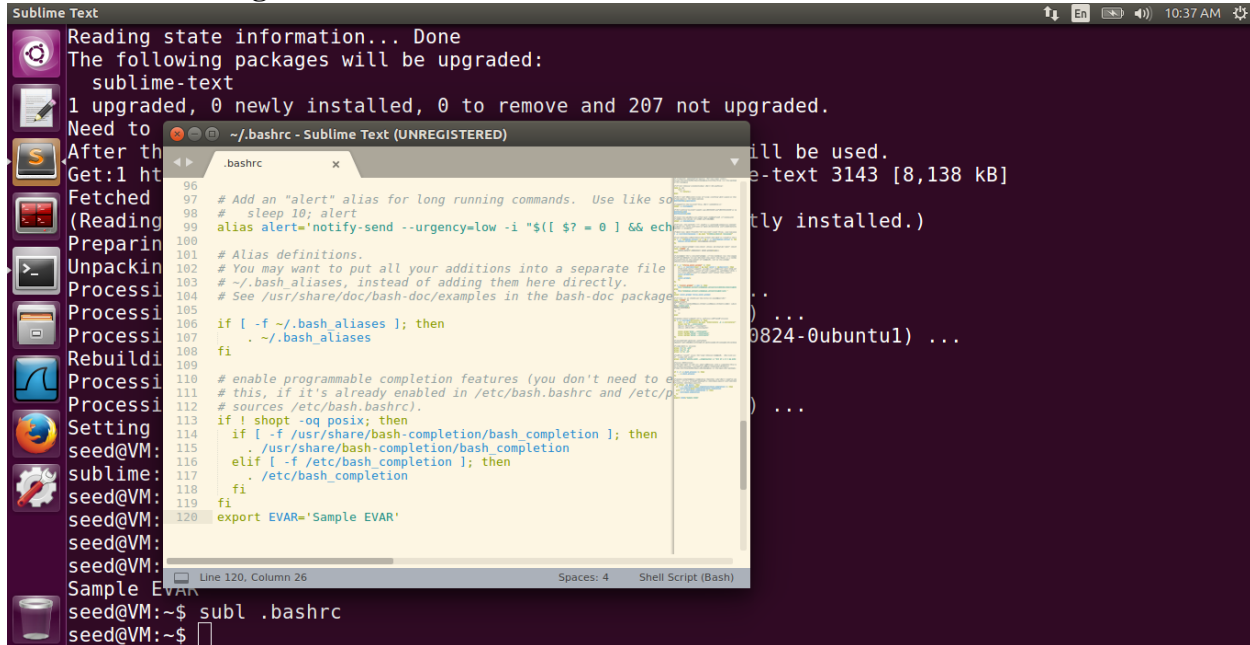


Figure 3

Observations: .bashrc file could also be modified to set or unset environment variables in the shell as we can see from the screenshot.

2.2 Task 2: Inheriting environment variables from parents

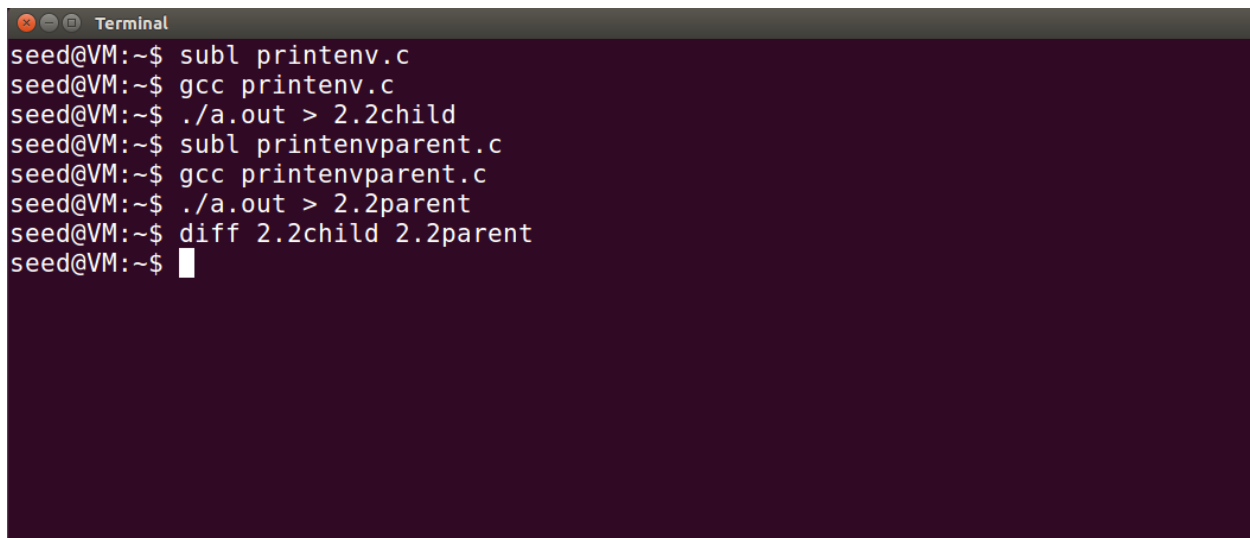


Figure 4

Observations: We compile and run the program and this prints out the environment variables of child process first and then prints out environment variables of parent process.

Findings: Child processes inherit a copy of the parent's environment. Hence there is no difference when we run the diff command on parent's environment variables and child's environment variables. They are the same.

Additional Comments: Child process has a new PID when it forks but the same environment in the inheritance cycle.

2.3 Task 3: Environment variables and execve()

```
seed@VM:~$ gedit execve.c
seed@VM:~$ gcc -o a execve.c
seed@VM:~$ ./a
seed@VM:~$ gedit execve.c
seed@VM:~$ gedit execve.c
seed@VM:~$ gcc -o a execve.c
seed@VM:~$ ./a
XDG_VTNR=7
XDG_SESSION_ID=c1
CLUTTER_IM_MODULE=xim
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed
SESSION=ubuntu
GPG_AGENT_INFO=/home/seed/.gnupg/S.gpg-agent:0:1
TERM=xterm-256color
SHELL=/bin/bash
VTE_VERSION=4205
QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
WINDOWID=62914570
EVAR=Sample EVAR
UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session/1000/1238
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:cs=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*
```

Figure 5

We create a program execve.c where the 3rd argument of the execve command i.e., environment variables are set to NULL. The output of this program is nothing since only the shell is returned. We then modify the 3rd argument of the execve command and set the environment variables. The output of this program is that the environment variables are printed.

Observations: When the environment variables argument of the execve command is set to NULL, it isn't stored in the environment and argument memory and so the calling process doesn't inherit the environment variables. But when the argument takes the environment variables, they are stored in memory, and then the calling process inherits the environment variables.

2.4 Task 4: Environment variables and system()

A terminal window with a dark purple background and a sidebar of application icons on the left. The terminal text shows the user 'seed' at a VM, setting environment variables like XAUTHORITY, LESSOPEN, GNOME_KEYRING_PID, EVAR, USER, LANGUAGE, UPSTART_INSTANCE, XDG_SEAT, SESSION, XDG_SESSION_TYPE, COMPIZ_CONFIG_PROFILE, LD_LIBRARY_PATH, SHLVL, LIBGL_ALWAYS_SOFTWARE, HOME, QT4_IM_MODULE, DESKTOP_SESSION, QT_LINUX_ACCESSIBILITY_ALWAYS_ON, GTK_MODULES, XDG_SEAT_PATH, and INSTANCE, and then running 'gedit system.c', 'gcc -o a system.c', and './a'.

```
root@VM: /home/seed
XAUTHORITY=/home/seed/.Xauthority
= ./a
seed@VM:~$ gedit system.c
seed@VM:~$ gcc -o a system.c
seed@VM:~$ ./a
LESSOPEN=| /usr/bin/lesspipe %s
GNOME_KEYRING_PID=
EVAR=Sample EVAR
USER=seed
LANGUAGE=en_US
UPSTART_INSTANCE=
XDG_SEAT=seat0
SESSION=ubuntu
XDG_SESSION_TYPE=x11
COMPIZ_CONFIG_PROFILE=ubuntu-lowgfx
LD_LIBRARY_PATH=/home/seed/source/boost_1_64_0/stage/lib:/home/seed/source/boost_1_64_0/stage/lib:
SHLVL=1
LIBGL_ALWAYS_SOFTWARE=1
HOME=/home/seed
QT4_IM_MODULE=xim
DESKTOP_SESSION=ubuntu
QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
GTK_MODULES=gail:atk-bridge:unity-gtk-module
XDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat0
INSTANCE=
DBUS_SESSION_BUS_ADDRESS=unix:abstract=/tmp/dbus-cKtCRa0Icg
```

Figure 6

On observing the processes, we find that the shell calls the executable program, which then calls the `sh -c` command, which then internally calls the `execve` command. We can observe the PIDs and PPIDs of the processes.

Observation: When the system function executes, it doesn't execute the command directly. It calls the shell instead and the shell executes the command. The shell internally calls the `execve` command, and the environment variables of the calling process are passed to the shell and the shell passes it to the `execve` command.

2.5 Task 5: Environment variable and Set-UID Programs

```
Terminal
seed@VM:~$ gedit evar.c
seed@VM:~$ gcc -o evar evar.c
seed@VM:~$ sudo chown root evar
[sudo] password for seed:
seed@VM:~$ sudo chmod 4755 evar
seed@VM:~$ /bin/ls -l evar
-rwsr-xr-x 1 root seed 7396 Sep 18 08:55 evar
seed@VM:~$ export PATH=/home/seed:$PATH
seed@VM:~$ export LD_LIBRARY_PATH=xyz
seed@VM:~$ export myenv=aastha
seed@VM:~$ ./evar > evarout.txt
seed@VM:~$ grep PATH evarout.txt
XDG_SESSION_PATH=/org/freedesktop/DisplayManager/Session0
XDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat0
DEFAULTS_PATH=/usr/share/gconf/ubuntu.default.path
PATH=/home/seed:/home/seed/bin:/home/seed/.local/bin:/usr/local/sbin:/usr/local/
bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
MANDATORY_PATH=/usr/share/gconf/ubuntu.mandatory.path
COMPIZ_BIN_PATH=/usr/bin/
seed@VM:~$
```

Figure 7

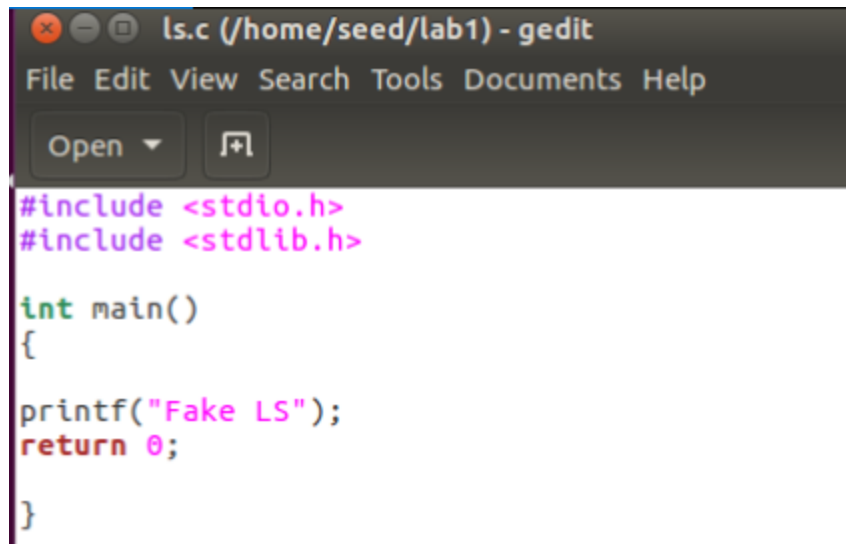
We create a program and compile it. Then we change the ownership of the program using `chown` command, make the program a set-UID program using `chmod` command. We can check this by running `ls -l` command. We then set 3 environment variables using the `export` command and then run the program. The environment variables `PATH` and `myenv` are inherited into the Set UID program. But the `LD_LIBRARY_PATH` environment variable is not inherited.

Observation: `LD_LIBRARY_PATH` is a path from which shared libraries are accessed and a privileged path which is automatically ignored if a Set UID program accesses it. It is a protection mechanism against malicious files being placed into shared libraries. There would be a predefined path from which the program accesses shared libraries which cannot be altered for Set UID programs.

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

```
root@VM:/home/seed/lab1# gcc -o ls ls.c
root@VM:/home/seed/lab1# export PATH=/home/seed:$PATH
root@VM:/home/seed/lab1# printenv PATH
/home/seed:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/game
s
root@VM:/home/seed/lab1# ./2.6
2.6 2.6.c ls ls.c
root@VM:/home/seed/lab1# /bin/ls
2.6 2.6.c ls ls.c
root@VM:/home/seed/lab1# gedit 2.6.c
```

Figure 8



```
ls.c (/home/seed/lab1) - gedit
File Edit View Search Tools Documents Help
Open [icon]

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

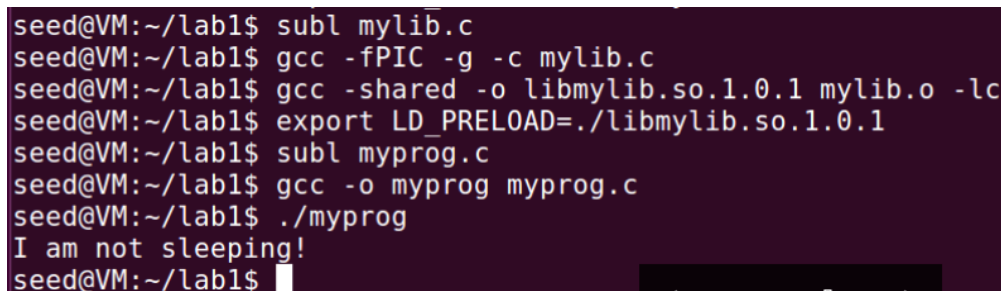
int main()
{
    printf("Fake LS");
    return 0;
}
```

Figure 9

A program 2.6.c is created with the system function containing ls and compiled. Then we change the ownership of the file using chown command, make it a Set-UID program using the chmod command. When we run the ls -l command, it shows that the given file is a Set-UID program. We then change the value of the PATH environment variable. We then create new program ls and compile it. This is the malicious file that we are placing in the path. This file is going to replace the functionality of the ls command. Hence the current working directory is printed as specified by the modified ls program.

Observation: The PATH environment variable looks for the command ls in the current directory first since it is specified. When it finds that ls exists, it runs that program instead of the shell ls command which proves to us that Set-UID programs may run malicious files with root privileges if the PATH variable is altered.

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



```
seed@VM:~/lab1$ subl mylib.c
seed@VM:~/lab1$ gcc -fPIC -g -c mylib.c
seed@VM:~/lab1$ gcc -shared -o libmylib.so.1.0.1 mylib.o -lc
seed@VM:~/lab1$ export LD_PRELOAD=./libmylib.so.1.0.1
seed@VM:~/lab1$ subl myprog.c
seed@VM:~/lab1$ gcc -o myprog myprog.c
seed@VM:~/lab1$ ./myprog
I am not sleeping!
seed@VM:~/lab1$
```

Figure 10

We create a program mylib.c, compile and make it a dynamic link library. We set the LD_PRELOAD environment variable using the export command to point to the DLL we just created. We then create a program myprog.c and compile it. When we run the myprog program,

the output is shown as above. It means that this program calls the mylib DLL that we just created instead of the lib.c DLL.

```
seed@VM:~/lab1$ sudo chown root myprog
[sudo] password for seed:
Sorry, try again.
[sudo] password for seed:
seed@VM:~/lab1$ sudo chmod 4755 myprog
seed@VM:~/lab1$ /bin/ls -l myprog
-rwsr-xr-x 1 root seed 7348 Sep 18 00:30 myprog
seed@VM:~/lab1$ ./myprog
seed@VM:~/lab1$ sudo su root
root@VM:/home/seed/lab1# export LD_PRELOAD=./libmylib.so.1.0.1
root@VM:/home/seed/lab1# printenv LD_PRELOAD
./libmylib.so.1.0.1
root@VM:/home/seed/lab1# ./myprog
I am not sleeping!
```

Figure 11

Observation: The above screenshot indicates that we are executing the myprog program from root account. It is a set-UID program owned by root. We set the LD_PRELOAD environment variable pointing to the DLL we created. When we run the program, the program calls the mylib dll we created.

```
root@VM:/home/seed/lab1# sudo adduser user1
Adding user `user1' ...
Adding new group `user1' (1001) ...
Adding new user `user1' (1001) with group `user1' ...
Creating home directory `/home/user1' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for user1
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
```

Figure 12

```
root@VM:/home/seed/lab1# sudo chown user1 myprog
root@VM:/home/seed/lab1# sudo chmod 4755 myprog
root@VM:/home/seed/lab1# /bin/ls -l myprog
-rwsr-xr-x 1 user1 seed 7348 Sep 18 00:30 myprog
root@VM:/home/seed/lab1# exit
exit
seed@VM:~/lab1$ sudo chown user1 myprog
[sudo] password for seed:
seed@VM:~/lab1$ sudo chmod 4755 myprog
seed@VM:~/lab1$ /bin/ls -l myprog
-rwsr-xr-x 1 user1 seed 7348 Sep 18 00:30 myprog
seed@VM:~/lab1$ export LD_PRELOAD=./libmylib.so.1.0.1
seed@VM:~/lab1$ ./myprog
seed@VM:~/lab1$ █
```

Figure 13

The above screenshots indicate that we created a new user seed1. We make the myprog program a set-UID program owned by seed1 user. We then set the LD_PRELOAD environment variable pointing to the DLL we created. When we execute the program from another user account like seed, the program sleeps for some time. This means that the program doesn't invoke the DLL we created.

Observation: The LD_PRELOAD environment variable is always ignored if a Set-UID program accesses it. It is basically a protection mechanism in UNIX. In the first case, myprog is a regular program and run by a normal user. Hence LD_PRELOAD is not ignored and the new malicious DLL file created by us is accessed. In the second case, myprog is a Set-UID root program and run by a normal user. Since it is a Set-UID program, LD_PRELOAD is ignored and the DLL file created by us isn't accessed, instead the default library file is accessed. In the third case, myprog is a Set-UID program and run by root. Here it checks for effective UID and real UID and since both are related to root, it trusts the DLL file and runs the DLL we created. In the last case, myprog is a Set-UID program owned by a user and run by another user. Hence LD_PRELOAD is ignored again, since it is a Set-UID program.

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


```

seed@VM:~/lab1$ subl 2.8.c
seed@VM:~/lab1$ gcc -o 2.8 2.8.c
seed@VM:~/lab1$ sudo chown root 2.8
[sudo] password for seed:
Sorry, try again.
[sudo] password for seed:
seed@VM:~/lab1$ sudo chmod 4755 2.8
seed@VM:~/lab1$ subl 2.8.c
seed@VM:~/lab1$ gcc -o 2.8 2.8.c
seed@VM:~/lab1$ sudo chown root 2.8
[sudo] password for seed:
seed@VM:~/lab1$ sudo chmod 4755 2.8
seed@VM:~/lab1$ subl xyz.txt
seed@VM:~/lab1$ subl xyz.txt
seed@VM:~/lab1$ sudo chown root:root xyz.txt
seed@VM:~/lab1$ /bin/ls -l xyz.txt
-rw-rw-r-- 1 root root 94 Sep 18 10:01 xyz.txt
seed@VM:~/lab1$ su user1
Password:
user1@VM:/home/seed/lab1$ ./2.8 "xyz.txt:rm xyz.txt"
/bin/cat: 'xyz.txt:rm': No such file or directory
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

}user1@VM:/home/seed/lab1$ cat xyz.txt
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{

printf("DUMMY");

}user1@VM:/home/seed/lab1$ exit
exit
seed@VM:~/lab1$ subl xyz.txt
seed@VM:~/lab1$ subl xyz.txt
seed@VM:~/lab1$ subl 2.8.c
seed@VM:~/lab1$ gcc -o 2.8 2.8.c
2.8.c: In function 'main':
2.8.c:17:2: warning: implicit declaration of function 'execve' [-Wimplicit-function-declaration]
   execve(v[0], v, NULL);
   ^
seed@VM:~/lab1$ subl 2.8.c
seed@VM:~/lab1$ gcc -o 2.8 2.8.c
seed@VM:~/lab1$ sudo chown root 2.8
seed@VM:~/lab1$ sudo chmod 4755 2.8

```

Figure 14

```

seed@VM:~/lab1$ sudo chmod 4755 2.8
seed@VM:~/lab1$ subl xyz.txt
seed@VM:~/lab1$ sudo chown root:root xyz.txt
seed@VM:~/lab1$ /bin/ls -l xyz.txt
-rw-rw-r-- 1 root root 5 Sep 18 10:13 xyz.txt
seed@VM:~/lab1$ su user1
Password:
user1@VM:/home/seed/lab1$ ./2.8 "xyz.txt:rm xyz.txt"
/bin/cat: 'xyz.txt:rm xyz.txt': No such file or directory
user1@VM:/home/seed/lab1$ ./2.8 xyz.txt:rm xyz.txt
/bin/cat: 'xyz.txt:rm': No such file or directory
user1@VM:/home/seed/lab1$ ./2.8 xyz.txt;rm xyz.txt
DUMMYrm: remove write-protected regular file 'xyz.txt'? y
rm: cannot remove 'xyz.txt': Permission denied
user1@VM:/home/seed/lab1$ █

```

Figure 15

We create a program with system command first and make it a Set-UID program. We then create a file that is owned by root and is an important file with no write privileges to other users. Now we login into user1 account and execute the program. The program displays the contents of the file and also deletes the file because of the rm command after the ;. Now we modify the code to have the execve command and comment the system command. When we execute the program with "", the program searches for the entire string. So a file by that name wouldn't exist. Now we try again without the "", the program executes only till the ; and displays the contents of the file. The remaining part of the argument consisting of the rm command is not executed since it does not have permissions.

Observation: When system() executes, it doesn't execute the command directly. It calls the shell instead and executes the command. So if the program is a Set-UID program, the user will have temporary root privileges and can remove any file he wants with root privileges. Multiple commands can be passed together using the "" and ;. System command calls the shell and the shell parses the string and handles "". Whereas execve command replaces the program with the called program and passes the argument strings exactly as specified and doesn't interpret quotes. So when we pass the something after the ; it is treated as a new command and root privileges would have been lost. So the rm command is executed using user1 privileges which is why it can't delete the file.

2.9 Task 9: Capability Leaking

```
seed@VM:~$ cd lab1
seed@VM:~/lab1$ subl 2.9.c
seed@VM:~/lab1$ gcc -o 2.9 2.9.c
seed@VM:~/lab1$ sudo chown root 2.9
[sudo] password for seed:
seed@VM:~/lab1$ sudo chmod 4755 2.9
seed@VM:~/lab1$ /bin/ls -l 2.9
-rwsr-xr-x 1 root seed 7636 Sep 18 10:47 2.9
seed@VM:~/lab1$ sudo su root
root@VM:/home/seed/lab1# cd /etc
root@VM:/etc# subl zzz
root@VM:/etc# cat zzz
Sample File
root@VM:/etc# /bin/ls -l zzz
-rw-r--r-- 1 root root 11 Sep 18 10:50 zzz
root@VM:/etc# exit
exit
seed@VM:~/lab1$ ./2.9
seed@VM:~/lab1$ cat zzz
Sample File
seed@VM:~/lab1$ cat /etc/zzz
Sample File
seed@VM:~/lab1$
```

Figure 16

We create a program 2.9.c, compile it and make it a Set-UID program owned by root. We then login as root and create a program zzz in the etc directory. We exit from the root account and execute the program from seed account. We find that the file /etc/zzz is modified by appending the content of the child process into the file.

Observation: The child inherits copies of the parent's set of open file descriptors during fork call. Each file descriptor in the child refers to the same open file description as the corresponding file descriptor in the parent. So, the privileges that the parent gained weren't downgraded and hence the child could also access the file /etc/zzz. To avoid such attacks, the file descriptor has to be closed before the fork call.