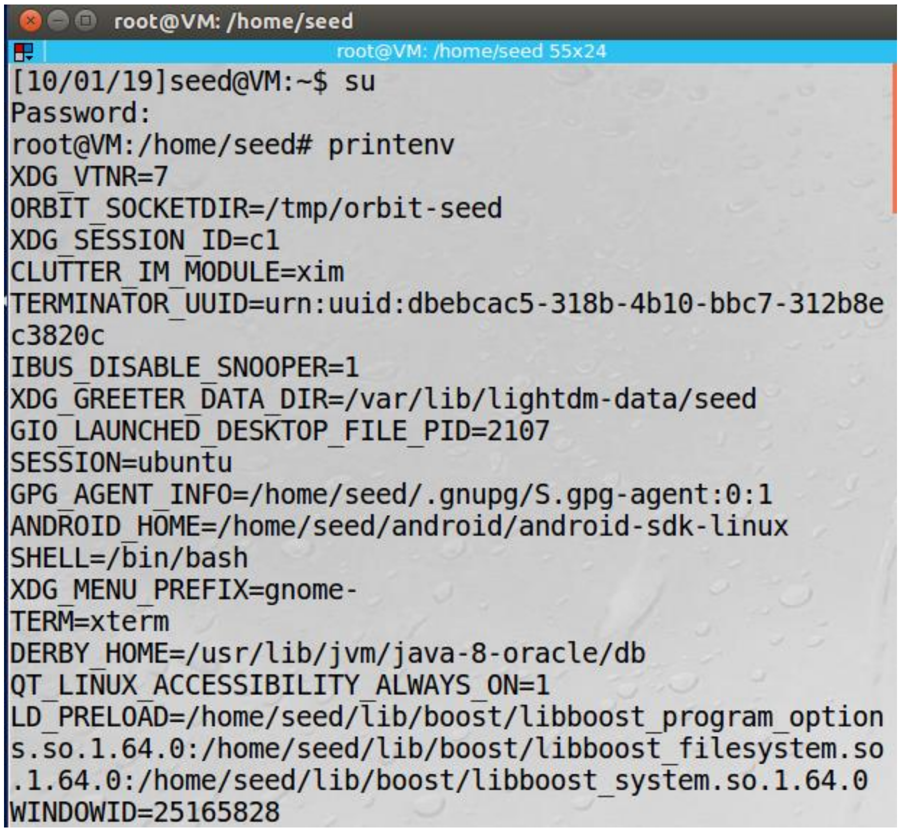
CSC 681 PRINCIPLES OF COMPUTER SECURITY
MOULIKA BOLLINADI

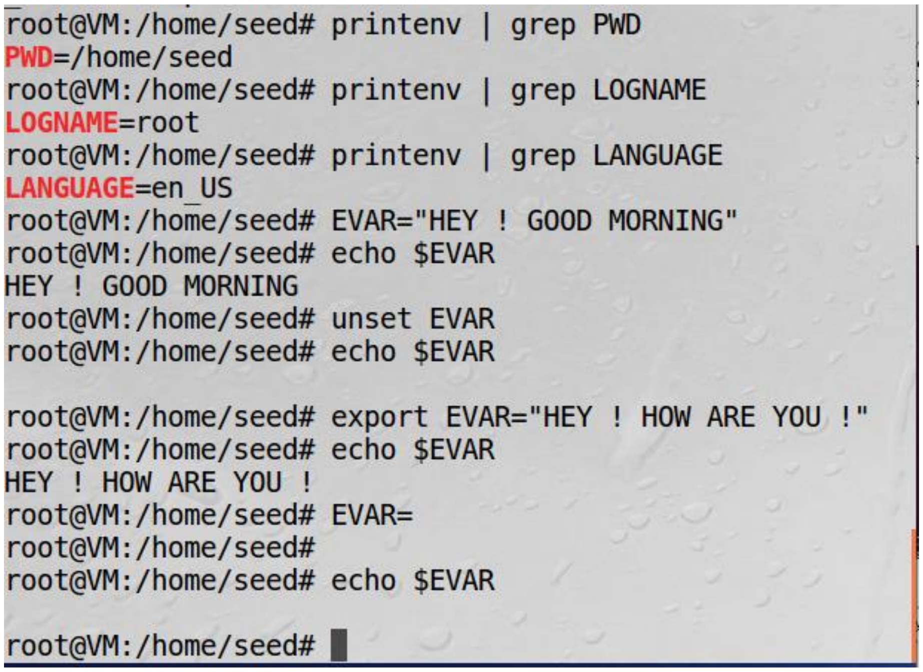
Task-1: Manipulating Environment Variables



root@VM: /home/seed

root@VM: /home/seed 55x24

```
[10/01/19]seed@VM:~$ su
Password:
root@VM:/home/seed# printenv
XDG_VTNR=7
ORBIT_SOCKETDIR=/tmp/orbit-seed
XDG_SESSION_ID=c1
CLUTTER_IM_MODULE=xim
TERMINATOR_UUID=urn:uuid:dbebcac5-318b-4b10-bbc7-312b8e
c3820c
IBUS_DISABLE_SNOOPER=1
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed
GIO_LAUNCHED_DESKTOP_FILE_PID=2107
SESSION=ubuntu
GPG_AGENT_INFO=/home/seed/.gnupg/S.gpg-agent:0:1
ANDROID_HOME=/home/seed/android/android-sdk-linux
SHELL=/bin/bash
XDG_MENU_PREFIX=gnome-
TERM=xterm
DERBY_HOME=/usr/lib/jvm/java-8-oracle/db
QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
LD_PRELOAD=/home/seed/lib/boost/libboost_program_option
s.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=25165828
```



```
root@VM:/home/seed# printenv | grep PWD
PWD=/home/seed
root@VM:/home/seed# printenv | grep LOGNAME
LOGNAME=root
root@VM:/home/seed# printenv | grep LANGUAGE
LANGUAGE=en_US
root@VM:/home/seed# EVAR="HEY ! GOOD MORNING"
root@VM:/home/seed# echo $EVAR
HEY ! GOOD MORNING
root@VM:/home/seed# unset EVAR
root@VM:/home/seed# echo $EVAR

root@VM:/home/seed# export EVAR="HEY ! HOW ARE YOU !"
root@VM:/home/seed# echo $EVAR
HEY ! HOW ARE YOU !
root@VM:/home/seed# EVAR=
root@VM:/home/seed#
root@VM:/home/seed# echo $EVAR

root@VM:/home/seed# 
```

a.

Did: I used the printenv or env command to print all global environment variables and 'grep' command to print specific environment variables. I logged into root first using 'su' command and entered the password to login into the root.

Saw: I observed the outputs such as path for the PWD and root as a LOGNAME for account I'm logged into.

Learned: I learned how to print the output for the specific environment variables, how to print all the environment variables and how to log into the root account.

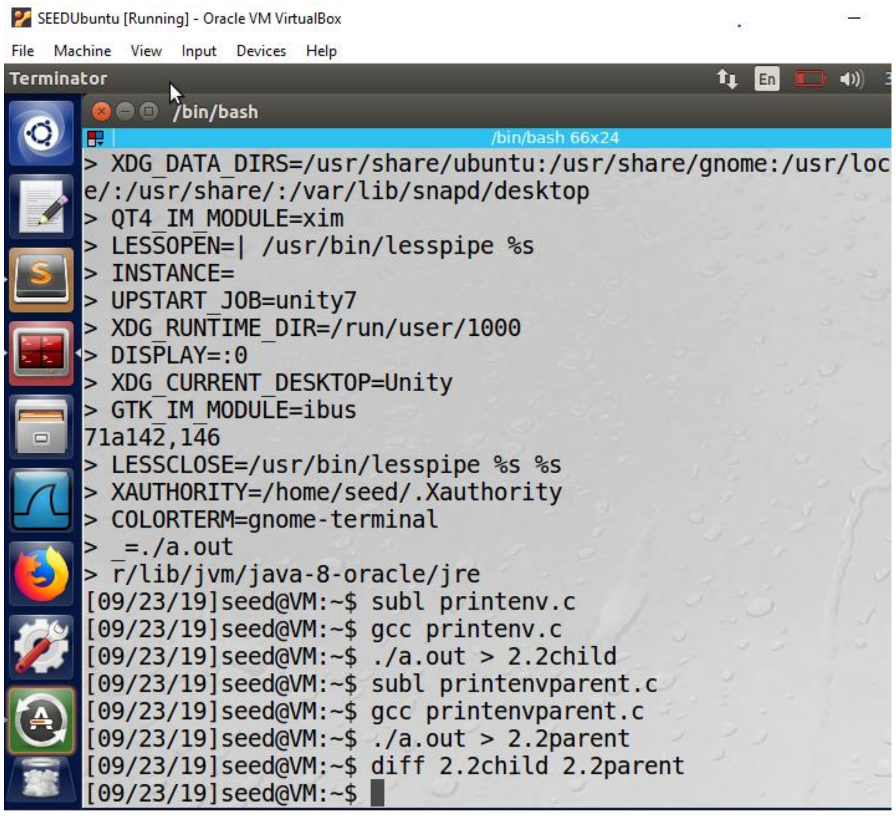
b.

Did: To set local variable, I used the command 'EVAR' and the 'echo' command to print the local variable that I set. I used unset command to unset the local variable that I set. I used 'EXPORT' command to set the environment variable.

Saw: When I used 'unset' command, I observed that the environment variable that I set previously has been unset and gives an empty output. After setting the environment variable using export command, even without resetting, if we set a new variable using EVAR, it prints the newly set variable.

Learned: The export and set command have the similar functionality, but the difference is: EVAR is used to set the variable which is accessible to any process run from the shell. EXPORT is used to specify the variable for that specific process which means that the scope of the variable is restricted to the shell and no other process can access it.

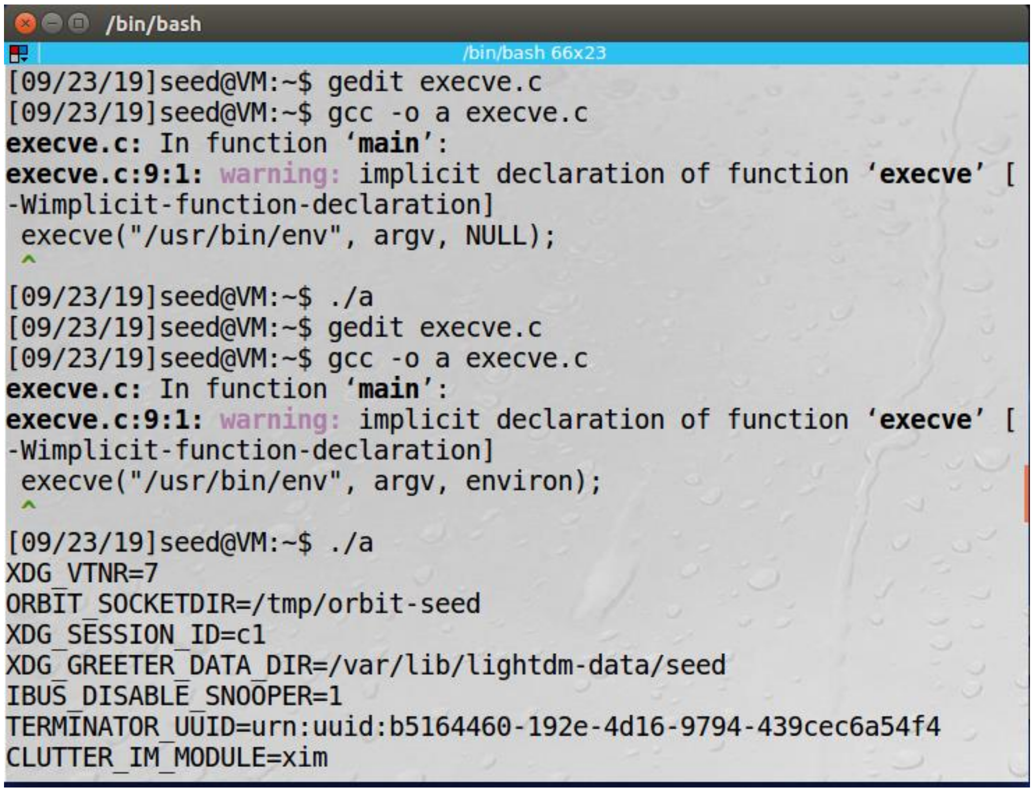
Task-2: Passing Environment Variables from Parent Process to Child Process.


SEEDUbuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminator
/bin/bash
/bin/bash 66x24

```
> XDG_DATA_DIRS=/usr/share/ubuntu:/usr/share/gnome:/usr/loc
e/:/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
71a142,146
> LESSCLOSE=/usr/bin/lesspipe %s %s
> XAUTHORITY=/home/seed/.Xauthority
> COLORTERM=gnome-terminal
> _=./a.out
> r/lib/jvm/java-8-oracle/jre
[09/23/19]seed@VM:~$ subl printenv.c
[09/23/19]seed@VM:~$ gcc printenv.c
[09/23/19]seed@VM:~$ ./a.out > 2.2child
[09/23/19]seed@VM:~$ subl printenvparent.c
[09/23/19]seed@VM:~$ gcc printenvparent.c
[09/23/19]seed@VM:~$ ./a.out > 2.2parent
[09/23/19]seed@VM:~$ diff 2.2child 2.2parent
[09/23/19]seed@VM:~$
```

Did: I compiled and ran the program to understand how the child process gets the environment variables from parent process. In-order to compare the obtained outputs, I saved the output into a file such as a.out where a.out is an executable filename and then used a diff command to compare them.
Saw: I saw several lines of strings which are different environment variables.
Learned: I did not see any difference in between two output files. I understood and learned that, while using fork(), the child process will automatically inherit the environment variable conditions from the parent process.

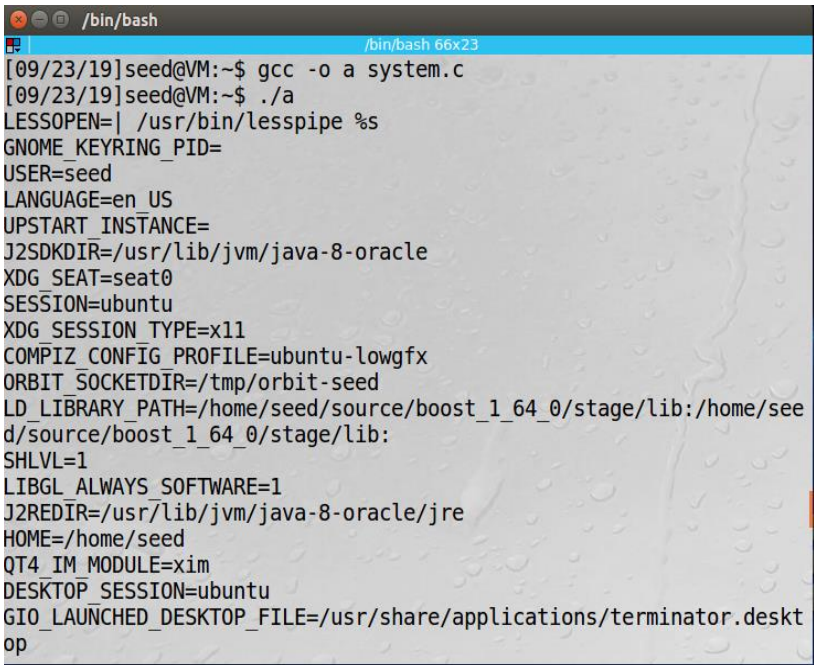
Task-3 Environment Variables and execve()



```
[09/23/19]seed@VM:~$ gedit execve.c
[09/23/19]seed@VM:~$ gcc -o a execve.c
execve.c: In function ‘main’:
execve.c:9:1: warning: implicit declaration of function ‘execve’ [
-Wimplicit-function-declaration]
 execve("/usr/bin/env", argv, NULL);
 ^
[09/23/19]seed@VM:~$ ./a
[09/23/19]seed@VM:~$ gedit execve.c
[09/23/19]seed@VM:~$ gcc -o a execve.c
execve.c: In function ‘main’:
execve.c:9:1: warning: implicit declaration of function ‘execve’ [
-Wimplicit-function-declaration]
 execve("/usr/bin/env", argv, environ);
 ^
[09/23/19]seed@VM:~$ ./a
XDG_VTNR=7
ORBIT_SOCKETDIR=/tmp/orbit-seed
XDG_SESSION_ID=c1
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed
IBUS_DISABLE_SNOOPER=1
TERMINATOR_UUID=urn:uuid:b5164460-192e-4d16-9794-439cec6a54f4
CLUTTER_IM_MODULE=xim
```

Did: I Created a program execve.c to check if the environment variables are automatically inherited by new program. Firstly, I set the environment variables to NULL in the execve() command in the code.
Saw: I observed that, as the shell is returned, the output of this program was nothing. Later, after changing the execve() to environment variables and re-running the program, I observed that the output prints the environment variables.
Learned: I learned that the first parameter of execve() is the command name, the second parameter is the array of arguments, and the third parameter consists of environment variables. Therefore, when the execve() command is set to NULL and executed, it is stored in the arguments of execve() command and sets environment variables. So, the **environ variable generally stores the environment variables by default in the memory as no specific value is assigned anytime and then the calling process is inherited for the environment variables.

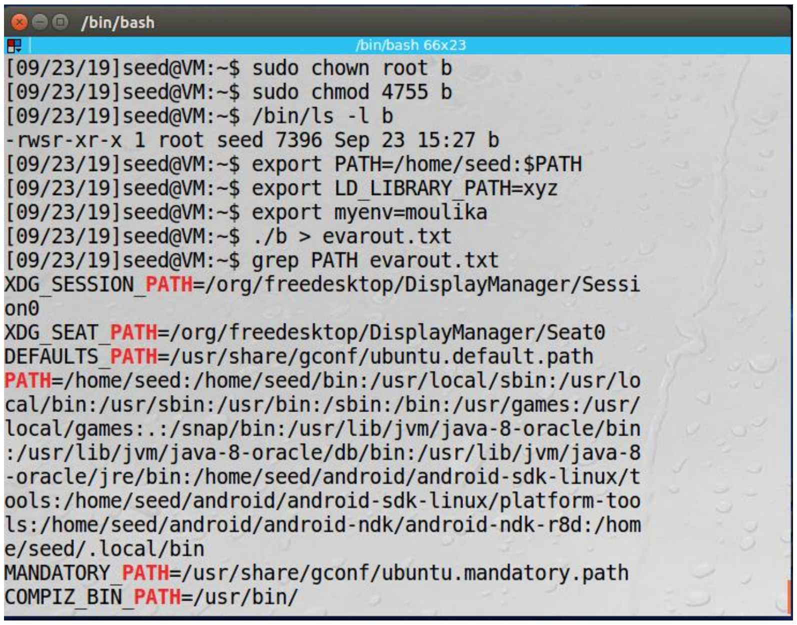
Task-4: Environment Variables and system()



```
/bin/bash
/bin/bash 66x23
[09/23/19]seed@VM:~$ gcc -o a system.c
[09/23/19]seed@VM:~$ ./a
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
ORBIT_SOCKETDIR=/tmp/orbit-seed
LD_LIBRARY_PATH=/home/seed/source/boost_1_64_0/stage/lib:/home/see
d/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
DESKTOP_SESSION=ubuntu
GIO_LAUNCHED_DESKTOP_FILE=/usr/share/applications/terminator.deskt
op
```

<u>Did:</u> I complied and ran the program which used the function() for the execution.
<u>Saw:</u> I observed that, first the shell calls the executable program, then the excve() command. The PID, LD_LIBRARY_PATH, Directories are listed along with the desktop sessions in the output.
<u>Learned:</u> The shell is called and then the shell executes the function() command. I learned that there is an indirect execution of the process. When the excve() command is called into the shell, the environment variables are passed to the shell, then to the excve command.

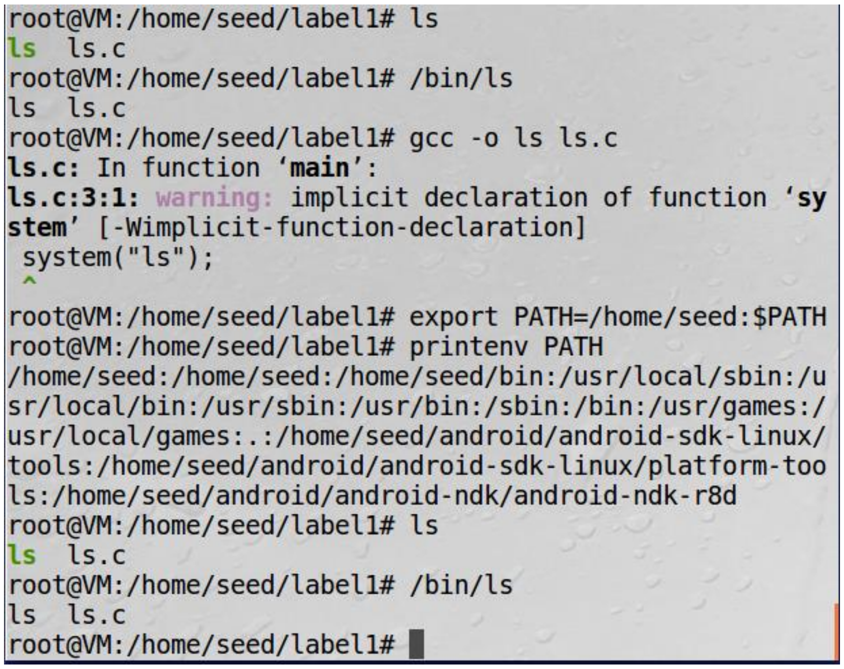
Task-5: Environment Variable and Set-UID Programs.



```
/bin/bash
/bin/bash 66x23
[09/23/19]seed@VM:~$ sudo chown root b
[09/23/19]seed@VM:~$ sudo chmod 4755 b
[09/23/19]seed@VM:~$ /bin/ls -l b
-rwsr-xr-x 1 root seed 7396 Sep 23 15:27 b
[09/23/19]seed@VM:~$ export PATH=/home/seed:$PATH
[09/23/19]seed@VM:~$ export LD_LIBRARY_PATH=xyz
[09/23/19]seed@VM:~$ export myenv=moulika
[09/23/19]seed@VM:~$ ./b > evarout.txt
[09/23/19]seed@VM:~$ grep PATH evarout.txt
XDG_SESSION_PATH=/org/freedesktop/DisplayManager/Sessi
on0
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/lo
cal/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/t
ools:/home/seed/android/android-sdk-linux/platform-too
ls:/home/seed/android/android-ndk/android-ndk-r8d:/hom
e/seed/.local/bin
MANDATORY_PATH=/usr/share/gconf/ubuntu.mandatory.path
COMPIZ_BIN_PATH=/usr/bin/
```

Did: I complied the given program and changed the user ownership to root using “sudo chown root” and made the program a set-UID program using “sudo chmod 4755”. I set environment variable for the “path” i.e; xyz and a random word for “myenv” using export command and run the program.
Saw: I observed the change of ownership using ‘ls -l’ command. When the program is executed after setting the environment variables, I observed the list of PATHS in the outputs where the program searches for the environment variables in each path. The myenv and environment variables PATH is inherited into the set UID program but the LD_LIBRARY_PATH is not inherited.
Learned: As explained in the class, the LD_LIBRARY_PATH is a path from which shared libraries are accessed. It protects against the changing the path i.e; by adding a ‘.’ Infront of the path maybe and malicious files being placed into shared libraries. There might be a possibility of a pre-defined path which cannot be changed for set UID programs. Because of this security feature, the LD_LIBRARY_PATH cannot be inherited into the set UID program.

Task-6: The PATH Environment Variable and set-UID Programs:

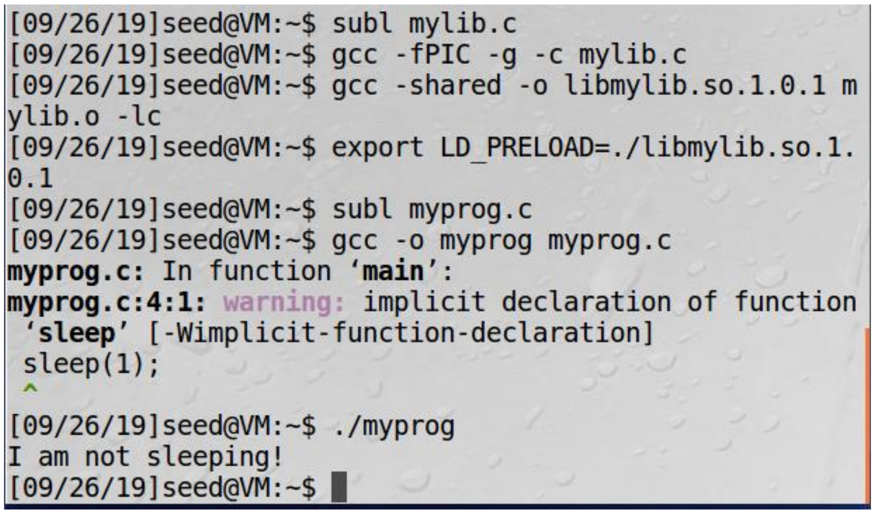


```
root@VM:/home/seed/label1# ls
ls  ls.c
root@VM:/home/seed/label1# /bin/ls
ls  ls.c
root@VM:/home/seed/label1# gcc -o ls ls.c
ls.c: In function ‘main’:
ls.c:3:1: warning: implicit declaration of function ‘sy
stem’ [-Wimplicit-function-declaration]
 system("ls");
 ^

root@VM:/home/seed/label1# export PATH=/home/seed:$PATH
root@VM:/home/seed/label1# printenv PATH
/home/seed:/home/seed:/home/seed/bin:/usr/local/sbin:/u
sr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/
usr/local/games:.:/home/seed/android/android-sdk-linux/
tools:/home/seed/android/android-sdk-linux/platform-too
ls:/home/seed/android/android-ndk/android-ndk-r8d
root@VM:/home/seed/label1# ls
ls  ls.c
root@VM:/home/seed/label1# /bin/ls
ls  ls.c
root@VM:/home/seed/label1# 
```

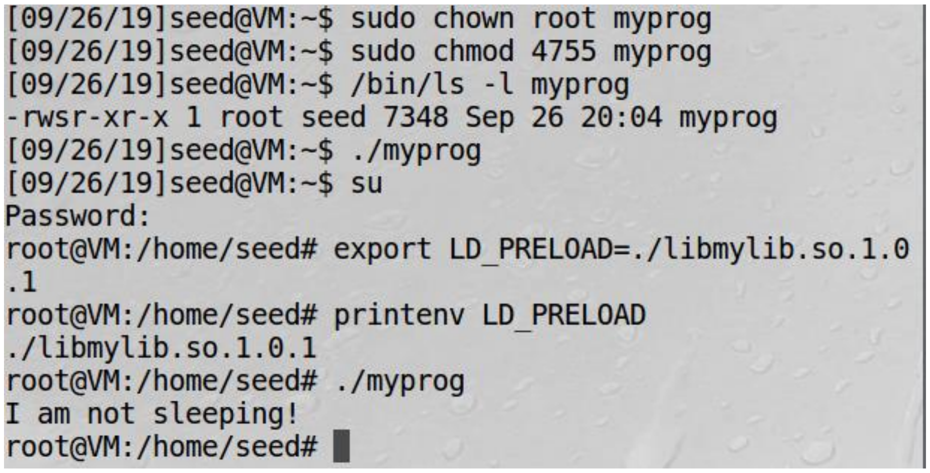
Did: I complied and ran a program with the filename ‘ls’. The code in the file basically does the same functionality as the ‘ls’ command. I changed the ownership of the file to root using “sudo chown root” and set UID privileges using “sudo chmod 4755” and executed it.
Saw: I observed that the /bin/ls and the ls command do the same functionality and the same number of files. If the code written is in the current directory and if we run the code, then the path first looks for the command ‘ls’ in the current directory and runs it instead of the ls file located in the shell command.
Learned: I learned that we can create our own command code files in our chosen directory and use it as a command. It’s just that the path we will be using will be changed but the functionality is the same. The observation can explain us that, someone can alter the path and insert the malicious files with root privileges if the PATH variable is altered.

Task-7: The LD_PRELOAD Environment Variables and Set- UID Programs
Step-1:



```
[09/26/19]seed@VM:~$ subl mylib.c
[09/26/19]seed@VM:~$ gcc -fPIC -g -c mylib.c
[09/26/19]seed@VM:~$ gcc -shared -o libmylib.so.1.0.1 m
ylib.o -lc
[09/26/19]seed@VM:~$ export LD_PRELOAD=./libmylib.so.1.
0.1
[09/26/19]seed@VM:~$ subl myprog.c
[09/26/19]seed@VM:~$ gcc -o myprog myprog.c
myprog.c: In function 'main':
myprog.c:4:1: warning: implicit declaration of function
 'sleep' [-Wimplicit-function-declaration]
 sleep(1);
 ^
[09/26/19]seed@VM:~$ ./myprog
I am not sleeping!
[09/26/19]seed@VM:~$ 
```

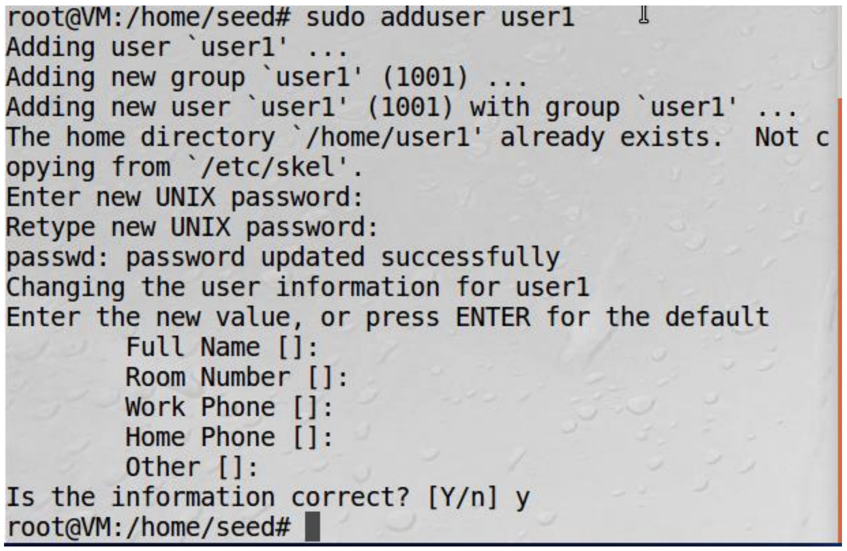
<u>Did:</u> I created a program mylib.c, compiled and made it a dynamic link library. The LD_PRELOAD environment variable is set using the export command to record a route for the recently created DLL. Later, I created a myprog.c program, compiled and ran it.
<u>Saw:</u> When we execute myprog.c program, I observed that the output is "I'm not sleeping".
<u>Learned:</u> I learned that the myprog.c program calls the created mylib DLL instead of the lib.c DLL.
Step-2:



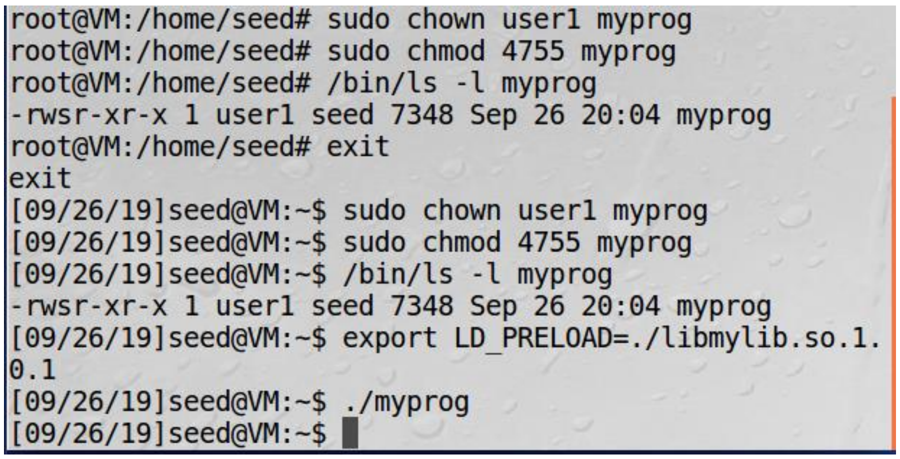
```
[09/26/19]seed@VM:~$ sudo chown root myprog
[09/26/19]seed@VM:~$ sudo chmod 4755 myprog
[09/26/19]seed@VM:~$ /bin/ls -l myprog
-rwsr-xr-x 1 root seed 7348 Sep 26 20:04 myprog
[09/26/19]seed@VM:~$ ./myprog
[09/26/19]seed@VM:~$ su
Password:
root@VM:/home/seed# export LD_PRELOAD=./libmylib.so.1.0
.1
root@VM:/home/seed# printenv LD_PRELOAD
./libmylib.so.1.0.1
root@VM:/home/seed# ./myprog
I am not sleeping!
root@VM:/home/seed# 
```

<u>Did:</u> I changed the ownership of the file using "chown root" command and the privileges to the root by using "chmod 4755". Executed the myprog.c program.
<u>Saw:</u> When myprog.c program is executed, I observed there is no output as it is shown in the previous screenshot for the seed user.
<u>Learned:</u> I learned that I need to set the LD_PRELOAD environment variable pointing to the DLL. I set everything and learned the output changed to "I'm not sleeping!" after executing the myprog program again which typically means that the program calls the mylib DLL we created previously.

Step-2 and Step-3: Adding user1



```
root@VM:/home/seed# sudo adduser user1
Adding user `user1' ...
Adding new group `user1' (1001) ...
Adding new user `user1' (1001) with group `user1' ...
The home directory `/home/user1' already exists.  Not c
opying 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
root@VM:/home/seed# 
```



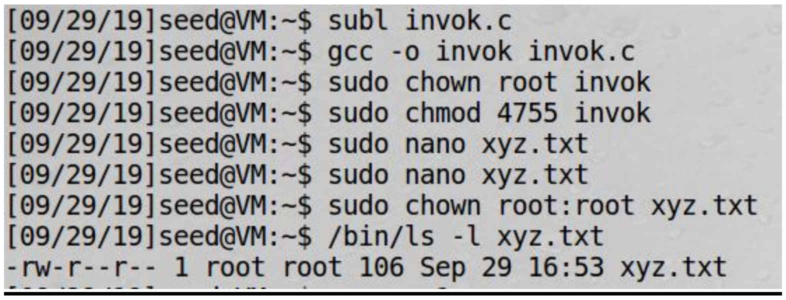
```
root@VM:/home/seed# sudo chown user1 myprog
root@VM:/home/seed# sudo chmod 4755 myprog
root@VM:/home/seed# /bin/ls -l myprog
-rwsr-xr-x 1 user1 seed 7348 Sep 26 20:04 myprog
root@VM:/home/seed# exit
exit
[09/26/19]seed@VM:~$ sudo chown user1 myprog
[09/26/19]seed@VM:~$ sudo chmod 4755 myprog
[09/26/19]seed@VM:~$ /bin/ls -l myprog
-rwsr-xr-x 1 user1 seed 7348 Sep 26 20:04 myprog
[09/26/19]seed@VM:~$ export LD_PRELOAD=./libmylib.so.1.
0.1
[09/26/19]seed@VM:~$ ./myprog
[09/26/19]seed@VM:~$ 
```

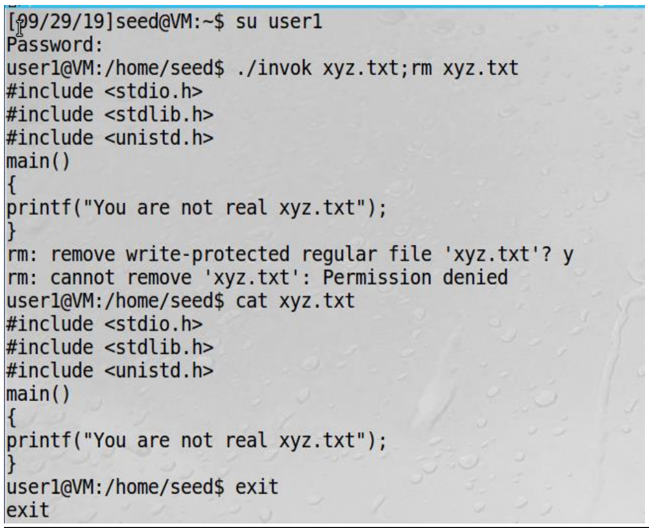
Did: I created new user user1. I made the myprog program a set UID program owned by user1 and then obtained the LD-PRELOAD environment variable pointing to the DLL created previously.
Saw: After changing the ownership of the file to user1, I observed that the LD_PRELOAD environment variable is ignored if a set UID program tries to access it.

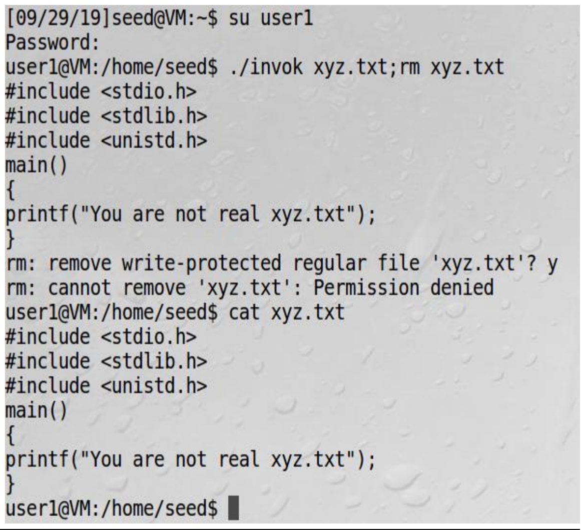
a. In the first scenario, myprog is a regular program run by general user. Hence, LD_PRELOADER is not ignored and the malicious file created by us is accessible.
b. In the second scenario, my prog is a set UID root program and run by normal user. Hence, LD_PRELOADER is ignored and the malicious file created by us is not accessed but the default library file is accessed.
c. In the third scenario, myprog is a set UID root program. Here, an inspection for fake UID and real UID is done and as both has the root privileges, the DLL file is executed.
d. In the fourth Scenario, myprog is a set UID user program which is run by another user. Therefore, LD_PRELOAD is ignored again as it is a set UID program for the user seed but not user1.

Learned: The above situation can be considered as the security mechanism opted by UNIX where UNIX does allow user1 to access the set UID program owned by another user i.e; seed.

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

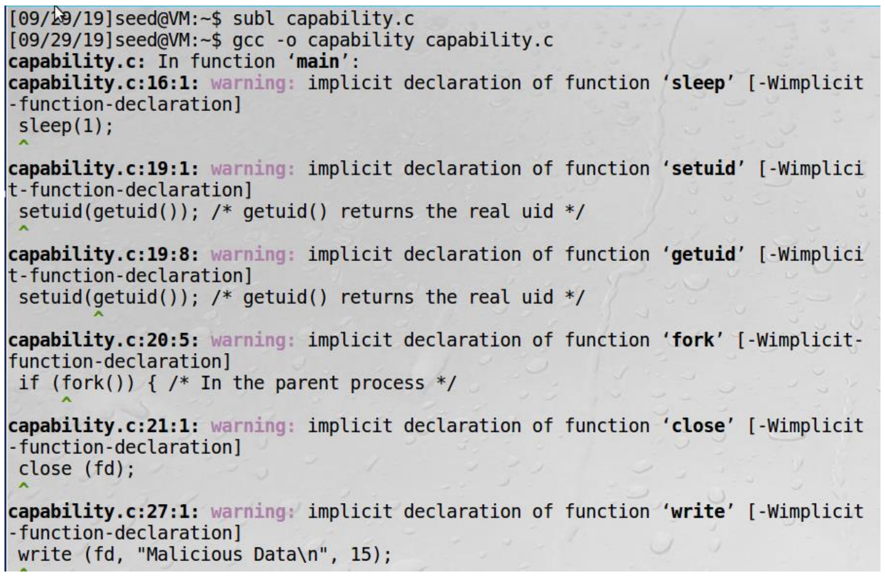

```
[09/29/19]seed@VM:~$ subl invok.c
[09/29/19]seed@VM:~$ gcc -o invok invok.c
[09/29/19]seed@VM:~$ sudo chown root invok
[09/29/19]seed@VM:~$ sudo chmod 4755 invok
[09/29/19]seed@VM:~$ sudo nano xyz.txt
[09/29/19]seed@VM:~$ sudo nano xyz.txt
[09/29/19]seed@VM:~$ sudo chown root:root xyz.txt
[09/29/19]seed@VM:~$ /bin/ls -l xyz.txt
-rw-r--r-- 1 root root 106 Sep 29 16:53 xyz.txt
```


```
[09/29/19]seed@VM:~$ su user1
Password:
user1@VM:/home/seed$ ./invok xyz.txt;rm xyz.txt
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
main()
{
printf("You are not real xyz.txt");
}
rm: remove write-protected regular file 'xyz.txt'? y
rm: cannot remove 'xyz.txt': Permission denied
user1@VM:/home/seed$ cat xyz.txt
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
main()
{
printf("You are not real xyz.txt");
}
user1@VM:/home/seed$ exit
exit
```

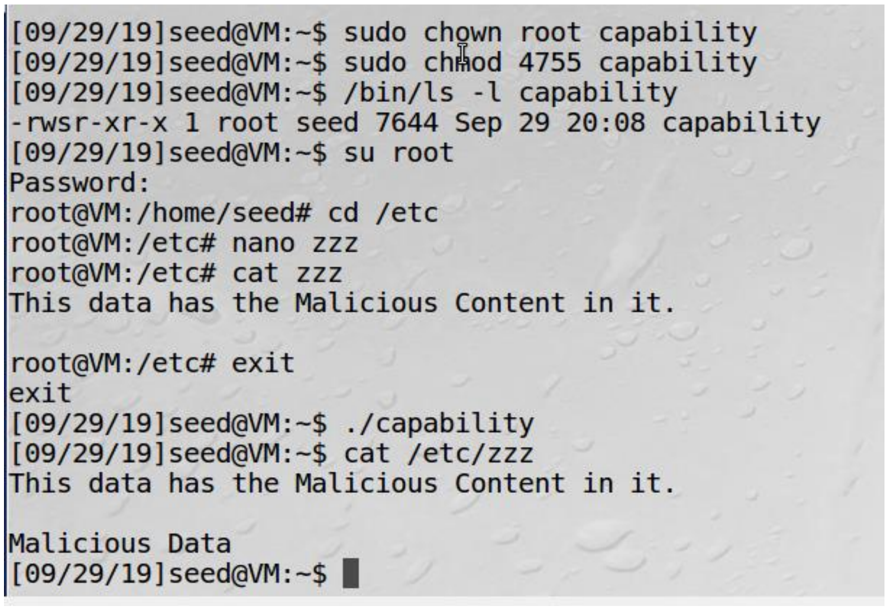

```
[09/29/19]seed@VM:~$ su user1
Password:
user1@VM:/home/seed$ ./invok xyz.txt;rm xyz.txt
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
main()
{
printf("You are not real xyz.txt");
}
rm: remove write-protected regular file 'xyz.txt'? y
rm: cannot remove 'xyz.txt': Permission denied
user1@VM:/home/seed$ cat xyz.txt
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
main()
{
printf("You are not real xyz.txt");
}
user1@VM:/home/seed$ 
```

Did: I created a program with the system() command in code and made it a root owned set - UID program. I created a random .txt file i.e; xyz.txt which is root owned and does not have any written privileges to other users. I logged into the user1 account to execute the invok.c file including the commands for executing the contents of xyz.txt file and deleting the xyz.txt file. Then, I exited from the user1 and logged into the seed user account and executed the invok.c file with the execve() code and repeated the above steps.
Saw: When I executed the "./invok xyz.txt;rm xyz.txt" command, I observed that, I was asked if I want to remove written write-protected file xyz.txt for rm command. I typed "yes" and it said permission denied. I then typed "cat xyz.txt", it executes the contents in the file.
Learned: When the system() code is executed in the invok.c file, it calls the shell and executes the code. I learned that, if the program is set-UID program the user will have temporary root privileges and can remove any file with the root privileges. If we replace the execve() command in the place of system() command in the invok.c code, the execve() command replaces the program with the called program and passes the argument exactly as specified. I also learned that, anything passed after ";" is considered as the new command and root prvileges are lost. So, when the "rm" command is executed using user1 privileges, it cannot delete the file.

Task-9: Capability Leaking



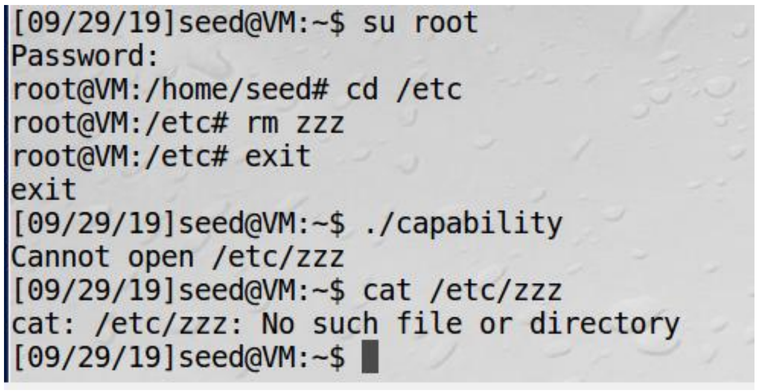
```
[09/29/19]seed@VM:~$ subl capability.c
[09/29/19]seed@VM:~$ gcc -o capability capability.c
capability.c: In function ‘main’:
capability.c:16:1: warning: implicit declaration of function ‘sleep’ [-Wimplicit
-function-declaration]
 sleep(1);
 ^
capability.c:19:1: warning: implicit declaration of function ‘setuid’ [-Wimplici
t-function-declaration]
 setuid(getuid()); /* getuid() returns the real uid */
 ^
capability.c:19:8: warning: implicit declaration of function ‘getuid’ [-Wimplici
t-function-declaration]
 setuid(getuid()); /* getuid() returns the real uid */
        ^
capability.c:20:5: warning: implicit declaration of function ‘fork’ [-Wimplicit-
function-declaration]
 if (fork()) { /* In the parent process */
     ^
capability.c:21:1: warning: implicit declaration of function ‘close’ [-Wimplicit
-function-declaration]
 close (fd);
 ^
capability.c:27:1: warning: implicit declaration of function ‘write’ [-Wimplicit
-function-declaration]
 write (fd, "Malicious Data\n", 15);
```



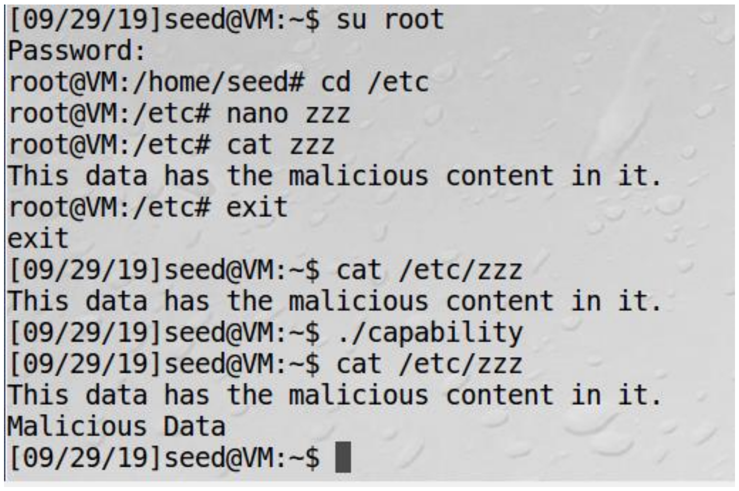
```
[09/29/19]seed@VM:~$ sudo chown root capability
[09/29/19]seed@VM:~$ sudo chmod 4755 capability
[09/29/19]seed@VM:~$ /bin/ls -l capability
-rwsr-xr-x 1 root seed 7644 Sep 29 20:08 capability
[09/29/19]seed@VM:~$ su root
Password:
root@VM:/home/seed# cd /etc
root@VM:/etc# nano zzz
root@VM:/etc# cat zzz
This data has the Malicious Content in it.

root@VM:/etc# exit
exit
[09/29/19]seed@VM:~$ ./capability
[09/29/19]seed@VM:~$ cat /etc/zzz
This data has the Malicious Content in it.

Malicious Data
[09/29/19]seed@VM:~$
```



```
[09/29/19]seed@VM:~$ su root
Password:
root@VM:/home/seed# cd /etc
root@VM:/etc# rm zzz
root@VM:/etc# exit
exit
[09/29/19]seed@VM:~$ ./capability
Cannot open /etc/zzz
[09/29/19]seed@VM:~$ cat /etc/zzz
cat: /etc/zzz: No such file or directory
[09/29/19]seed@VM:~$
```



```
[09/29/19]seed@VM:~$ su root
Password:
root@VM:/home/seed# cd /etc
root@VM:/etc# nano zzz
root@VM:/etc# cat zzz
This data has the malicious content in it.
root@VM:/etc# exit
exit
[09/29/19]seed@VM:~$ cat /etc/zzz
This data has the malicious content in it.
[09/29/19]seed@VM:~$ ./capability
[09/29/19]seed@VM:~$ cat /etc/zzz
This data has the malicious content in it.
Malicious Data
[09/29/19]seed@VM:~$ 
```

Did: I logged into the seed account and created a program capability.c, made it a root owned set-UID program. Then, I logged into the root account and created a "zzz" file in /etc directory which has the text "This data has the malicious content in it". I executed the zzz file using "cat zzz" and exited from root account. Later, I logged into the seed account again, executed the capability file with "./capability" command and executed the "cat /etc/zzz" command.
Saw: Whenever I executed the capability file with "./capability" command and read the contents in zzz file using "cat /etc/zzz" command, I found that, the file /etc/zzz is modified by appending the contents of the child process of capability.c program in the zzz file.
Learned: I learned that, whenever the fork is called, the child inherits the copies of the parent's open file descriptors. The child's file descriptor is same the parent's file descriptor. So, whatever privileges the parent gains, the same applies to the child too and the child can access /etc/zzz file.

This allows some unprivileged user to insert some malicious content into the root privileged files. In order to avoid such attacks, we must close the file descriptors before calling the fork.

Additional findings: I also found that the without executing the ./capability file if we execute the contents of the /etc/zzz file, it just displays the zzz file contents without inheriting the child process of the .capability.c file. However, an attacker will always execute the malicious file he created and then checks if the root file has inherited his malicious content. The execution may be infinite times. If an attacker executes ./capability account infinite times, the /etc/zzz file in the root account inherits the child process contents of capability.c file infinite times and stores it in zzz file such as:

This data has the malicious content in it.
Malicious data

This data has the malicious content in it.
Malicious data
Malicious data
.
.
.

Infinite times.