Author: Rajat Sethi

Class: CPSC-6200

Date: September 21, 2021

**Assignment 1 Report – Environment Variables and Set-UID**

**Task 1.1:**

The following screenshots show the results of running “env” and “printenv PWD.”

Graphical user interface, text, application

Description automatically generated

A picture containing text, fish

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Chart

Description automatically generated with low confidence

**Task 1.2:**

I used “export” to create an environment variable TASK, then removed it with “unset”

Text

Description automatically generated

**Task 2:**

Conclusion - As shown in the screenshot, there is no difference between the child process’ environment variables and the parent process’

Text

Description automatically generated

**Task 3.1:**

In its original form, ‘myenv.c’ cannot list any environment variables because execve() was not given an environment to look at.

Text

Description automatically generated

**Task 3.2:**

When “environ” is passed in as a parameter to execve(), the program prints out all of the environment variables.

Text

Description automatically generated

**Conclusion:** “environ” is a special variable that exists in “GLIBC” library and can be declared using the extern keyword. It is a pointer to a list of strings (char\*\*) that points to whatever the system environment is. When “environ” is declared and passed into the execve() function, it prints out the environment variables.

**Task 4:**

The output of running the code provided for Task 4 with the system(“usr/bin/env”) function.

A screenshot of a computer

Description automatically generated with medium confidence

**Task 5:**

As shown in the following picture, PATH and ANY\_NAME both existed in environ. However, even though LD\_LIBRARY\_PATH was exported like ANY\_NAME, it was surprisingly not called in the program.

Text

Description automatically generated

**Task 6:**

After changing the shell from /bin/sh to /bin/zsh, I was able to use the system() to print out the contents of /etc/shadow. Of course, this was only possible by changing the owner to root and turning on the Set-UID bit.



A picture containing text

Description automatically generated

Text, table

Description automatically generated

**Task 7:**

Running “myprog” as a normal user without changing owner or Set-UID. In this scenario, the code runs with the new sleep command, since LD\_PRELOAD was changed and Set-UID is off.

Text

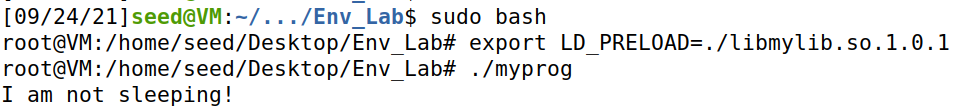
Description automatically generated

Running “myprog” as the normal user, when it’s owned by root user, and Set-UID is on. In this scenario, nothing outputs (and the terminal goes to the next command). This is because the root user currently does not have the LD\_PRELOAD environment variable set to libmylib.so.1.0.1, so the “sleep” command was not overwritten.

Text

Description automatically generated

Running “myprog” as the root user, owned by the root user, and Set-UID is on. In this scenario, the code runs with the new sleep command as expected.



Running “myprog” as “user1,” owned by “user1,” and Set-UID is on. In this scenario, the regular user is denied because Set-UID is on.

Text

Description automatically generated

**Task 8.1:**

In the “catall” program, the system() function runs any command put into it. The input is not adequately filtered, and so anyone can use the “&&” operation to run any other command they want, as shown in the screenshot.

Text

Description automatically generated

**Task 8.2:**

With system() replaced with execve(), the exploit no longer works and I can no longer run any command of my choosing, as shown in the screenshot.

Text

Description automatically generated

**Task 9:**

Using the file pointer and capability leak given in the problem, I was able to echo a line into /etc/zzz and overwrite the file’s contents.

Text

Description automatically generated