Indian Institute of Technology Jammu
Information Security (CSC050P1M)
Assignment 4
Environment Variable and Set-UID
Deadline: 11:59 PM, February 10, 2023

1 Overview

The learning objective of this lab is for students to understand how environment variables affect program and system behaviours. Environment variables (EV) are a set of dynamic named values that can affect the way running processes will behave during computation. EVs are used by most operating systems. EVs were introduced in Unix in 1979. Although environment variable affect program behaviours, how they achieve their infleunce is not well understood by many programmers, which is key for execution behaviour including security. As a result, if a program uses environment variables, but the programmer does not know that EVs are used, the program may have vulnerabilities.

After completing this lab, students will understand how environment variables work, how they are propagated from parent process to child, and how they affect system/program be haviors. The students need to understand how environment variables affect the behavior of Set-UID programs, which are usually privileged programs. This lab covers the following topics:

- Environment variables
- Set-UID programs

2 Problems

1. Manipulating Environment Variables

   In this task, we study the commands that can be used to set and unset environment vari ables. Here, we will use Bash. The default shell that a user uses is set in the /etc/passwd file (the last field of each entry). You can change this to another shell using the command chsh (please do not do it for this lab). Please do the following tasks:

   - Use *printenv* or *env* command to print out the environment variables. If you are interested in some particular environment variables, such as PWD, you can use “print env PWD” or “env — grep PWD”.
   - Use export and unset to set or unset environment variables. It should be noted that these two commands are not separate programs; they are two of the Bash’s internal
     commands (you will not be able to find them outside of Bash). Use “script” on

command prompt to store your report.

2. Passing Environment Variables from Parent Process to Child Process

   In this task, we study how a child process gets its environment variables from its parent. In Unix, fork() creates a new process by duplicating the calling process. The new process, referred to as the child, is an exact duplicate of the calling process, referred to as the parent; however, several things are not inherited by the child (please read the man pages or info of fork() by typing the following command: *man fork*). In this task, we would like to know whether the parent's environment variables are inherited by the child process or not.

   Step 1: Please compile and run the following program, and describe your observation. Because the output contains many strings, you should save the output into a file, such as using *a.out* > *child* (assuming that a.out is your executable file name).

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
extern char **environ;
void printenv()
{
    int i = 0;
    while (environ[i] != NULL) {
        printf("%s\n", environ[i]);
        i++;
    }
}
void main()
{
    pid_t childPid;
    switch(childPid = fork()) {
        case 0: /* child process */
        printenv();
        exit(0);
        default: /* parent process */
        //printenv();
    exit(0);
    }
}
```

   Step 2: Now comment out the *printenv*() statement in the child process case, and un comment the *printenv*() statement in the parent process case. Compile and run the code again, and describe your observation. Save the output in another file.

Step 3: Compare the difference between these two files using the *diff* command. Please draw your conclusion in your response.

3. Environment Variables and *execve*()

In this task, we study how environment variables are affected when a new program is executed via execve(). The function execve() calls a system call to load a new command and execute it; this function never returns. No new process is created; instead, the calling process's text, data, bss, and stack are overwritten by that of the loaded program. Es sentially, execve() runs the new program inside the calling process. We are interested to know what happens to the environment variables; are they automatically inherited by the new program? Write your conclusions by observation.

Step 1: Please compile and run the following program, and describe your observation. This program simply executes a program called /usr/bin/env, which prints out the envi ronment variables of the current process.

```
#include <stdio.h>
#include <stdlib.h>
extern char **environ;
int main()
{
      char *argv[2];
      argv[0] = "/usr/bin/env";
      argv[1] = NULL;
      execve("/usr/bin/env", argv, NULL);
      return 0 ;
}
```

Step 2: Change the invocation of execve() to the following; describe your observation.

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

Step 3: Please draw your conclusion regarding how the new program gets its environment variables. Write your observation in your report.

4. Environment Variables and *system*()

In this task, we study how environment variables are affected when a new program is executed via the system() function. This function is used to execute a command, but unlike execve(), which directly executes a command, system() actually executes ”/bin/sh

-c command", i.e., it executes /bin/sh, and asks the shell to execute the command.

If you look at the implementation of the system() function, you will see that it uses execl() to execute /bin/sh; execl() calls execve(), passing to it the environment variables array. Therefore, using system(), the environment variables of the calling process is passed to the new program /bin/sh. Please compile and run the following program to verify this.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
system("/usr/bin/env");
return 0 ;
}
```

5. Environment Variable and Set-UID Programs

Set-UID is an important security mechanism in Unix/Linux operating systems. When a Set-UID program runs, it assumes the owner's privileges. For example, if the program's owner is root, then when anyone runs this program, the program gains the root's privileges during its execution. Set-UID allows us to do many interesting things, but it escalates the user's privilege when executed, making it risky from privileges point of view. Although the behaviours of Set-UID programs are decided by their program logic, not by users, users can indeed affect the behaviours via environment variables. To understand how Set-UID programs are affected, let us first figure out whether environment variables are inherited by the Set-UID program's process from the user's process.

Step 1: Write the following program that can print out all the environment variables in the current process.

```
#include <stdio.h>
#include <stdlib.h>
extern char **environ;
void main()
{
    int i = 0;
    while (environ[i] != NULL) {
    printf("%s\n", environ[i]);
    i++;
```

```
        }
        }
```

Step 2: Compile the above program, change its ownership to root, and make it a Set-UID program.

```
// Asssume the program's name is foo
$ sudo chown root foo
$ sudo chmod 4755 foo
```

Step 3: In your command line shell (you need to be in a normal user account, not the root account), use the export command to set the following environment variables (they may have already existed):

- *P AT H*
- *LDLIBRARY P AT H*
- *ANY NAME* (this is an environment variable defined by you, so pick whatever name you want).

These environment variables are set in the user's shell process. Now, run the Set-UID program from Step 2 in your shell. After you type the name of the program in your shell, the shell forks a child process, and uses the child process to run the program. Please check whether all the environment variables you set in the shell process (parent) get into the Set-UID child process. Describe your observation. Does anything surprise you? Describe the same.

6. The PATH Environment Variable and Set-UID Programs

Because of the shell program invoked, calling system() within a Set-UID program is quite dangerous. This is because the actual behavior of the shell program can be affected by environment variables, such as PATH; these environment variables are provided by the user, who may be malicious. By changing these variables, malicious users can control the behavior of the Set-UID program. In Bash, you can change the PATH environment variable in the following way (this example adds the directory /home/seed to the beginning of the PATH environment variable):

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

The Set-UID program below is supposed to execute the /bin/ls command; however, the programmer only uses the relative path for the ls command, rather than the absolute path:

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

Please compile the above program, and change its owner to root, and make it a Set-UID program. Can you let this Set-UID program run your code instead of /bin/ls? If you can, is your code running with the root privilege? Describe and explain your observations regarding consequences.

3 Submission

You need to submit a detailed lab report, having the following:

- with screenshots and typescript outputs to describe what you have done and what you have observed. Give your own assessment individually, of what you have learned from this assignment.
- You also have to submit all the codes you have written with the necessary comments to describe your code.
- Submit a single zip folder containing the report and properly commented code. The size of the zip file should not exceed 25MB. The name of the zip file should be ⟨Institute Id⟩Assg4.zip.

Best wishes