Lab 3 Environment Variable and Set-UID Program Lab

> 16307130212 管佳乐

Task 1: Manipulating Environment Variables

Task 1.1: Print out the environment variables

```
# printenv - print all or part of environment
$ printenv PWD
# env - run a program in a modified environment
# grep, egrep, fgrep, rgrep - print lines matching a pattern
$ env | grep PWD
```

```
[04/03/19]seed@VM:~/Desktop$ printenv PWD
/home/seed/Desktop
[04/03/19]seed@VM:~/Desktop$ env | grep PWD
OLDPWD=/home/seed
PWD=/home/seed/Desktop
```

The first line treat PWD as a argument.

The second line would print out all the environment variables and we use `regex PWD` to find corresponding record.

Task 1.2 Set or unset environment variables

```
# The supplied names are marked for automatic export to the environment of subsequently
executed commands.
$ export CHIALE="16307130212"
# For each name, remove the corresponding variable or function
$ unset CHIALE
```

```
[04/03/19]seed@VM:~/Desktop$ export CHIALE="16307130212"
[04/03/19]seed@VM:~/Desktop$ printenv CHIALE
16307130212
[04/03/19]seed@VM:~/Desktop$ unset CHIALE
[04/03/19]seed@VM:~/Desktop$ printenv chiale
[04/03/19]seed@VM:~/Desktop$
```

As the manual of `bash` says, `export` would be marked for subsequently executed commands. While `unset` would remove the record.

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

Task 2.1: Compile and Run fork()

```
[04/03/19]seed@VM:~/.../lab3$ make 2
gcc 2.c -o 2.o
./2.o > 2.txt
[04/03/19]seed@VM:~/.../lab3$ vi 2.txt
[04/03/19]seed@VM:~/.../lab3$ cat 2.txt | grep CHIALE
CHIALE=16307130212
```

As the manual of fork puts it, there are many differences between the parent process and its child. Since environment variables are not mentioned, we can speculate that they are shared. And our observation comply with the speculation. The environment variable `CHIALE` is shared by parent and child process.

Task 2.2: The Environment Variables of Parent Process

```
[04/03/19]seed@VM:~/.../lab3$ make 22
gcc 22.c -o 22.o
./22.o > 22.txt
[04/03/19]seed@VM:~/.../lab3$ diff 2.txt 22.txt
9c9
< _=/home/seed/Desktop/lab3/./2.o
---
> _=/home/seed/Desktop/lab3/./22.o
```

The difference lies in line 9. And `_` indicates the current file name. That is the only difference of their environment variables. Parent and child process share most of their environment variables.

Task 3: Environment Variables and execve()

Task 3.1: Compile and Run execve()

the pith of `3.c`

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

And we got no environment variable. There is no output.

```
[04/04/19]seed@VM:~/.../lab3$ make 3
gcc 3.c -o 3.o
[04/04/19]seed@VM:~/.../lab3$ ./3.o
[04/04/19]seed@VM:~/.../lab3$ 
```

Task 3.2: Edit and Try

the pith of `32.c`

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

Then we got the environment variable list.

```
[04/04/19]seed@VM:~/.../lab3$ make 32
gcc 32.c -o 32.o
[04/04/19]seed@VM:~/.../lab3$ ./32.o
XDG_VTNR=7
XDG_SESSION_ID=c1
CLUTTER_IM_MODULE=xim
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed
SESSION=ubuntu
GPG_AGENT_INFO=/home/seed/.gnupg/S.gpg-agent:0:1
SHELL=/bin/bash
TERM=xterm-256color
XDG_MENU_PREFIX=gnome-
VTE_VERSION=4205
QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
WINDOWID=60817418
UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session/1000/1373
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;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:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.
```

Task 3.3: How the New Program Gets Its Environment Variables.

I've consulted the manual of `execve`. As it reads, the environment variables are passed to the new program by argument `envp`. That explains the difference in prior tasks.

```
int execve(const char *filename, char *const argv[], char *const envp[]);
// envp is an array of strings,
// conventionally of the form key=value,
// which are passed as environment to the new program.
```

Task 4: Environment Variables and system()

```
/*The  system()  library  function uses fork(2) to create a child process that executes the
shell command specified in command using execl(3) as follows:*/
           execl("/bin/sh", "sh", "-c", command, (char *) 0);
```

From the manual of `system()`, we can tell that actually it executed as

```
$ /bin/sh -c /usr/bin/env
```

And the environment variable array was passed to `sh`.

```
[04/04/19]seed@VM:~/.../lab3$ make 4
gcc 4.c -o 4.o
[04/04/19]seed@VM:~/.../lab3$ ./4.o
XDG_VTNR=7
XDG_SESSION_ID=c1
CLUTTER_IM_MODULE=xim
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed
SESSION=ubuntu
GPG_AGENT_INFO=/home/seed/.gnupg/S.gpg-agent:0:1
SHELL=/bin/bash
TERM=xterm-256color
XDG_MENU_PREFIX=gnome-
VTE_VERSION=4205
QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
WINDOWID=60817418
UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session/1000/1373
```

I compared the output of directly using `execve()` and using `system()`. The difference lies in the current file name. And line 64 tells `system()` might have changed the current directory.

```
[04/04/19]seed@VM:~/.../lab3$ diff 3.txt 4.txt
63c63,64
< _=./32.o
---
> _=/usr/bin/env
> OLDPWD=/home/seed/Desktop/lab3
```

Task 5: Environment Variable and Set-UID Programs

Compile and run, then make it a root and Set-UID program.

```
$ sudo chown root 5.o
$ sudo chmod 4755 5.o
# append .. folder to original
export
PATH="/home/seed/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/game
s:/usr/local/games:.:/snap/bin:.."
export
LD_LIBRARY_PATH="/home/seed/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bi
n:/usr/games:/usr/local/games:.:/snap/bin:.."
"
```

```
[04/04/19]seed@VM:~/.../lab3$ vi Makefile
[04/04/19]seed@VM:~/.../lab3$ make 5
gcc 5.c -o 5.o
[04/04/19]seed@VM:~/.../lab3$ sudo chown root 5.o
[sudo] password for seed:
[04/04/19]seed@VM:~/.../lab3$ sudo chmod 4755 5.o
```

```
[04/04/19]seed@VM:~/.../lab3$ cat 5.txt | grep CHIALE
CHIALE=16307130212
[04/04/19]seed@VM:~/.../lab3$ cat 5.txt | grep PATH
XDG_SESSION_PATH=/org/freedesktop/DisplayManager/Session0
XDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat0
DEFAULTS_PATH=/usr/share/gconf/ubuntu.default.path
PATH=/home/seed/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:.:/snap/bin:..
MANDATORY_PATH=/usr/share/gconf/ubuntu.mandatory.path
```

We can tell that Set-UID program's process has inherited `CHIALE` and `PATH` from the shell process.

But `LD_LIBRARY_PATH` was not inherited. The linker is quite discreet about choosing its path.

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

Task 6.1: Preparations

Link to another shell first.

```
$ sudo rm /bin/sh
$ sudo ln -s /bin/zsh /bin/sh
```

Append a malicious directory to `PATH`

```
export PATH=/home/seed:$PATH
```

Task 6.2: My Malicious Code

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

extern char **environ;
int main(){
    char *argv[2];
    argv[0] = "/usr/bin/whoami";
    argv[1] = NULL;
    execve("/usr/bin/whoami", argv, environ);
    return 0 ;
}
```

So we we type `ls` in the shell, it will invoke `/home/seed/ls` instead of invoking `/bin/ls`

```
$ gcc ls.c -o ls
```

Task 6.3: Compile and Run

```
[04/15/19]seed@VM:~/.../lab3$ make 6
gcc 6.c -o 6.o
sudo chown root 6.o
sudo chmod 4755 6.o
[04/15/19]seed@VM:~/.../lab3$ sudo chown root 6.o
[04/15/19]seed@VM:~/.../lab3$ sudo chmod 4755 6.o
[04/15/19]seed@VM:~/.../lab3$ ./6.o
root
[04/15/19]seed@VM:~/.../lab3$ which ls
/home/seed/ls
[04/15/19]seed@VM:~/.../lab3$ 
```

The root Set-UID program ran my customized `ls` successfully since my malicious directory is in front of `/bin` in `PATH`. If we use `which ls` to locate the command, we can tell that it locate `ls` in my malicious directory instead of `/bin`.

Moreover, we can tell that my code is running with the root privilege. Since a Set-UID program would give temporary permissions to a user to run a program/file with the permissions of the file owner rather that the user who runs it. So we would get the privilege of the file owner, root.

Task 7: The LD PRELOAD Environment Variable and Set-UID Programs

Task 7.1: Compile

Build a dynamic link library

```
#include <stdio.h>
void sleep (int s){
/* If this is invoked by a privileged program, you can do damages here! */
    printf("I am not sleeping!\n");
}
```

Change the environment variable for dynamic loader/linker.

```
export LD_PRELOAD=./libmylib.so.1.0.1
```

```
[04/04/19]seed@VM:~/.../lab3$ make 7
gcc -fPIC -g -c mylib.c
gcc -shared -o libmylib.so.1.0.1 mylib.o -lc
[04/04/19]seed@VM:~/.../lab3$ export LD_PRELOAD=./libmylib.so.1.0.1
[04/04/19]seed@VM:~/.../lab3$ printenv LD_PRELOAD
./libmylib.so.1.0.1
```

Task 7.2: Run

```
/* myprog.c */
int main(){
    sleep(1);
    return 0;
}
```

Make `myprog` a regular program, and run it as a normal user.

```
[04/04/19]seed@VM:~/.../lab3$ ./myprog.o
I am not sleeping!
```

Make `myprog` a Set-UID root program, and run it as a normal user.

```
$ sudo chmod 4755 myprog.o
$ sudo chown root myprog.o
```

```
[04/04/19]seed@VM:~/.../lab3$ sudo chown root myprog.o
[04/04/19]seed@VM:~/.../lab3$ sudo chmod 4755 myprog.o
[04/04/19]seed@VM:~/.../lab3$ ./myprog.o
[04/04/19]seed@VM:~/.../lab3$
```

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

```
[04/04/19]seed@VM:~/.../lab3$ sudo su
root@VM:/home/seed/Desktop/lab3# export LD_PRELOAD=./libmylib.so.1.0.1
root@VM:/home/seed/Desktop/lab3# ./myprog.o
I am not sleeping!
```

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.

```
root@VM:/home/seed/Desktop/lab3# adduser flora
Adding user `flora' ...
Adding new group `flora' (1001) ...
Adding new user `flora' (1001) with group `flora' ...
Creating home directory `/home/flora' ...
Copying files from `/etc/skel' ...
ERROR: ld.so: object './libmylib.so.1.0.1' from LD_PRELOAD cannot be preloaded (cannot open shared object file): ignored.
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for flora
Enter the new value, or press ENTER for the default
        Full Name []: Flora
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] y
```

```
root@VM:/home/seed/Desktop/lab3# chown flora myprog.o
root@VM:/home/seed/Desktop/lab3# exit
exit
```

```
[04/04/19]seed@VM:~/.../lab3$ su flora
Password:
flora@VM:/home/seed/Desktop/lab3$ export LD_PRELOAD=./libmylib.so.1.0.1
flora@VM:/home/seed/Desktop/lab3$ ./myprog.o
I am not sleeping!
```

Task 7.3: Analyze

```
[04/15/19]seed@VM:~/.../lab3$ export LD_PRELOAD=./libmylib.so.1.0.1
[04/15/19]seed@VM:~/.../lab3$ printenv > seed.txt
[04/15/19]seed@VM:~/.../lab3$ sudo su
[sudo] password for seed:
root@VM:/home/seed/Desktop/lab3# printenv > root.txt
root@VM:/home/seed/Desktop/lab3# diff seed.txt root.txt
3,4c3
< LD_PRELOAD=./libmylib.so.1.0.1
< USER=seed
---
> USER=root
10,11c9,10
< MAIL=/var/mail/seed
< PATH=/home/seed:/home/seed/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:.
---
> MAIL=/var/mail/root
> PATH=/home/seed/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:.
14c13
< SHLVL=3
---
> SHLVL=1
16c15
< HOME=/home/seed
---
> HOME=/root
18c17
< LOGNAME=seed
---
> LOGNAME=root
root@VM:/home/seed/Desktop/lab3#
```

After I have exported the environment variable `LD_PRELOAD`, we can see from the output of `printenv` that different users do not share `LD_PRELOAD`.

And from Task 5, we have known that `LD_*` will not be inherited from user's process to Set-UID program.

So in the 2nd scenario, when we run the program with root's privilege and using root's environment variables, `LD_PRELOAD` exists only in seed's environment variable array. Therefore we would not successfully link the library.

In other scenarios, we have explicitly export `LD_PRELOAD`, so the library would be successfully linked.

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

Task 8.1 Run with system()

Our implementation will invoke `execl()`

```
// command = /bin/cat argv[1]
execl("/bin/sh", "sh", "-c", command, (char *) 0);
```

Since the program would treat our argument as a string, if we pass a string `test; rm test`, then the command would be `/bin/cat test; rm test`. We could first check test and then delete it with root privilege.

```
[04/04/19]seed@VM:~/.../lab3$ make 8
gcc 8.c -o 8.o
sudo chown root 8.o
sudo chmod 4755 8.o
```

```
[04/04/19]seed@VM:~/.../lab3$ ls
22.c  22.txt  2.o    32.c  3.c  3.txt  4.o    5.c  5.txt  6.o  8.o                Makefile  mylib.o   myprog.o
22.o  2.c     2.txt  32.o  3.o  4.c    4.txt  5.o  6.c    8.c  libmylib.so.1.0.1  mylib.c   myprog.c
[04/04/19]seed@VM:~/.../lab3$ echo "test file">test
[04/04/19]seed@VM:~/.../lab3$ ls
22.c  22.txt  2.o    32.c  3.c  3.txt  4.o    5.c  5.txt  6.o  8.o                Makefile  mylib.o   myprog.o
22.o  2.c     2.txt  32.o  3.o  4.c    4.txt  5.o  6.c    8.c  libmylib.so.1.0.1  mylib.c   myprog.c  test
[04/04/19]seed@VM:~/.../lab3$ ./8.o "test; rm test"
test file
[04/04/19]seed@VM:~/.../lab3$ ls
22.c  22.txt  2.o    32.c  3.c  3.txt  4.o    5.c  5.txt  6.o  8.o                Makefile  mylib.o   myprog.o
22.o  2.c     2.txt  32.o  3.o  4.c    4.txt  5.o  6.c    8.c  libmylib.so.1.0.1  mylib.c   myprog.c
```

Task 8.2 Run with execve()

If we invoke `execve()`, since our input will not be append to `filename`, the file to be executed is `/bin/cat`. We can see that our input locates in `argv` and it will be regarded as the argument.

```
// filename = /bin/cat
// argv[] = /bin/cat argv[1] NULL
// envp[] = NULL
int execve(const char *filename, char *const argv[], char *const envp[]);
```

So it actually executes as

```
$ /bin/cat 'test; rm test'
```

And there is no such file. The try failed.

```
[04/04/19]seed@VM:~/.../lab3$ make 82
gcc 82.c -o 82.o
82.c: In function ‘main’:
82.c:18:3: warning: implicit declaration of function ‘execve’ [-Wimplicit-function-declaration]
   execve(v[0], v, NULL);
   ^
sudo chown root 82.o
sudo chmod 4755 82.o
```

```
[04/04/19]seed@VM:~/.../lab3$ echo "test file">test
[04/04/19]seed@VM:~/.../lab3$ ./82.o "test; rm test"
/bin/cat: 'test; rm test': No such file or directory
```

Task 9: Capability Leaking

Even though when the child process is executed the program has relinquished root privilege, the program did not clean up its capability of reading and writing `/etc/zzz`. So the file can still be accessible by the child process. So we can still append `Malicious Data` to `/etc/zzz`.

```
[04/04/19]seed@VM:~/.../lab3$ make 9
gcc 9.c -o 9.o
sudo chown root 9.o
sudo chmod 4755 9.o
[04/04/19]seed@VM:~/.../lab3$ ./9.o
[04/04/19]seed@VM:~/.../lab3$ cat /etc/zzz
zzzzzz
Malicious Data
```

It has changed the content successfully.

Question

- Why the program could read and write `/etc/zzz` in the first place.

From the experiment below, we can see that only if we have changed the file mode bits can we open the file. I think it might have something to do with `4`, the SUID bit.

> 92.c is an excerpt of 9.c to detect whether can we open the file

```
[04/04/19]seed@VM:~/.../lab3$ make 92
gcc 92.c -o 92.o
[04/04/19]seed@VM:~/.../lab3$ ./92.o
Cannot open /etc/zzz
[04/04/19]seed@VM:~/.../lab3$ sudo chown root 92.o
[sudo] password for seed:
[04/04/19]seed@VM:~/.../lab3$ ./92.o
Cannot open /etc/zzz
[04/04/19]seed@VM:~/.../lab3$ sudo chmod 4755 92.o
[04/04/19]seed@VM:~/.../lab3$ ./92.o
Open successfully
[04/04/19]seed@VM:~/.../lab3$
```

I referred to the site https://www.linux.com/blog/what-suid-and-how-set-suid-linuxunix

> Normally in Linux/Unix when a program runs, it inherits access permissions from the logged in user. SUID is defined as giving temporary permissions to a user to run a program/file with the permissions of the file owner rather that the user who runs it

Seemingly we are running as a normal user. But since we've set the SUID bit, we are actually running with the permission of the file owner (root). So we can read and write the file in the first place.