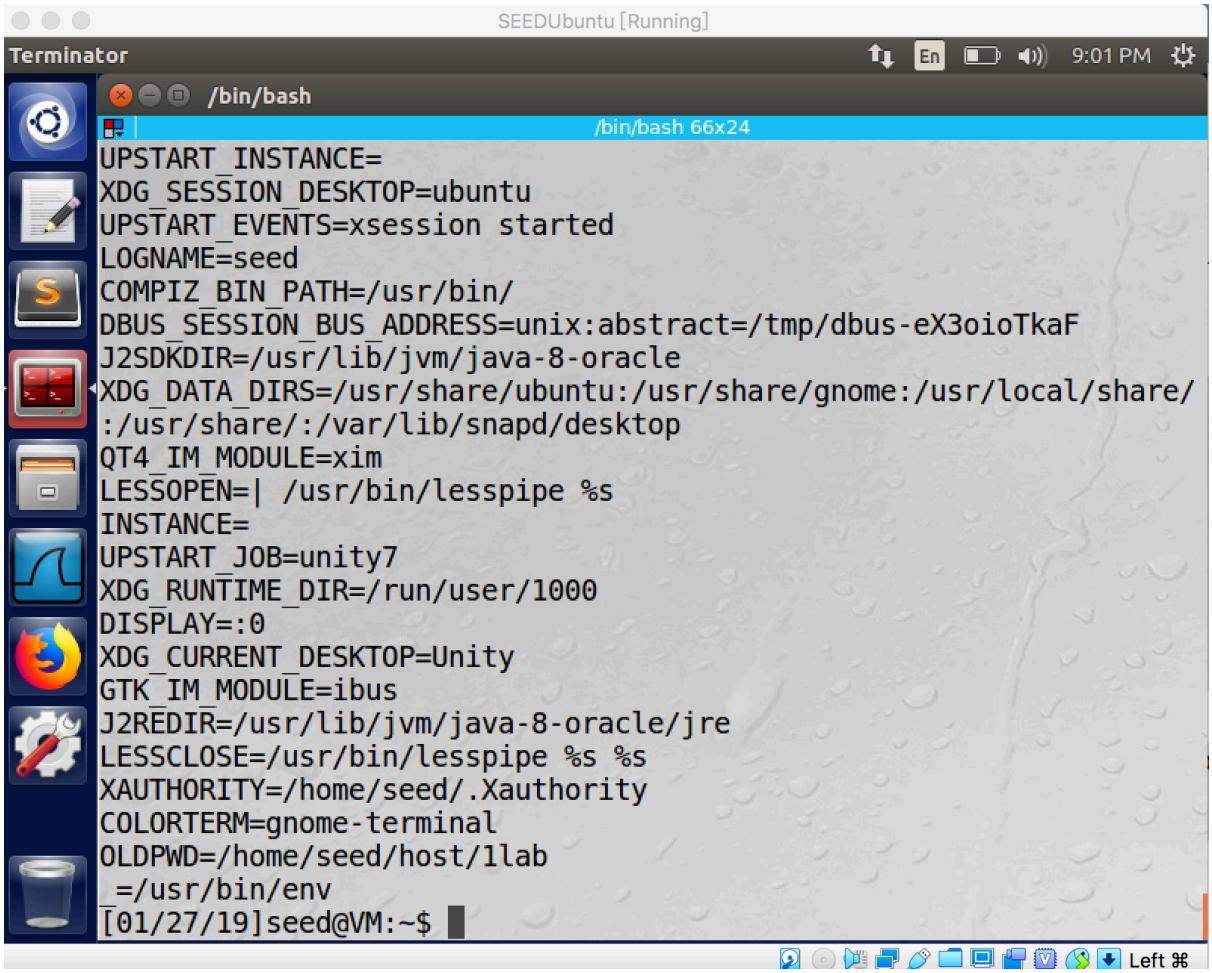
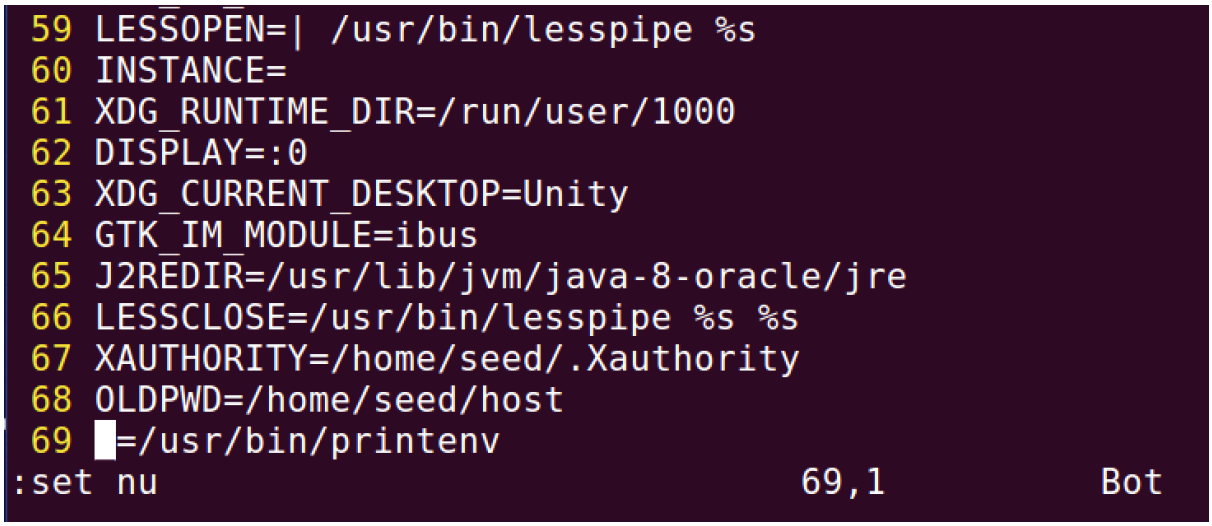
201P Lab 1

- 2.1 Task 1: Manipulating Environment Variables

1) With printenv or env, see 69 environment variables:





Observations: :set nu

or env command is utilized to print all environment variables.

P.S.: It seems no parameter to print the line number directly. I use cat + vim to check the amount.

View some particular environment variables, such as PWD:

```
[01/27/19]seed@VM:~/.../1lab$ env | grep PWD
PWD=/home/seed/host/1lab
OLDPWD=/home/seed/host
```

Observations: "printenv PWD" or "env | grep PWD" command can be used to print particular environment variables.

2) Use export and unset to set or unset environment variables.

```
[01/27/19]seed@VM:~$ export lab1env='Lab One Env'
[01/27/19]seed@VM:~$ echo $lab1env
Lab One Env
[01/27/19]seed@VM:~$ unset lab1env
[01/27/19]seed@VM:~$ echo $lab1env

[01/27/19]seed@VM:~$ lab1env='Lab One Env'
[01/27/19]seed@VM:~$ echo $lab1env
Lab One Env
[01/27/19]seed@VM:~$ unset lab1env
[01/27/19]seed@VM:~$ echo $lab1env

[01/27/19]seed@VM:~$
```

Observations: export command and Variable=’’ can be used to set values to environment variables. (Be cautious that there cannot have space between the equal sign.)

Unset command can be used to unset environment variables.

P.S.: It seems more traditional and eternal to change environment variables in file ~/.bashrc.

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

```
[01/27/19]seed@VM:~/.../lab1$ gcc 2.2.1.c
[01/27/19]seed@VM:~/.../lab1$ ll
total 12
-rw-r--r-- 1 seed seed  374 Jan 27 22:36 2.2.1.c
-rwxr-xr-x 1 seed seed 7496 Jan 27 22:37 a.out
[01/27/19]seed@VM:~/.../lab1$ a.out > child
[01/27/19]seed@VM:~/.../lab1$ vi child
```

```
[01/27/19]seed@VM:~/.../lab1$ vi 2.2.1.c
[01/27/19]seed@VM:~/.../lab1$ gcc 2.2.1.c
[01/27/19]seed@VM:~/.../lab1$ a.out > parent
[01/27/19]seed@VM:~/.../lab1$ vi parent
[01/27/19]seed@VM:~/.../lab1$ diff parent child
[01/27/19]seed@VM:~/.../lab1$
```

Observations: Compile and run the program and this prints out the environment variables of child process first and then comment and unset comment to prints out environment variables of parent process.

The results show that 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.

- 2.3 Task 3: Environment Variables and execve()

```
[01/27/19]seed@VM:~/.../lab1$ ll -rst
total 24
4 -rw-r--r-- 1 seed seed 4017 Jan 27 22:38 child
4 -rw-r--r-- 1 seed seed  374 Jan 27 22:40 2.2.1.c
4 -rw-r--r-- 1 seed seed 4017 Jan 27 22:40 parent
4 -rw-r--r-- 1 seed seed  189 Jan 27 23:31 2.3.1.c
8 -rwxr-xr-x 1 seed seed 7396 Jan 27 23:31 a.out
[01/27/19]seed@VM:~/.../lab1$ ./a.out
[01/27/19]seed@VM:~/.../lab1$ vi 2.3.1.c
[01/27/19]seed@VM:~/.../lab1$ gcc 2.3.1.c
2.3.1.c: In function ‘main’:
2.3.1.c:9:2: warning: implicit declaration of function
‘execve’ [-Wimplicit-function-declaration]
  execve("/usr/bin/env", argv, environ);
  ^
[01/27/19]seed@VM:~/.../lab1$ ./a.out
XDG_VTNR=7
XDG_SESSION_ID=c1
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed
CLUTTER_IM_MODULE=xim
SESSION=ubuntu
```

Observations: Create a program using “int execve(const char *filename, char *const argv[], char *const envp[]);”, a kernel-level system call, where the 3rd argument of the execve(), envp, is an array of strings, conventionally of the form key=value, which are passed as environment to the new program. 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

In step 1 “execve("/usr/bin/env", argv, NULL);//1”, the environment variables are set to NULL, thus the output of this program print nothing with only the shell returned.

When changed to “execve("/usr/bin/env", argv, environ);” in step 2, the environment variable is set to be **environ, the new program gets its environment variables, so the output of this program is all the environment variables.

The results show that when the environment variables argument of the execve() command is set to NULL, the exec one just get the environment variable here and thus nothing, so the calling process doesn’t inherit any valid environment variables. But when the

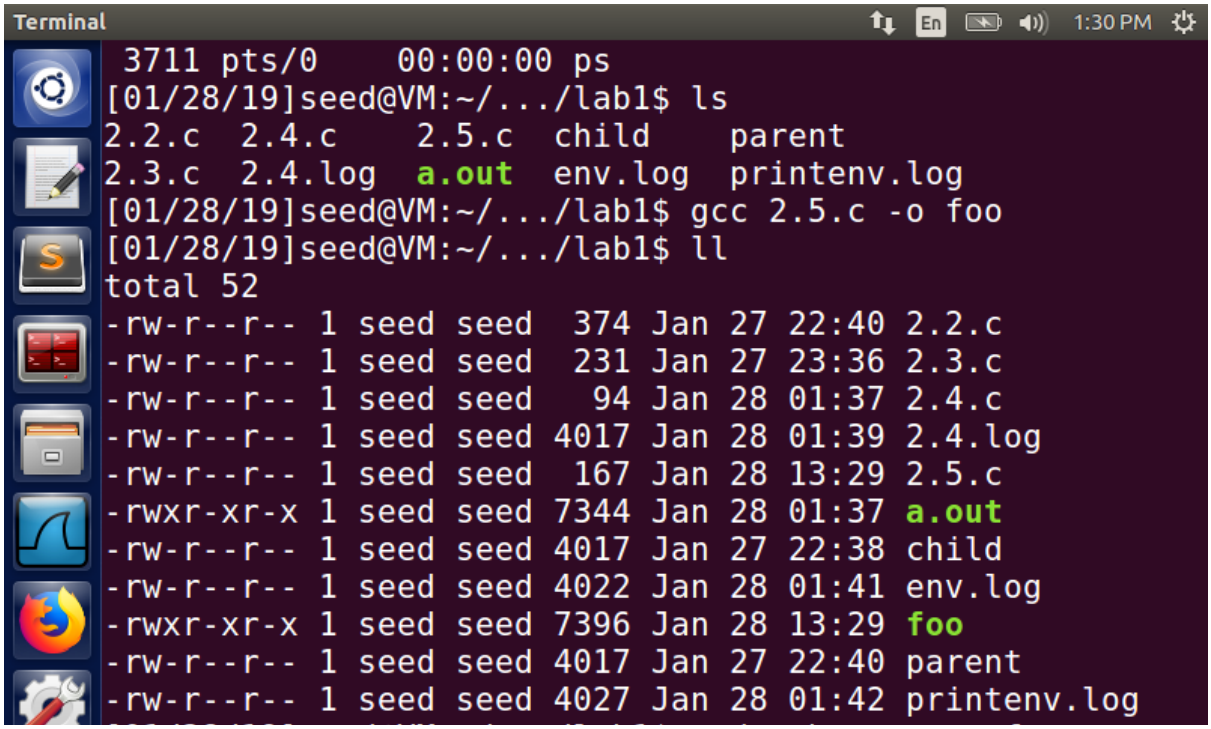
argument takes the environment variables, they are stored in memory, and then the calling process inherits the environment variables.

- Task 4: Environment Variables and system()

```
[01/28/19]seed@VM:~/.../lab1$ mv 2.2.1.c 2.2.c
[01/28/19]seed@VM:~/.../lab1$ mv 2.3.1.c 2.3.c
[01/28/19]seed@VM:~/.../lab1$ vi 2.4.c
[01/28/19]seed@VM:~/.../lab1$ gcc 2.4.c
[01/28/19]seed@VM:~/.../lab1$ ./a.out
LESSOPEN=| /usr/bin/lesspipe %s
GNOME_KEYRING_PID=
USER=seed
LANGUAGE=en_US
J2SDKDIR=/usr/lib/jvm/java-8-oracle
XDG_SEAT=seat0
SESSION=ubuntu
XDG_SESSION_TYPE=x11
COMPIZ_CONFIG_PROFILE=ubuntu
LD_LIBRARY_PATH=/home/seed/source/boost_1_64_0/stage/li
b:/home/seed/source/boost_1_64_0/stage/lib:
SHLVL=1
LIBGL_ALWAYS_SOFTWARE=1
J2REDIR=/usr/lib/jvm/java-8-oracle/jre
HOME=/home/seed
QT4_IM_MODULE=xim
OLDPWD=/home/seed/host
```

Observation: As designed, when run this a.out, the inner function system() executes, instead of executing the command directly, it uses execl(), which calls execve() to execute /bin/sh, while passing the environment variables array. And then asks the shell to execute the command of env. We can observe the PIDs and PPIDs of the processes. So the output is the environment variables.

- 2.5 Task 5 : Environment Variable and Set-UID Programs




```
[01/29/19]seed@VM:~/lab1$ sudo chown root foo
[sudo] password for seed:
[01/29/19]seed@VM:~/lab1$ sudo chmod 4755 foo
[01/29/19]seed@VM:~/lab1$ ll foo
-rwsr-xr-x 1 root seed 7396 Jan 28 16:31 foo
[01/29/19]seed@VM:~/lab1$
```

compile and change its ownership to root, and make it a Set-UID program. Check that we're in a normal user accout (ctrl+D, exit, logout).

```
[01/29/19]seed@VM:~/lab1$ export PATH=/home/seed:$PATH
[01/29/19]seed@VM:~/lab1$ export LD_LIBRARY_PATH=task2_5
[01/29/19]seed@VM:~/lab1$ export myenv=task2_5_PATH
[01/29/19]seed@VM:~/lab1$ ./foo>foo.txt
[01/29/19]seed@VM:~/lab1$ grep PATH foo.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:/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-l
inux/tools:/home/seed/android/android-sdk-linux/platform-tools:/ho
me/seed/android/android-ndk/android-ndk-r8d:/home/seed/.local/bin
MANDATORY_PATH=/usr/share/gconf/ubuntu.mandatory.path
COMPIZ_BIN_PATH=/usr/bin/
myenv=task2_5_PATH
[01/29/19]seed@VM:~/lab1$
```

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

Observation: When 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.

It is because that 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

```
[01/28/19]seed@VM:~/.../lab1$ vi 2.6.c
[01/28/19]seed@VM:~/.../lab1$ gcc 2.6.c -o 2_6
2.6.c: In function 'main':
2.6.c:3:5: warning: implicit declaration of function 'system' [-Wi
mplicit-function-declaration]
     system("ls");
     ^
[01/28/19]seed@VM:~/.../lab1$ vi 2.6.c
[01/28/19]seed@VM:~/.../lab1$ gcc 2.6.c -o 2_6
```

```
[01/29/19]seed@VM:~/lab1$ sudo chown root 2_6
[01/29/19]seed@VM:~/lab1$ sudo chmod 4755 2_6
[01/29/19]seed@VM:~/lab1$ ls 2_6
2_6
[01/29/19]seed@VM:~/lab1$ ll 2_6
-rwsr-xr-x 1 root seed 7344 Jan 28 16:31 2_6
[01/29/19]seed@VM:~/lab1$
```

compile the above program, and change its owner to root, and make it a Set-UID program.

Considering the countermeasure in Ubuntu 16.04, We use the following commands to link /bin/sh to zsh：

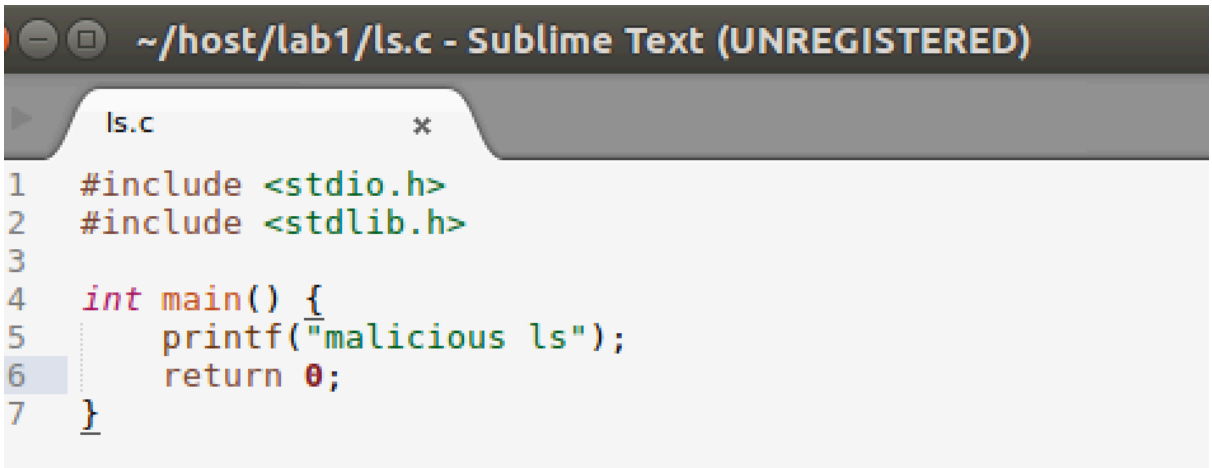
```
[01/28/19]seed@VM:~/.../lab1$ sudo rm /bin/sh
```

```
[01/28/19]seed@VM:~/.../lab1$ sudo ln -s /bin/zsh /bin/sh
```

Then we first run ls in bash and in our 2.6 program:

```
[01/29/19]seed@VM:~/lab1$ ls
2.2.c     2_6        2.9.c     libmylib.so.1.0.1  myprog.c
2.3.c     2.6.c      a.out     ls                 parent
2.4       2.8        child     ls.c               printenv.log
2.4.c     2.8.c      env.log   mylib.c            xyz.txt
2.4.log   2.8.exec   foo       mylib.o
2.5.c     2.9        foo.txt   myprog2_7
[01/29/19]seed@VM:~/lab1$ ./2_6
2.2.c     2_6        2.9.c     libmylib.so.1.0.1  myprog.c
2.3.c     2.6.c      a.out     ls                 parent
2.4       2.8        child     ls.c               printenv.log
2.4.c     2.8.c      env.log   mylib.c            xyz.txt
2.4.log   2.8.exec   foo       mylib.o
2.5.c     2.9        foo.txt   myprog2_7
[01/29/19]seed@VM:~/lab1$
```

To see this Set-UID program run your code instead of /bin/ls, we create and compile a new ls program. This is the malicious file that shown up in this path (with adding this path the env). 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:




```
[01/29/19]seed@VM:~/lab1$ export PATH=/home/seed/lab1:$PATH
[01/29/19]seed@VM:~/lab1$ ./2_6
malicious ls[01/29/19]seed@VM:~/lab1$ ls
malicious ls[01/29/19]seed@VM:~/lab1$ ls
malicious ls[01/29/19]seed@VM:~/lab1$
```

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 :

1) build a dynamic link library, mylib.c, to overrides the sleep() function in libc. Complie it using:

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

set the LD PRELOAD environment variable:

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

compile myprog, in the same path as libmylib.so.1.0.1

```
[01/28/19]seed@VM:~/.../lab1$ vi mylib.c
[01/28/19]seed@VM:~/.../lab1$ gcc -fPIC -g -c mylib.c
[01/28/19]seed@VM:~/.../lab1$ gcc -shared -o libmylib.so.1.0.1 myl
ib.o -lc
[01/28/19]seed@VM:~/.../lab1$ export LD_PRELOAD=./libmylib.so.1.0.
1
[01/28/19]seed@VM:~/.../lab1$ vi myprog.c
[01/28/19]seed@VM:~/.../lab1$ gcc -o myprog2_7 myprog.c
myprog.c: In function ‘main’:
myprog.c:4:1: warning: implicit declaration of function ‘sleep’ [-
Wimplicit-function-declaration]
 sleep(1);
 ^
[01/28/19]seed@VM:~/.../lab1$ ./myprog2_7
I am not sleeping!
[01/28/19]seed@VM:~/.../lab1$ ls -rst
total 112
4 child    4 env.log       4 2.6.c    8 libmylib.so.1.0.1
4 2.2.c    4 printenv.log  8 2_6      4 myprog.c
4 parent   4 2.5.c         4 ls.c     8 myprog2_7
4 2.3.c    8 a.out         8 ls
```

2) run myprog under the following conditions

I. condition 1: Make myprog a regular program, and run it as a normal user.

```
[01/28/19]seed@VM:~/lab1$ ll myprog2_7
-rwxrwxr-x 1 seed seed 7348 Jan 28 16:49 myprog2_7
[01/28/19]seed@VM:~/lab1$ ./myprog2_7
I am not sleeping!
[01/28/19]seed@VM:~/lab1$
```

While setting the LD_PRELOAD environment variable to point to the libmylib.so.1.0.1 we just created, it means that this program calls the mylib.o that we just created instead of the lib.c. So in this case, when calling sleep(), the output is our malicious one.

II. condition 2: Make myprog a Set-UID root program, and run it as a normal user.

```
[01/28/19]seed@VM:~/lab1$ sudo chown root myprog2_7
[sudo] password for seed:
[01/28/19]seed@VM:~/lab1$ sudo chmod 4755 myprog2_7
[01/28/19]seed@VM:~/lab1$ ll myprog2_7
-rwsr-xr-x 1 root seed 7348 Jan 28 16:49 myprog2_7
[01/28/19]seed@VM:~/lab1$ ./myprog2_7
[01/28/19]seed@VM:~/lab1$
```

set myprog to a Set-UID root program, it sleeps for a while.

III. condition 3: Make myprog a Set-UID root program, export the LD PRELOAD environment variable again in the root account and run it.

```
[01/28/19]seed@VM:~/lab1$ sudo su root
[sudo] password for seed:
root@VM:/home/seed/lab1# export LD_PRELOAD=./libmylib.so.1.0.1
root@VM:/home/seed/lab1# env|grep LD_
LD_PRELOAD=./libmylib.so.1.0.1
LD_LIBRARY_PATH=/home/seed/source/boost_1_64_0/stage/lib:/home/see
d/source/boost_1_64_0/stage/lib:
root@VM:/home/seed/lab1# ./myprog2_7
I am not sleeping!
root@VM:/home/seed/lab1#
```

Based on II, go to the root account, set the LD_PRELOAD environment variable pointing to the libmylib.so.1.0.1 we created, and run the program. As shown, it calls the libmylib we created.

IV. condition 4: 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.

```
[01/28/19]seed@VM:~/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
[01/28/19]seed@VM:~/lab1$ sudo chown user1 myprog2_7
[01/28/19]seed@VM:~/lab1$ sudo chmod 4755 myprog2_7
[01/28/19]seed@VM:~/lab1$ ll myprog2_7
-rwsr-xr-x 1 user1 seed 7348 Jan 28 16:49 myprog2_7
[01/28/19]seed@VM:~/lab1$ export LD_PRELOAD=./libmylib.so.1.0.1
[01/28/19]seed@VM:~/lab1$ ./myprog2_7
[01/28/19]seed@VM:~/lab1$
```

Creat a new user seed1 by “sudo adduser use1”. Then change myprog program to a set-UID program owned by seed1 user. Set the LD_PRELOAD environment variable pointing to the libmylib.so.1.0.1 we created, and run the program. As shown, the program sleeps for some time. This means that the program doesn’t invoke the libmylib we created.

In conlusion, the LD_PRELOAD environment variable is always ignored if a Set-UID program accesses it. It is basically a protection mechanism in UNIX. For detail:

In condition I, myprog is a regular program and run by a normal user. Hence LD_PRELOAD is not ignored and thus access the the new malicious LD_PRELOAD file created by us.

In condition II, 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 condition III, 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 condition IV, myprog is a Set-UID program owned by a user and run by another user. Hence LD_PRELOAD is ignored again, since is it a Set-UID program.

- 2.8 Task 8 : Task 8: Invoking External Programs Using system() versus execve()

```
[01/28/19]seed@VM:~/lab1$ vi 2.8.c
[01/28/19]seed@VM:~/lab1$ gcc 2.8.c -o 2.8
[01/28/19]seed@VM:~/lab1$ sudo chown root 2.8
[sudo] password for seed:
[01/28/19]seed@VM:~/lab1$ sudo chmod 4755 2.8
[01/28/19]seed@VM:~/lab1$ ll 2.8
-rwsr-xr-x 1 root seed 7544 Jan 28 18:38 2.8
[01/28/19]seed@VM:~/lab1$
```

Compile the above program, make it a root-owned Set-UID program.

To test the system(), we 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 run program 2.8. The program displays the contents of the file and also deletes the file because of the rm command after the ; .

```
[01/28/19]seed@VM:~/lab1$ vi xyz.txt
[01/28/19]seed@VM:~/lab1$ sudo chown root:root xyz.txt
[01/28/19]seed@VM:~/lab1$ ll xyz.txt
-rw-rw-r-- 1 root root 20 Jan 28 18:47 xyz.txt
[01/28/19]seed@VM:~/lab1$ su user1
Password:
```

```
user1@VM:/home/seed/lab1$ ./2.8 "xyz.txt;rm xyz.txt"
important!No change
user1@VM:/home/seed/lab1$ ls
2.2.c    2_6    child              ls          myprog.c
2.3.c    2.6.c  env.log            ls.c        parent
2.4.c    2.8    foo                mylib.c     printenv.log
2.4.log  2.8.c  foo.txt            mylib.o
2.5.c    a.out  libmylib.so.1.0.1  myprog2 7
user1@VM:/home/seed/lab1$ cat xyz.txt
cat: xyz.txt: No such file or directory
user1@VM:/home/seed/lab1$
```

Now on we exit user1 account and modify the code to use the execve() command and comment the system command. Compile the program, and make it a root-owned Set-UID. Lock the permission of xyz.txt and come to user1 account. When we execute the program with “”, the program searches for the entire string and the result is that a file by that name wouldn’t exist. Next 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.

```
user1@VM:/home/seed/lab1$ exit
exit
[01/28/19]seed@VM:~/lab1$ vi xyz.txt
[01/28/19]seed@VM:~/lab1$ vi 2.8
2.8    2.8.c
[01/28/19]seed@VM:~/lab1$ vi 2.8.c
[01/28/19]seed@VM:~/lab1$ gcc -o 2.8.exec 2.8.c
2.8.c: In function ‘main’:
2.8.c:17:1: warning: implicit declaration of function ‘execve’ [-W
implicit-function-declaration]
 execve(v[0], v, NULL);
 ^
[01/28/19]seed@VM:~/lab1$ sudo chown root 2.8.exec
[sudo] password for seed:
[01/28/19]seed@VM:~/lab1$ sudo chmod 4755 2.8.exec
[01/28/19]seed@VM:~/lab1$ ll 2.8.exec
-rwsr-xr-x 1 root seed 7544 Jan 28 21:38 2.8.exec
[01/28/19]seed@VM:~/lab1$ sudo chown root:root xyz.txt
[01/28/19]seed@VM:~/lab1$ ll xyz.txt
-rw-rw-r-- 1 root root 22 Jan 28 21:37 xyz.txt
[01/28/19]seed@VM:~/lab1$ su user1
Password:
user1@VM:/home/seed/lab1$
```

```
user1@VM:/home/seed/lab1$ ./2.8.exec "xyz.txt;rm xyz.txt"
/bin/cat: 'xyz.txt;rm xyz.txt': No such file or directory
user1@VM:/home/seed/lab1$ ./2.8.exec ^Cz.txt;rm xyz.txt
user1@VM:/home/seed/lab1$ ll xyz.txt
-rw-rw-r-- 1 root root 22 Jan 28 21:46 xyz.txt
user1@VM:/home/seed/lab1$ ./2.8.exec xyz.txt;rm xyz.txt
important!No change.

rm: remove write-protected regular file 'xyz.txt'? y
rm: cannot remove 'xyz.txt': Permission denied
user1@VM:/home/seed/lab1$ ll xyz.txt
-rw-rw-r-- 1 root root 22 Jan 28 21:46 xyz.txt
user1@VM:/home/seed/lab1$
```

Observation: In the first case, system() command is not executed directly, but call the shell instead of executing the command. Therefore, if the program is a Set UID program, the user has temporary root privileges and can delete all the files needed for root privileges. You can use multiple commands with "" and ";", while the system() command calls the shell, and the shell parses the string "". However, if the execve() command replaces the program with the calling program, the parameter string will be passed as specified and the quotes will not be interpreted. If you submit something later, it will be treated as a new command and root privileges will be lost. Therefore, the rm command runs with user right1, so you cannot delete the file.

- 2.9 Task 9: Capability Leaking

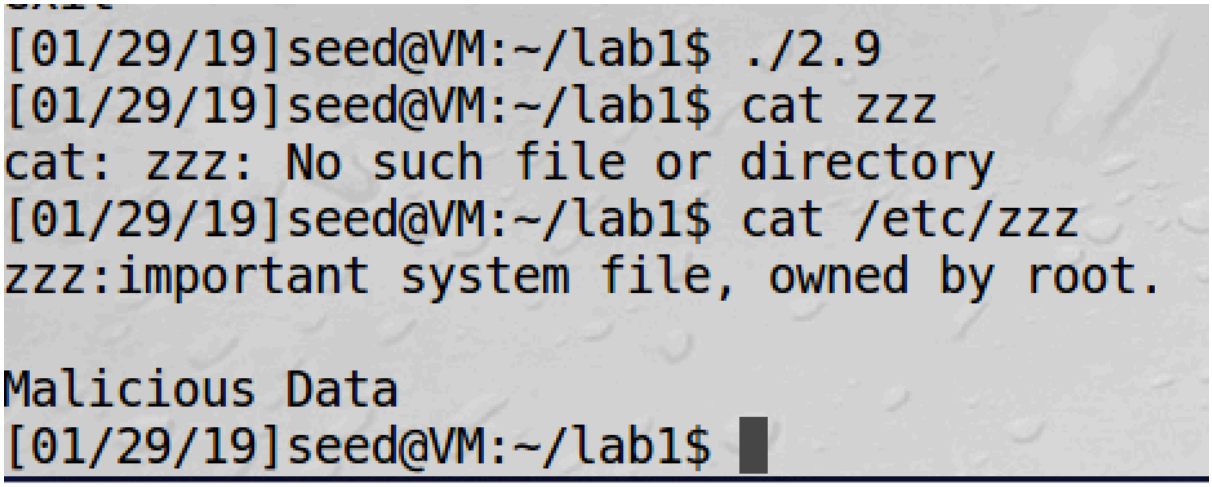
```
[01/29/19]seed@VM:~/lab1$ vi 2.9.c
[01/29/19]seed@VM:~/lab1$ gcc 2.9.c -o 2.9
2.9.c: In function ‘main’:
2.9.c:16:1: warning: implicit declaration of function ‘sleep’ [-Wi
mplicit-function-declaration]
 sleep(1);
 ^
2.9.c:19:1: warning: implicit declaration of function ‘setuid’ [-W
implicit-function-declaration]
 setuid(getuid());  /* getuid() returns the real uid */
 ^
2.9.c:19:8: warning: implicit declaration of function ‘getuid’ [-W
implicit-function-declaration]
 setuid(getuid());  /* getuid() returns the real uid */
        ^
2.9.c:20:5: warning: implicit declaration of function ‘fork’ [-Wim
plicit-function-declaration]
 if (fork()) { /* In the parent process */
     ^
2.9.c:21:3: warning: implicit declaration of function ‘close’ [-Wi
mplicit-function-declaration]
```

```
2.9.c:21:3: warning: implicit declaration of function ‘close’ [-Wi
mplicit-function-declaration]
   close (fd);
   ^
2.9.c:27:3: warning: implicit declaration of function ‘write’ [-Wi
mplicit-function-declaration]
   write (fd, "Malicious Data\n", 15);
   ^
[01/29/19]seed@VM:~/lab1$ sudo chown root 2.9
[sudo] password for seed:
[01/29/19]seed@VM:~/lab1$ sudo chmod 4755 2.9
[01/29/19]seed@VM:~/lab1$ ll 2.9
-rwsr-xr-x 1 root seed 7636 Jan 29 00:42 2.9
[01/29/19]seed@VM:~/lab1$
```

```
[01/29/19]seed@VM:~/lab1$ sudo su
root@VM:/home/seed/lab1# vi /etc/zzz
root@VM:/home/seed/lab1# cat /etc/zzz
zzz:important system file, owned by root.

root@VM:/home/seed/lab1# ll /etc/zzz
-rw-r--r-- 1 root root 43 Jan 29 00:49 /etc/zzz
root@VM:/home/seed/lab1# exit
exit
[01/29/19]seed@VM:~/lab1$
```

Compile the program, change its owner to root, and make it a Set-UID program. In root account create /etc/zzz, an important system file, owned by root with permission 0644. Run the program as a normal user.



Observation: The result shows that the important file / etc / zzz has been modified by adding the contents of the child process to the file. The reason is that during a fork call, the child process inherits a copy of the descriptor set of the parent file's open file. Each file descriptor in a child node references the same description of the open file as the corresponding file descriptor in the parent node. Therefore, the privileges received from the parent are not degraded, so the child can also access the / etc / zzz file. To avoid such an attack, you must close the file descriptor before the fork call.