Computer Security

Lab 1 Report

Set UID

Chirag Sachdev

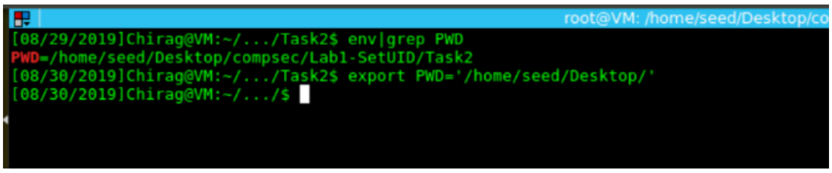
680231131

Task 1:

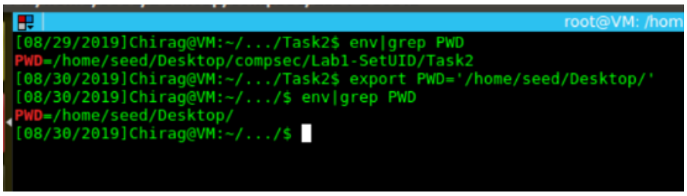
Manipulating Environment Variables

Printing environment variable PWD using:

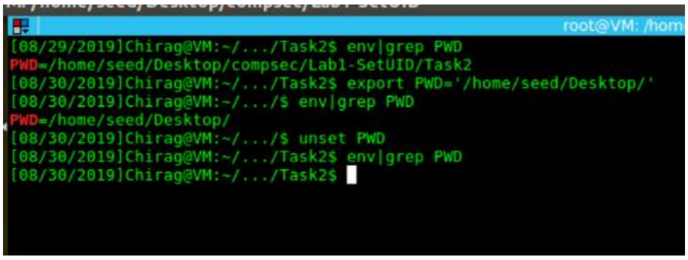
env | grep PWD



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



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



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.

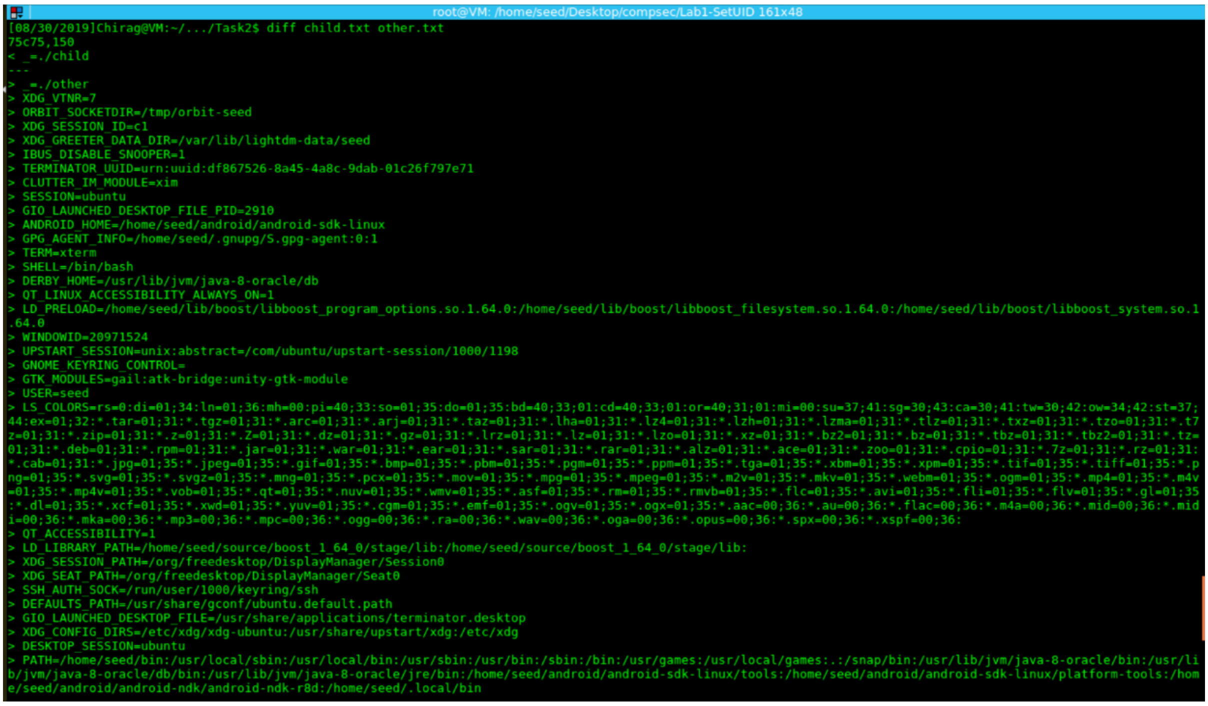
`gcc -o child 1-child.c`

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.



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.

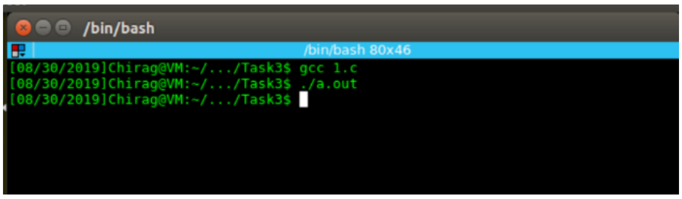
Task 3:

execve() and Environment variables

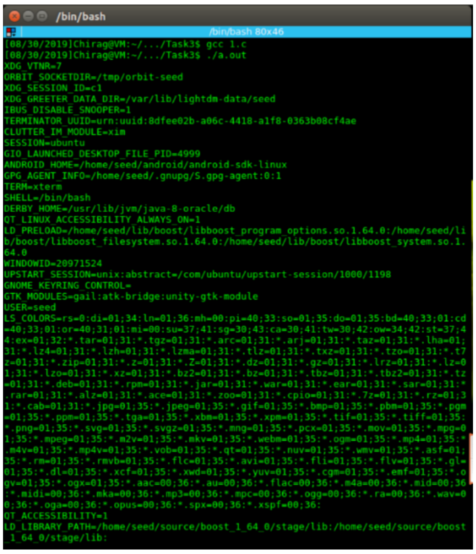
Code:

```c
#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.



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



Task 4:

System() and environment variables

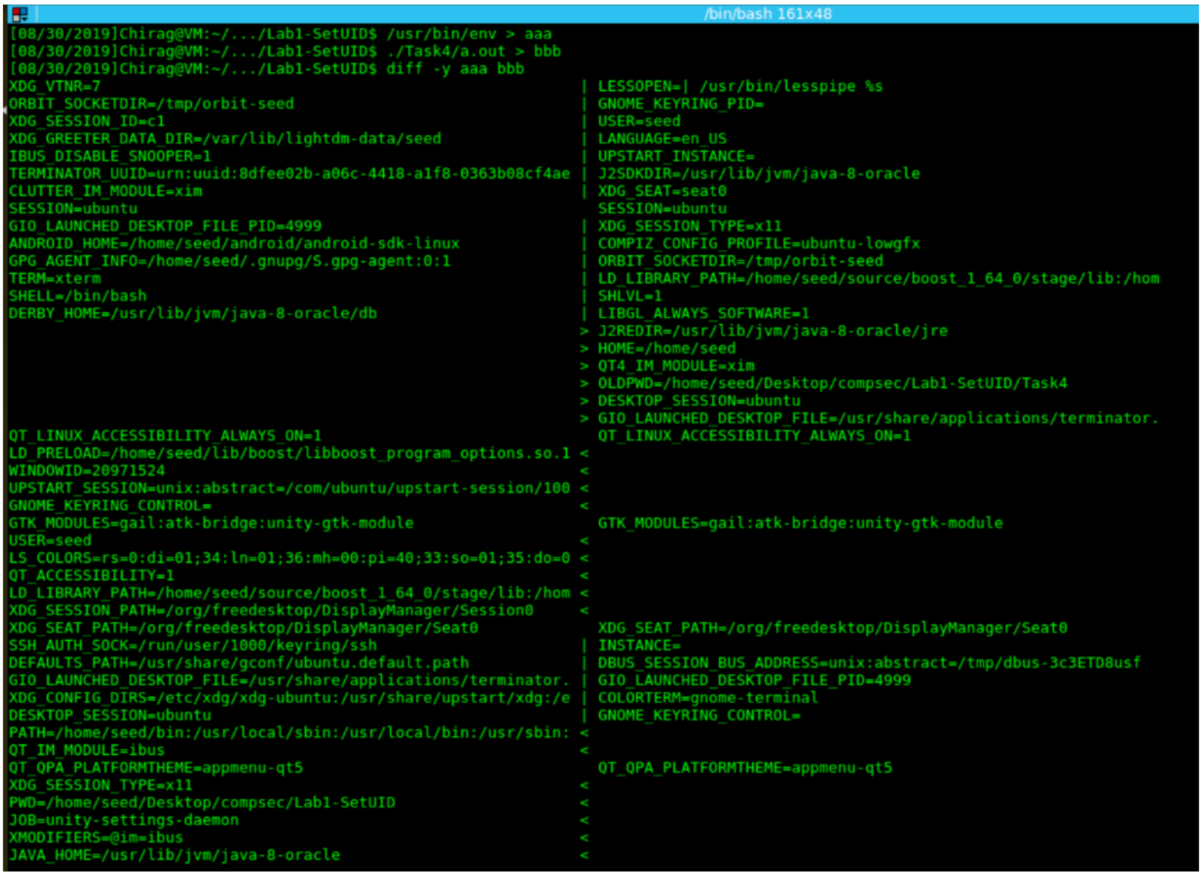
Code:

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



```
/bin/bash 161x48
[08/30/2019]Chirag@VM:~/.../Lab1-SetUID$ /usr/bin/env > aaa
[08/30/2019]Chirag@VM:~/.../Lab1-SetUID$ ./Task4/a.out > bbb
[08/30/2019]Chirag@VM:~/.../Lab1-SetUID$ diff -y aaa bbb
XDG_VTNR=7                                                | LESSOPEN=| /usr/bin/lesspipe %s
ORBIT_SOCKETDIR=/tmp/orbit-seed                           | GNOME_KEYRING_PID=
XDG_SESSION_ID=c1                                         | USER=seed
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed           | LANGUAGE=en_US
IBUS_DISABLE_SNOOPER=1                                    | UPSTART_INSTANCE=
TERMINATOR_UUID=urn:uuid:8dfee02b-a06c-4418-a1f8-0363b08cf4ae | J2SDKDIR=/usr/lib/jvm/java-8-oracle
CLUTTER_IM_MODULE=xim                                     | XDG_SEAT=seat0
SESSION=ubuntu                                              SESSION=ubuntu
GIO_LAUNCHED_DESKTOP_FILE_PID=4999                        | XDG_SESSION_TYPE=x11
ANDROID_HOME=/home/seed/android/android-sdk-linux         | COMPIZ_CONFIG_PROFILE=ubuntu-lowgfx
GPG_AGENT_INFO=/home/seed/.gnupg/S.gpg-agent:0:1          | ORBIT_SOCKETDIR=/tmp/orbit-seed
TERM=xterm                                                | LD_LIBRARY_PATH=/home/seed/source/boost_1_64_0/stage/lib:/hom
SHELL=/bin/bash                                           | SHLVL=1
DERBY_HOME=/usr/lib/jvm/java-8-oracle/db                  | 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                          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                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         XDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat0
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh                  | INSTANCE=
DEFAULTS_PATH=/usr/share/gconf/ubuntu.default.path        | DBUS_SESSION_BUS_ADDRESS=unix:abstract=/tmp/dbus-3c3ETD8usf
GIO_LAUNCHED_DESKTOP_FILE=/usr/share/applications/terminator. | GIO_LAUNCHED_DESKTOP_FILE_PID=4999
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/usr/share/upstart/xdg:/e | COLORTERM=gnome-terminal
DESKTOP_SESSION=ubuntu                                    | GNOME_KEYRING_CONTROL=
PATH=/home/seed/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin: <
QT_IM_MODULE=ibus                                         <
QT_QPA_PLATFORMTHEME=appmenu-qt5                            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:

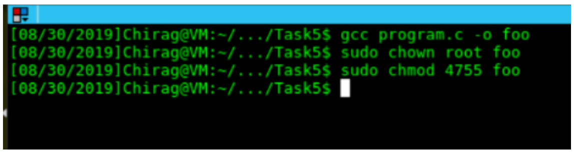
Environment Variable and SetUID programs

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

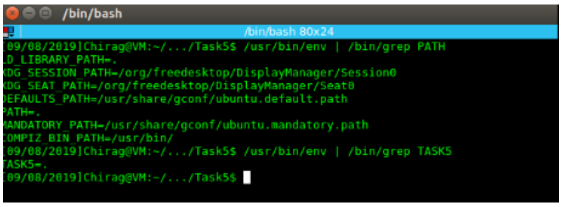
Code:

```c
#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);
}
```

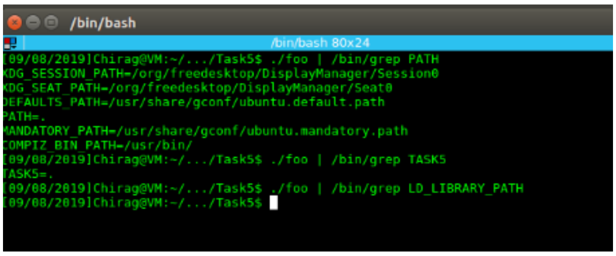
We compile and set SetUID privilages to the program above as shown below.



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



We run the SetUID program to print the environment variables.



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.

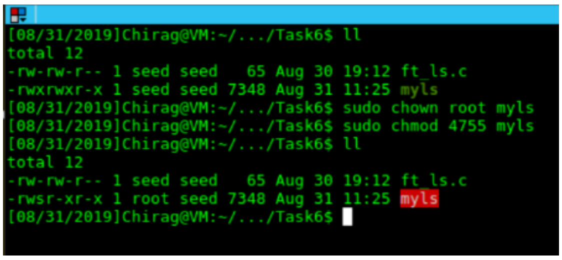
Task 6:

PATH environment varibale and SetUID programs.

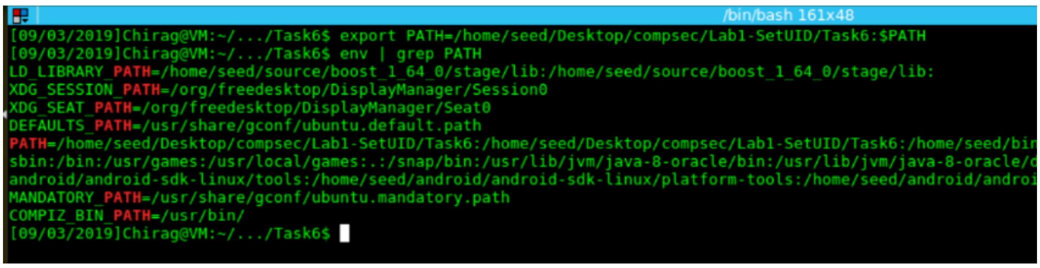
Code:

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

    return (0);
}
```

We compile the program above and give it SetUID privilages.



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

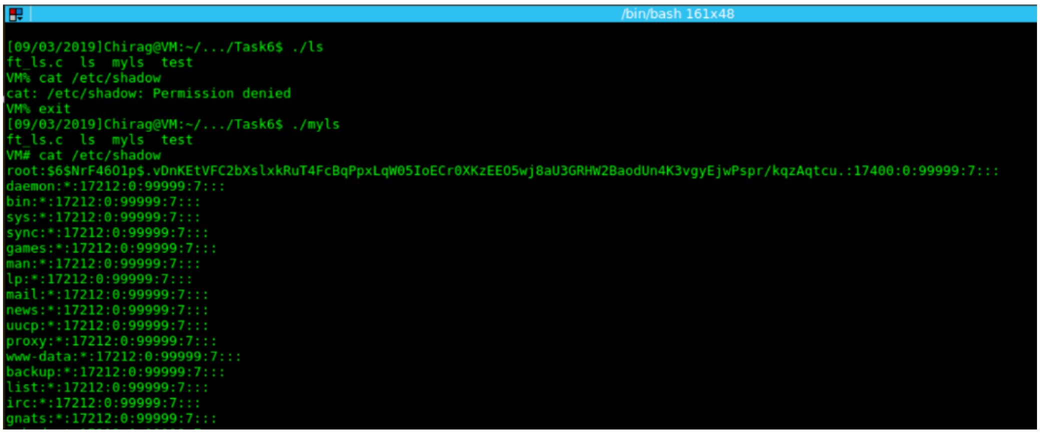


The contents of the malicious ls file are shown below.

We run the malicious ls program and try to read the /etc/shadow file which is a privilaged 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 privilages. This is verified by reading the contents of the /etc/shadow file.



Task 7.

LD_PRELOAD and SetUID Programs

Code for mylib:

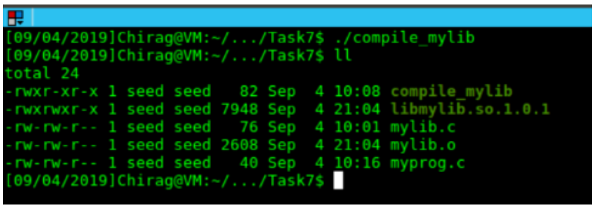
```c
#include <stdio.h>

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.

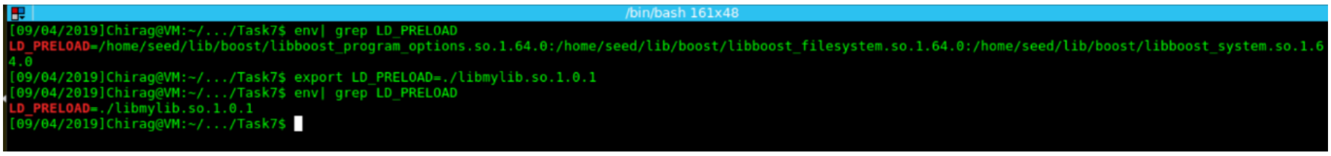
#!/bin/bash

gcc -fPIC -g -c mylib.c

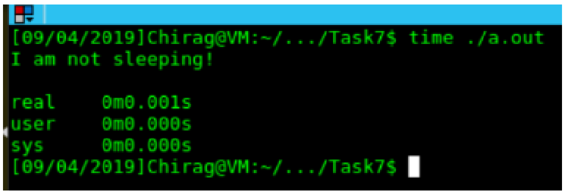
gcc -shared -o libmylib.so.1.0.1 mylib.o -lc



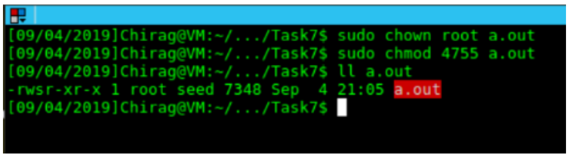
We then export the LD_PRELOAD variable to the custon library created as shown below.



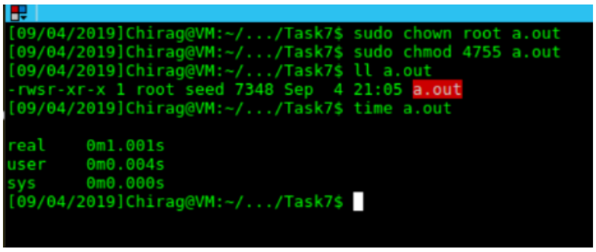
We run the program in the regular mode to see that the library is invoked.



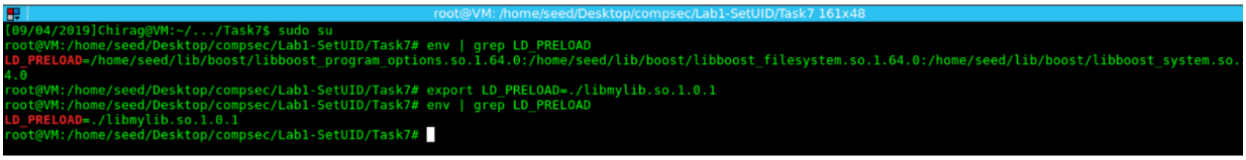
We then se the SetUID privilages to the program and then execute it.



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.

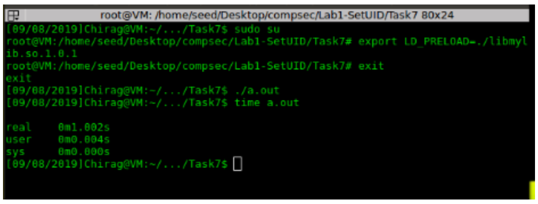


We then select the root account add export the LD_PRELOAD library to the root shell as shown below.



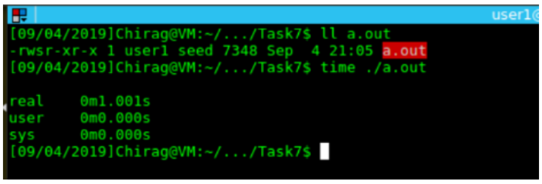
We exit the superuser account and then run the SetUID program again.

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



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

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



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().

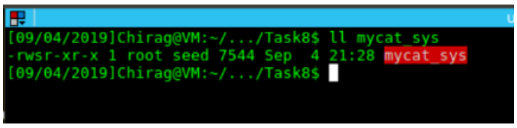
Program.

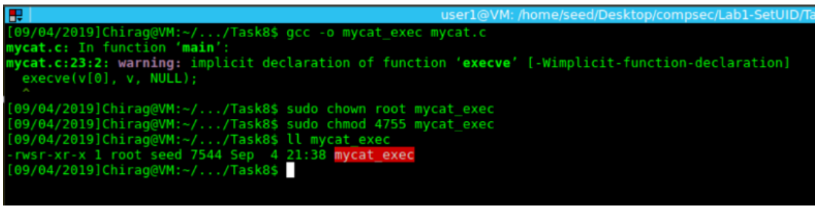
```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

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 sysstsem call out. I compile the program as mycat_exec.

I provide SetUID privilage to both programs as shown below

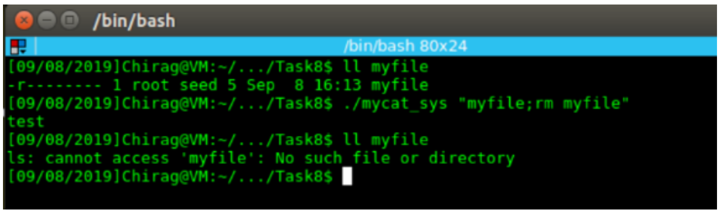




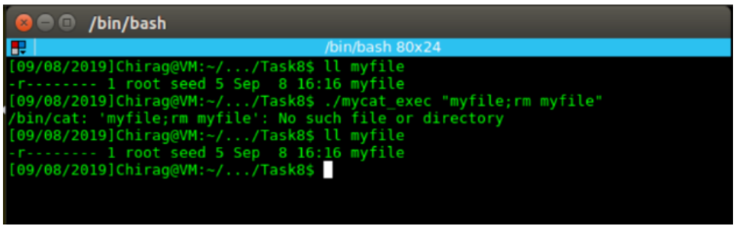
To test the exploit, I create a test file “myfile” with root as the owner and assign 400 permissions to it.

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 privilages and removes the file which is a root owned file.



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



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

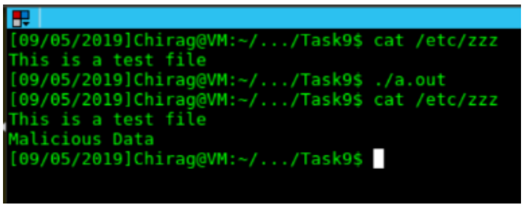
Task 9:

Capability Leaking

Code:

```c
#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.



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

The file descripter was assigned with root privilage and the file wasn’t closed before setting the user ID back to the execution flow.

The privilage of the file descripter was exploited and the malicious code was injected.

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