

## Hw Assignment 1 - OS Security

Purpose

The purpose of this assignment is to gain a better understanding of how privilege escalation and using environment variables can change the behavior of programs and systems. This lab helps the student get more familiar with manipulating program privileges and meaningful environment variables to simulate malicious attacks and understand how the OS handles these attacks. Additionally, this assignment allows one to practice the concepts learned in class.

Tasks

The lab was set up into 9 different tasks, starting with creating and manipulating environment variables and ends with seeing how changing a program's privileges affect child processes.

**Task 2.1**

The first task was to get familiar with how to see, set, and unset environment variables. The following picture shows how you can access the value of the environment variable *PWD*. Another way to see the environment variable is *echo \$PWD*.

(img 2.1.1)



```

[02/10/21] seed@VM: ~$ env | grep PWD
PWD=/home/seed
[02/10/21] seed@VM: ~$

```

Below is an example of creating a variable *MY\_VAR* and giving it a variable (img 2.1.2). Export is then called to display all environment variables. From the (img 2.1.3), one can see the created variable *MY\_VAR* in the list. By calling *unset*, the environment variable is removed from the list of environment variables until the user wants to set it again

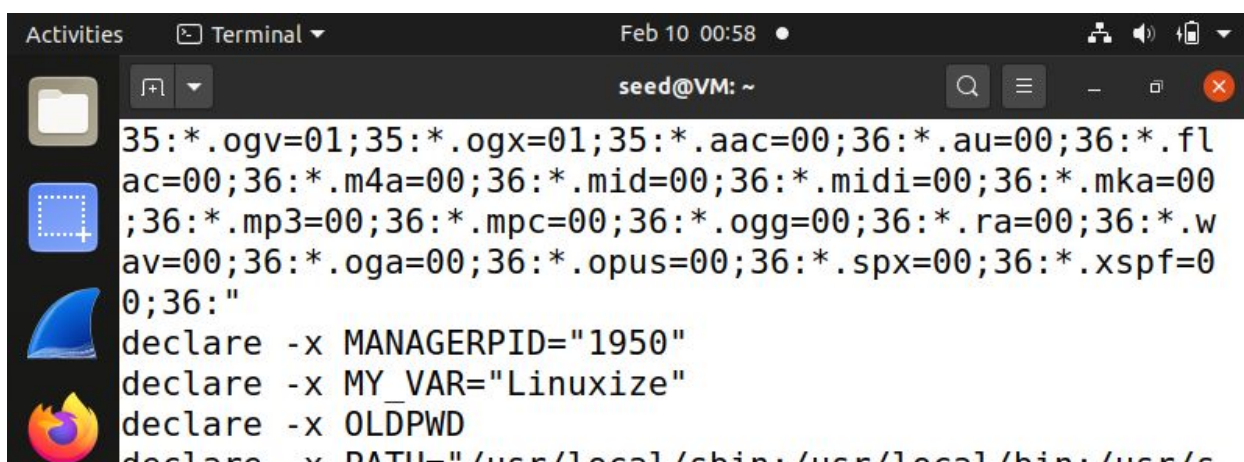
(img 2.1.2)

```

[02/10/21] seed@VM: ~$ export MY_VAR='Linuxize'
[02/10/21] seed@VM: ~$ export

```

(img 2.1.3)



```

35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.fl
ac=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:*.w
av=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=0
0;36:"
declare -x MANAGERPID="1950"
declare -x MY_VAR="Linuxize"
declare -x OLDPWD
declare -x PATH="/usr/local/sbin:/usr/local/bin:/usr/l

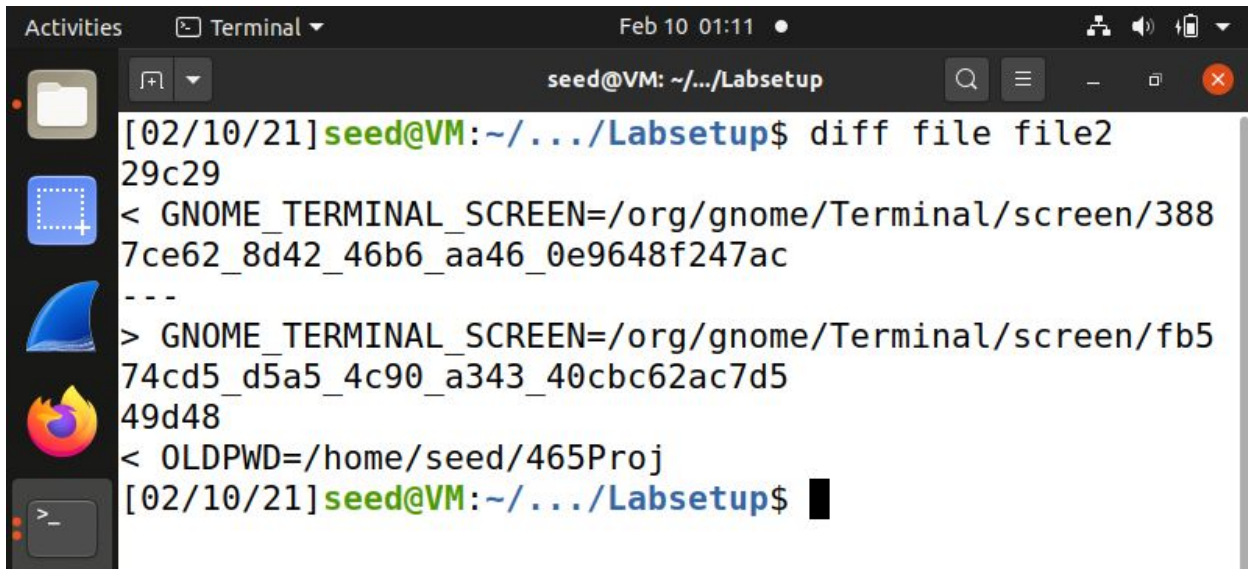
```

## Hw Assignment 1 - OS Security

**Task 2.2**

This task consisted of passing variables between child and parent processes. These two outputs were placed into file and file 2, respectively. In order to tell if everything is copied from the parent to the child process, the *diff* command was used on the files (**img 2.2.1**). Looking at both files, I noticed that the environment variables were passed to the child process. Looking at the results I got from **img 2.2.1**, It appears that the child process also keeps track of the old directory. Child processes run almost as an extension of a parent process, so it makes sense that a child process keeps track of the old working directory and inherits environment variables with the values the parent process had. This is likely done so the child process can work more efficiently as a child process to the original parent process.

(img 2.2.1)



```

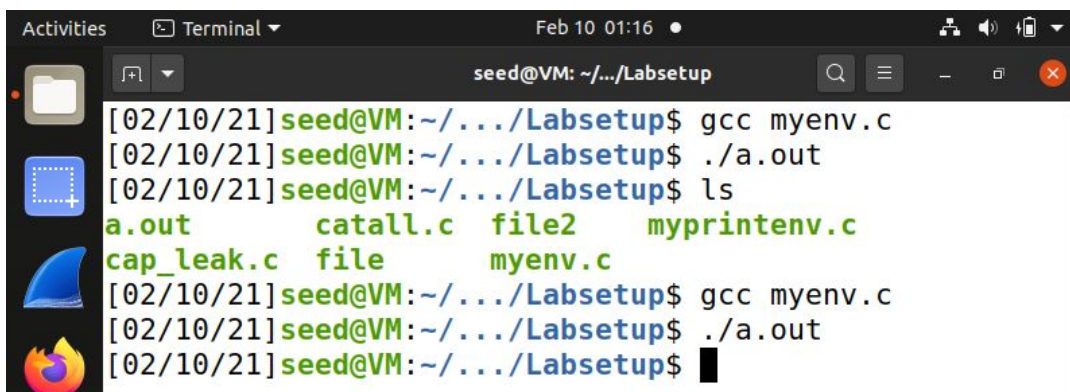
[02/10/21] seed@VM: ~/.../Labsetup$ diff file file2
29c29
< GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/388
7ce62_8d42_46b6_aa46_0e9648f247ac
---
> GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/fb5
74cd5_d5a5_4c90_a343_40cbc62ac7d5
49d48
< OLDPWD=/home/seed/465Proj
[02/10/21] seed@VM: ~/.../Labsetup$

```

**Task 2.3**

For this task we looked at how the function *execve()* runs programs in the current process. The program *myenv.c* is meant to print out all the environment variables in a program. After running the code with nothing changed, **img 2.3.1** is produced. Since nothing is printed out, I concluded that the new program does not automatically inherit the current process's environment variables. This is another reason why *execve()* is different from *fork()*.

(img 2.3.1)



```

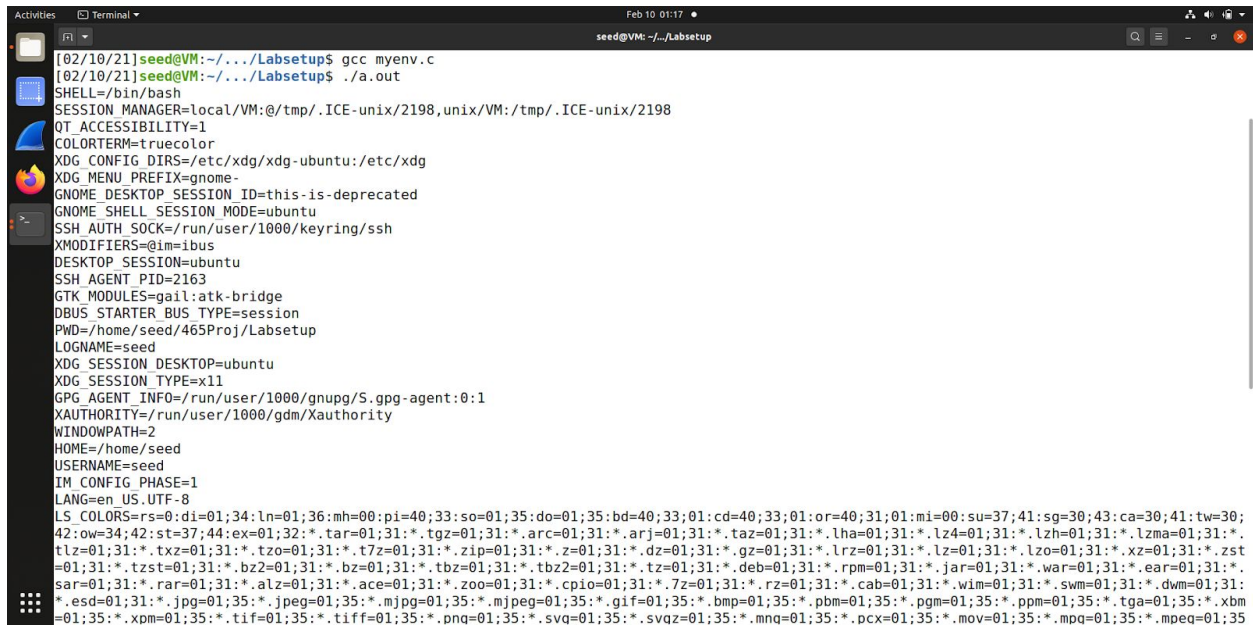
[02/10/21] seed@VM: ~/.../Labsetup$ gcc myenv.c
[02/10/21] seed@VM: ~/.../Labsetup$ ./a.out
[02/10/21] seed@VM: ~/.../Labsetup$ ls
a.out      catall.c  file2     myprintenv.c
cap_leak.c file      myenv.c
[02/10/21] seed@VM: ~/.../Labsetup$ gcc myenv.c
[02/10/21] seed@VM: ~/.../Labsetup$ ./a.out
[02/10/21] seed@VM: ~/.../Labsetup$

```

## Hw Assignment 1 - OS Security

This conclusion is further proved when including the environment variables *environ* produces the output in **img 2.3.2**. By providing environment variables instead of leaving that parameter NULL, led to the output of environment variables for that new program. So I was able to reach the conclusion that if one uses *execve()* environment variables won't be inherited to the new program.

(img 2.3.2)



```

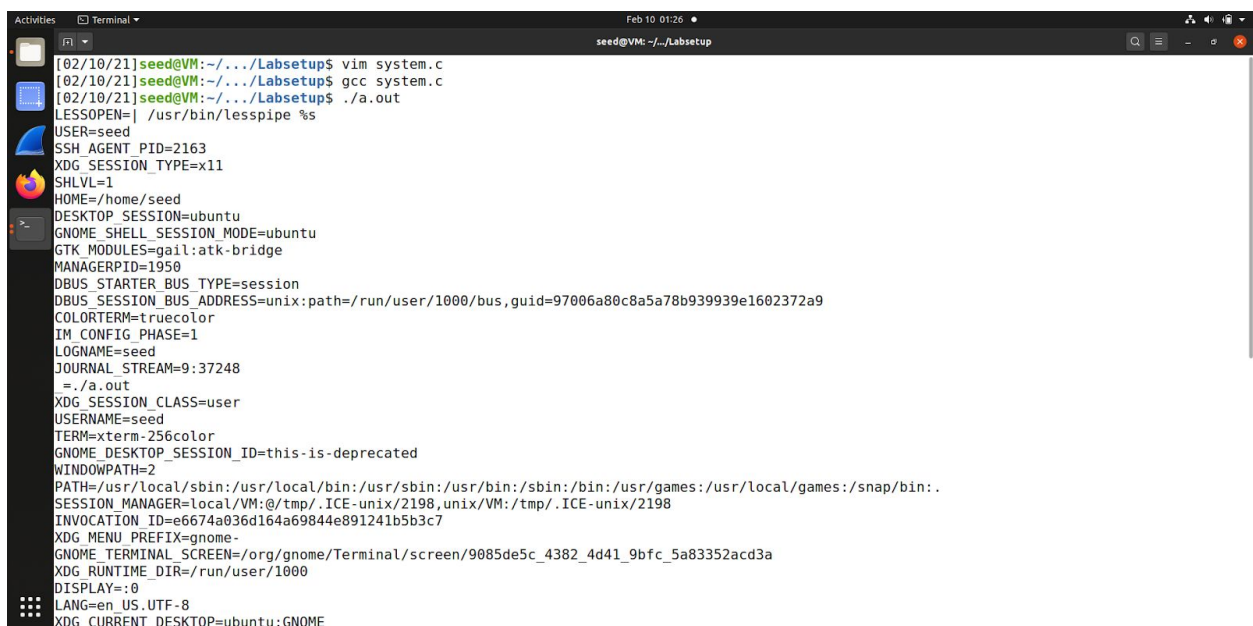
[02/10/21]seed@VM:~/.../Labsetup$ gcc myenv.c
[02/10/21]seed@VM:~/.../Labsetup$ ./a.out
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/2198,unix/VM:/tmp/.ICE-unix/2198
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
XDG_MENU_PREFIX=gnome-
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
GNOME_SHELL_SESSION_MODE=ubuntu
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
XMODIFIERS=@im=ibus
DESKTOP_SESSION=ubuntu
SSH_AGENT_PID=2163
GTK_MODULES=gail:atk-bridge
DBUS_STARTER_BUS_TYPE=session
PWD=/home/seed/465Proj/Labsetup
LOGNAME=seed
XDG_SESSION_DESKTOP=ubuntu
XDG_SESSION_TYPE=x11
GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1
XAUTHORITY=/run/user/1000/gdm/Xauthority
WINDOWPATH=2
HOME=/home/seed
USERNAME=seed
IM_CONFIG_PHASE=1
LANG=en_US.UTF-8
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33:01:cd=40;33:01:or=40;31:01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;
42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lzh=01;31:*.lzm=01;31:*.
tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.zst
=01;31:*.tztst=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.
sar=01;31:*.rar=01;31:*.alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.wim=01;31:*.swm=01;31:*.dwm=01;31:
*.esd=01;31:*.jpg=01;35:*.jpeg=01;35:*.mjpg=01;35:*.mjpeg=01;35:*.gif=01;35:*.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;35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpq=01;35:*.mpeg=01;35

```

## Task 2.4

For task 2.4, I observed how environment variables are affected when a new program calls the function *system()*. After running the program, I saw that the original program passes environment variables to the new program. This is seen by the created output in **img 2.4.1**. This task verified the concept that *system()* calls *exec()* and then calls *execve()* which gets the environment variables array automatically passed to *execve()*.

(img 2.4.1)



```

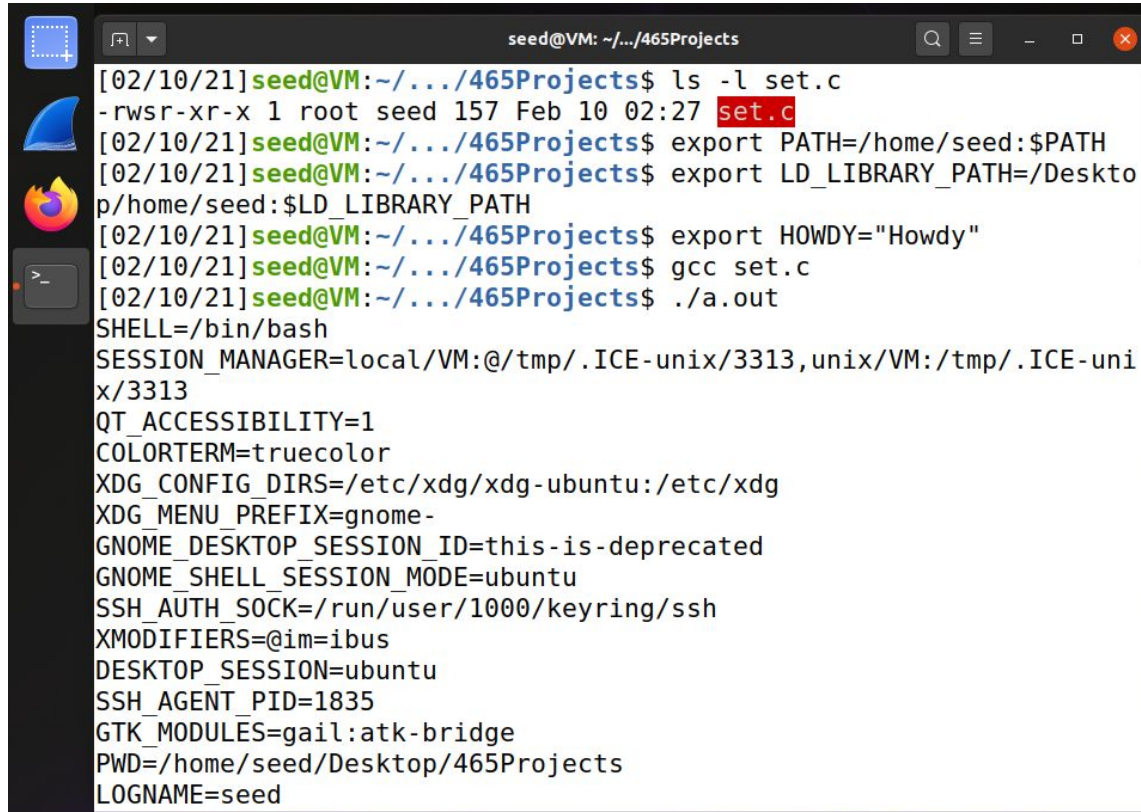
[02/10/21]seed@VM:~/.../Labsetup$ vim system.c
[02/10/21]seed@VM:~/.../Labsetup$ gcc system.c
[02/10/21]seed@VM:~/.../Labsetup$ ./a.out
LESSOPEN=| /usr/bin/lesspipe %s
USER=seed
SSH_AGENT_PID=2163
XDG_SESSION_TYPE=x11
SHLVL=1
HOME=/home/seed
DESKTOP_SESSION=ubuntu
GNOME_SHELL_SESSION_MODE=ubuntu
GTK_MODULES=gail:atk-bridge
MANAGERPID=1950
DBUS_STARTER_BUS_TYPE=session
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus,guid=97006a80c8a5a78b939939e1602372a9
COLORTERM=truecolor
IM_CONFIG_PHASE=1
LOGNAME=seed
JOURNAL_STREAM=9:37248
./a.out
XDG_SESSION_CLASS=user
USERNAME=seed
TERM=xterm-256color
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
WINDOWPATH=2
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/2198,unix/VM:/tmp/.ICE-unix/2198
INVOCATION_ID=e6674a036d164a69844e891241b5b3c7
XDG_MENU_PREFIX=gnome-
GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/9085de5c_4382_4d41_9bfc_5a83352acd3a
XDG_RUNTIME_DIR=/run/user/1000
DISPLAY=:0
LANG=en_US.UTF-8
XDG_CURRENT_DESKTOP=ubuntu:GNOME

```



## Hw Assignment 1 - OS Security

## Task 2.5

This  
task


```

seed@VM: ~/.../465Projects
[02/10/21] seed@VM: ~/.../465Projects$ ls -l set.c
-rwsr-xr-x 1 root seed 157 Feb 10 02:27 set.c
[02/10/21] seed@VM: ~/.../465Projects$ export PATH=/home/seed:$PATH
[02/10/21] seed@VM: ~/.../465Projects$ export LD_LIBRARY_PATH=/Desktop
/home/seed:$LD_LIBRARY_PATH
[02/10/21] seed@VM: ~/.../465Projects$ export HOWDY="Howdy"
[02/10/21] seed@VM: ~/.../465Projects$ gcc set.c
[02/10/21] seed@VM: ~/.../465Projects$ ./a.out
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/3313,unix/VM:/tmp/.ICE-unix/3313
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
XDG_MENU_PREFIX=gnome-
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
GNOME_SHELL_SESSION_MODE=ubuntu
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
XMODIFIERS=@im=ibus
DESKTOP_SESSION=ubuntu
SSH_AGENT_PID=1835
GTK_MODULES=gail:atk-bridge
PWD=/home/seed/Desktop/465Projects
LOGNAME=seed

```

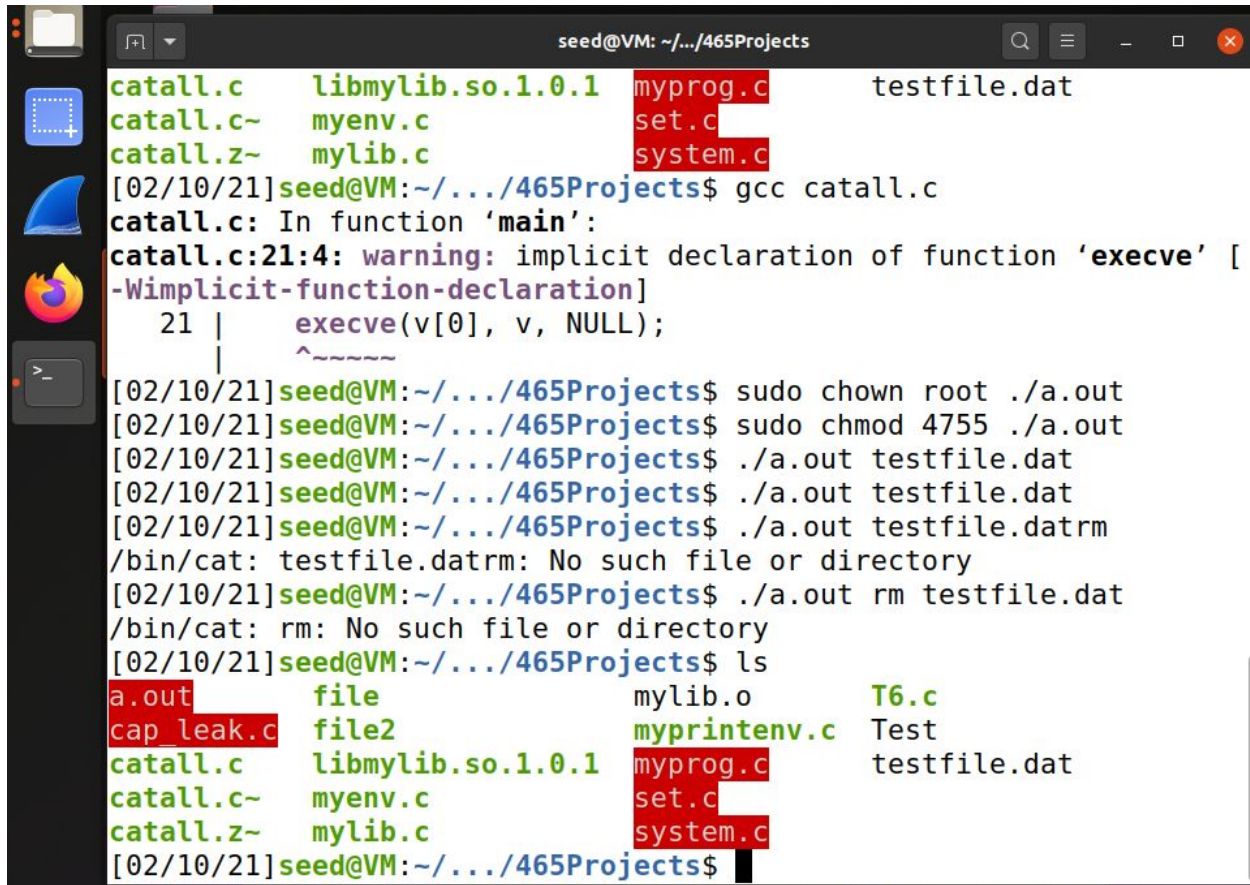
introduces manipulating program privileges. Because of this, the program can't be run in the shared file like the previous tasks. This is a result of the folder being shared with the host computer, which will have root privileges even when it says that the program's privileges is a user. This task and the ones after it will be executed in the virtual machine's copy of the files, located at *465Projects*. This shows that the HOWDY variable doesn't get passed in, but does with the `execl()`.

## Hw Assignment 1 - OS Security

### **Task 2.6**

Task 2.6 shows

## Hw Assignment 1 - OS Security



```

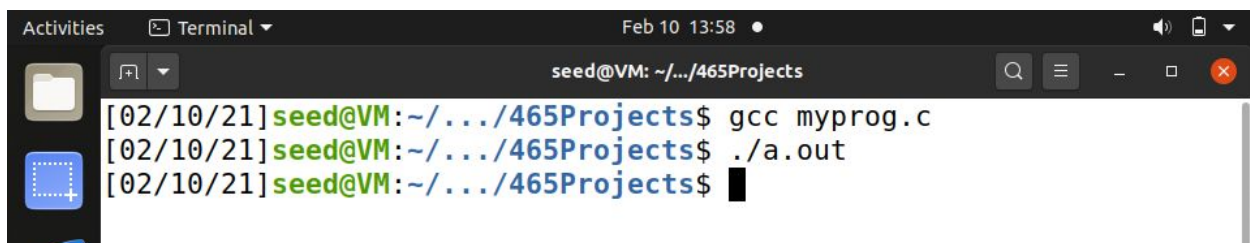
catall.c  libmylib.so.1.0.1  myprog.c  testfile.dat
catall.c~ myenv.c           set.c
catall.z~ mylib.c           system.c
[02/10/21] seed@VM: ~/.../465Projects$ gcc catall.c
catall.c: In function 'main':
catall.c:21:4: warning: implicit declaration of function 'execve' [-Wimplicit-function-declaration]
   21 |     execve(v[0], v, NULL);
       |     ^~~~~~
[02/10/21] seed@VM: ~/.../465Projects$ sudo chown root ./a.out
[02/10/21] seed@VM: ~/.../465Projects$ sudo chmod 4755 ./a.out
[02/10/21] seed@VM: ~/.../465Projects$ ./a.out testfile.dat
[02/10/21] seed@VM: ~/.../465Projects$ ./a.out testfile.dat
[02/10/21] seed@VM: ~/.../465Projects$ ./a.out testfile.datrm
/bin/cat: testfile.datrm: No such file or directory
[02/10/21] seed@VM: ~/.../465Projects$ ./a.out rm testfile.dat
/bin/cat: rm: No such file or directory
[02/10/21] seed@VM: ~/.../465Projects$ ls
a.out      file          mylib.o      T6.c
cap_leak.c file2         myprintenv.c Test
catall.c  libmylib.so.1.0.1  myprog.c    testfile.dat
catall.c~ myenv.c       set.c
catall.z~ mylib.c      system.c
[02/10/21] seed@VM: ~/.../465Projects$

```

**Task 2.7**

For task 2.7 I practiced privilege escalation and determined if and/or how environment variables affect the program's behavior. After setting the environment variable `LD_PRELOAD` and creating the overloaded `sleep()` function, I ran the program under the 4 different scenarios as outlined in the lab. Below are the scenarios and the respective outputs.

(img 2.7.1: Seed user, `LD_PRELOAD` not set)



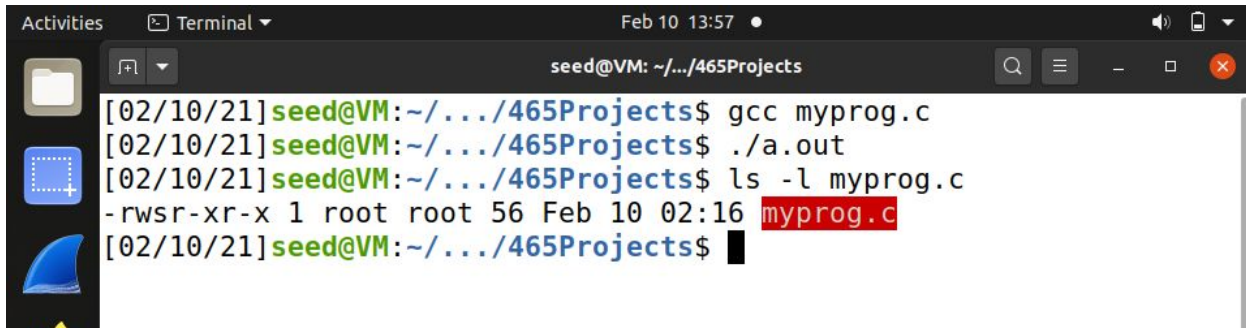
```

[02/10/21] seed@VM: ~/.../465Projects$ gcc myprog.c
[02/10/21] seed@VM: ~/.../465Projects$ ./a.out
[02/10/21] seed@VM: ~/.../465Projects$

```

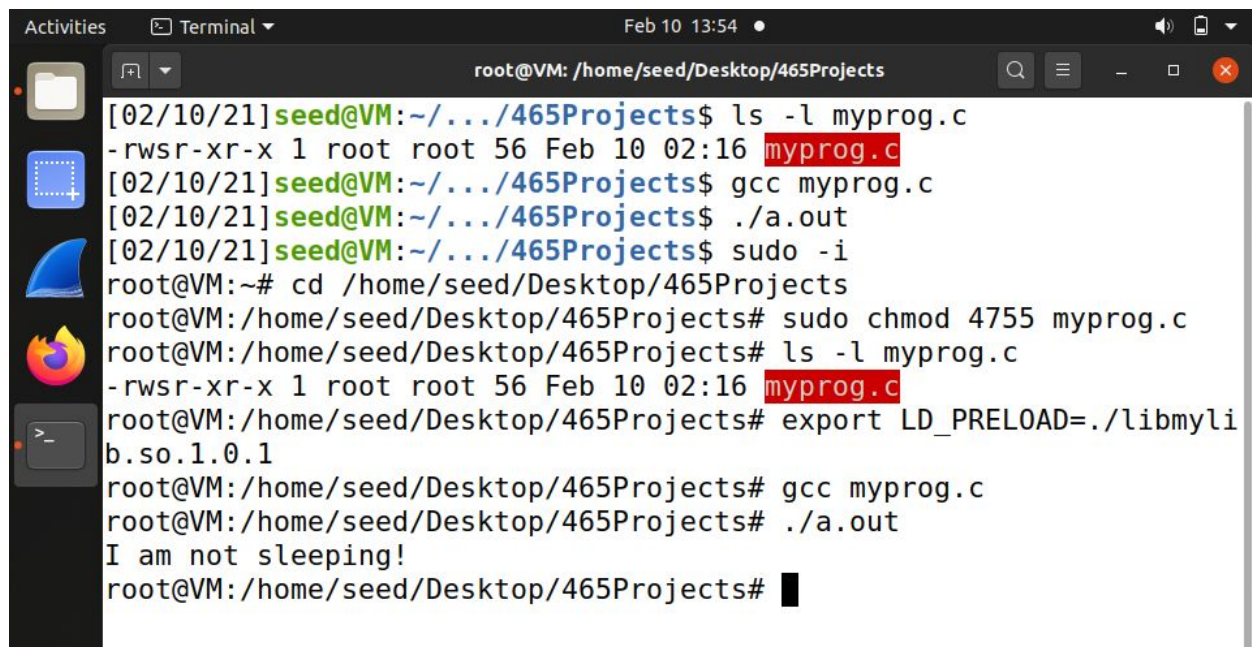
(img 2.7.2: Root owned program, `LD_PRELOAD` not set)

## Hw Assignment 1 - OS Security



```
Activities Terminal Feb 10 13:57
seed@VM: ~/.../465Projects
[02/10/21] seed@VM:~/.../465Projects$ gcc myprog.c
[02/10/21] seed@VM:~/.../465Projects$ ./a.out
[02/10/21] seed@VM:~/.../465Projects$ ls -l myprog.c
-rwsr-xr-x 1 root root 56 Feb 10 02:16 myprog.c
[02/10/21] seed@VM:~/.../465Projects$
```

(img 2.7.3: root owned, LD\_PRELOAD set)

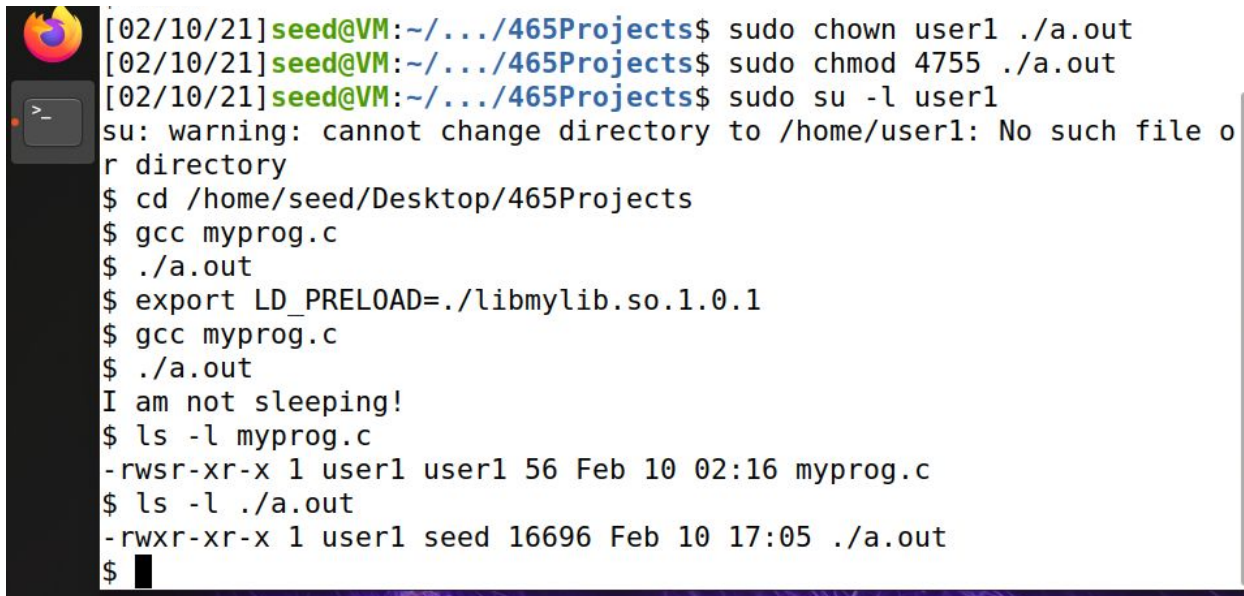


```
Activities Terminal Feb 10 13:54
root@VM: /home/seed/Desktop/465Projects
[02/10/21] seed@VM:~/.../465Projects$ ls -l myprog.c
-rwsr-xr-x 1 root root 56 Feb 10 02:16 myprog.c
[02/10/21] seed@VM:~/.../465Projects$ gcc myprog.c
[02/10/21] seed@VM:~/.../465Projects$ ./a.out
[02/10/21] seed@VM:~/.../465Projects$ sudo -i
root@VM:~# cd /home/seed/Desktop/465Projects
root@VM:/home/seed/Desktop/465Projects# sudo chmod 4755 myprog.c
root@VM:/home/seed/Desktop/465Projects# ls -l myprog.c
-rwsr-xr-x 1 root root 56 Feb 10 02:16 myprog.c
root@VM:/home/seed/Desktop/465Projects# export LD_PRELOAD=./libmyli
b.so.1.0.1
root@VM:/home/seed/Desktop/465Projects# gcc myprog.c
root@VM:/home/seed/Desktop/465Projects# ./a.out
I am not sleeping!
root@VM:/home/seed/Desktop/465Projects#
```

(img 2.7.4: user1 owned, LD\_PRELOAD set)



## Hw Assignment 1 - OS Security



```
[02/10/21] seed@VM:~/.../465Projects$ sudo chown user1 ./a.out
[02/10/21] seed@VM:~/.../465Projects$ sudo chmod 4755 ./a.out
[02/10/21] seed@VM:~/.../465Projects$ sudo su -l user1
su: warning: cannot change directory to /home/user1: No such file or
directory
$ cd /home/seed/Desktop/465Projects
$ gcc myprog.c
$ ./a.out
$ export LD_PRELOAD=./libmylib.so.1.0.1
$ gcc myprog.c
$ ./a.out
I am not sleeping!
$ ls -l myprog.c
-rwsr-xr-x 1 user1 user1 56 Feb 10 02:16 myprog.c
$ ls -l ./a.out
-rwxr-xr-x 1 user1 seed 16696 Feb 10 17:05 ./a.out
$
```

Based on these observations, it shows that if the environment variable *LD\_PRELOADED* isn't set, the program runs as supposed to, without using the overloaded *sleep()* function. This means that *LD\_PRELOADED* isn't inherited by the other process. However, further testing would be needed to find the real reason this is happening. I would perform experiments testing how the program behaves when the environment variable is set and not set. I would do this at different privileges and different users, changing one part at a time to rule out if that change is or isn't the culprit. I would also test the program with other environment variables to see if this happens for all environment variables or just a certain type of environment variable. However, based on the output, it seems that the privileges and user owner doesn't really matter for how the program chooses to override what. The values might be different in step 2 from step 1 because the environment variable wasn't originally set for the normal user, SEED. This could also be because the file is in a shared space, which allowed other users to access that modified *sleep()* program.

### Task 2.8

For task 2.8, we simulated how a privilege escalation with *setuid* can lead to file corruption or unauthorized access to modify files. To simulate this, I created a file called *testfile.dat* (img 2.8.) and made it root owned, while escalating the privileges of *catal.c* to simulate this potential attack. When calling *system()* I was able to remove the *testfile.dat* when I wasn't supposed to. This is because of how it gets called and what gets passed into it. However this attack doesn't work for the *execve()*.



## Hw Assignment 1 - OS Security

(img 2.8.1)

```

seed@VM: ~/.../465Projects
[02/10/21]seed@VM:~/.../465Projects$ ls -l catall.c
-rwsr-xr-x 1 root seed 450 Feb 10 20:04 catall.c
[02/10/21]seed@VM:~/.../465Projects$ gcc catall.c
[02/10/21]seed@VM:~/.../465Projects$ ./catall.c "testfile.dat;rm"
./catall.c: line 5: syntax error near unexpected token `('
./catall.c: line 5: `int main(int argc, char *argv[])'
[02/10/21]seed@VM:~/.../465Projects$ ./catall "testfile.dat;rm"
bash: ./catall: No such file or directory
[02/10/21]seed@VM:~/.../465Projects$ ./a.out "testfile.dat;rm"
/bin/cat: testfile.dat: Permission denied
rm: missing operand
Try 'rm --help' for more information.
[02/10/21]seed@VM:~/.../465Projects$ ./a.out "testfile.dat;rm testfile.dat"
/bin/cat: testfile.dat: Permission denied
rm: remove write-protected regular file 'testfile.dat'? y
[02/10/21]seed@VM:~/.../465Projects$ ls
a.out      catall.z~      myenv.c      myprog.c
cap_leak.c file            mylib.c      set.c
catall.c   file2          mylib.o      system.c
catall.c~  libmylib.so.1.0.1 myprintenv.c T6.c
[02/10/21]seed@VM:~/.../465Projects$

```

(img 2.8.2)

```

seed@VM: ~/.../465Projects
[02/10/21]seed@VM:~/.../465Projects$ sudo chmod 4755 catall.c
[02/10/21]seed@VM:~/.../465Projects$ ls -l catall.c
-rwsr-xr-x 1 root seed 450 Feb 10 20:55 catall.c
[02/10/21]seed@VM:~/.../465Projects$ ls -l testfile.dat
ls: cannot access 'testfile.dat': No such file or directory
[02/10/21]seed@VM:~/.../465Projects$ 'TEXT' > testfile.dat
TEXT: command not found
[02/10/21]seed@VM:~/.../465Projects$ echo 'TEXT' > testfile.dat
[02/10/21]seed@VM:~/.../465Projects$ cat testfile.dat
TEXT
[02/10/21]seed@VM:~/.../465Projects$ sudo chown root testfile.dat
[02/10/21]seed@VM:~/.../465Projects$ sudo chmod 4755 testfile.dat
[02/10/21]seed@VM:~/.../465Projects$ gcc catall.c
catall.c: In function 'main':
catall.c:21:4: warning: implicit declaration of function 'execve' [-Wimplicit-function-declaration]
   21 |     execve(v[0], v, NULL);
      |         ^~~~~
[02/10/21]seed@VM:~/.../465Projects$ ./a.out "testfile.dat; rm testfile.dat"
/bin/cat: 'testfile.dat; rm testfile.dat': No such file or directory
[02/10/21]seed@VM:~/.../465Projects$ ./a.out "testfile.dat;rm testfile.dat"

```

## Hw Assignment 1 - OS Security

**Task 2.9**

After compiling the code, I was able to successfully write to the file and create malicious data. This proves that the child process does inherit the privileges of the parent process if it's not reset before being passed into it.

(img 2.9.1)

```

Activities  Terminal  Feb 10 16:47
seed@VM: ~/.../465Projects

[02/10/21] seed@VM:~/.../465Projects$ vim cap_leak.c
[02/10/21] seed@VM:~/.../465Projects$ sudo touch /etc/zzz
[02/10/21] seed@VM:~/.../465Projects$ ls -l /etc/zzz
-rw-r--r-- 1 root root 0 Feb 10 16:46 /etc/zzz
[02/10/21] seed@VM:~/.../465Projects$ rm /etc/zzz
rm: remove write-protected regular empty file '/etc/zzz'? y
rm: cannot remove '/etc/zzz': Permission denied
[02/10/21] seed@VM:~/.../465Projects$ sudo rm /etc/zzz
[02/10/21] seed@VM:~/.../465Projects$ ls -l /etc/zzz
ls: cannot access '/etc/zzz': No such file or directory
[02/10/21] seed@VM:~/.../465Projects$

```

(img 2.9.2)

```

cap_leak.c:26:2: warning: implicit declaration of function 'close';
did you mean 'pclose'? [-Wimplicit-function-declaration]
 26 |   close(fd);
    |   ^~~~~
    |   pclose
cap_leak.c:29:4: warning: implicit declaration of function 'write';
did you mean 'fwrite'? [-Wimplicit-function-declaration]
 29 |     write(fd,"Malicious Data\n", 15);
    |     ^~~~~
    |     fwrite
[02/10/21] seed@VM:~/.../465Projects$ sudo chown root ./a.out
[02/10/21] seed@VM:~/.../465Projects$ sudo chmod 4755 ./a.out
[02/10/21] seed@VM:~/.../465Projects$ ./a.out
[02/10/21] seed@VM:~/.../465Projects$ ./a.out cat
[02/10/21] seed@VM:~/.../465Projects$ sudo cat zzz
cat: zzz: No such file or directory
[02/10/21] seed@VM:~/.../465Projects$ sudo cat etc/zzz
cat: etc/zzz: No such file or directory
[02/10/21] seed@VM:~/.../465Projects$ sudo touch etc/zzz
touch: cannot touch 'etc/zzz': No such file or directory
[02/10/21] seed@VM:~/.../465Projects$

```

## Hw Assignment 1 - OS Security

