CSCI 191T – Computer Security

Assignment 1: SetUID and Env.Variables Due:9/18/2022 11:59 PM

Instructions:

There are two sections in this assignment. For Section 1, answer all questions completely. For Section 2, execute the code (using files from Assignment1_Files.zip) and answer all questions.

The final submission is a single report (PDF file). You need to submit a detailed report, with:

Section 1 - Detailed answers and observations.

Section 2 - Screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Also list the important code snippets followed by explanation. **Simply attaching code or screenshot without any explanation will not receive credits.**

Section 1

10 points each.

1. Alice runs a Set-UID program that is owned by Bob. The program tries to read from /tmp/x, which is readable to Alice, but not to anybody else. Can this program successfully read from the file?

2. Both system() and execve() can be used to execute external programs. Why is system() unsafe while execve() is safe?

3. The superuser wants to give Alice a permission to view all the files in the system using the **more** command. He does the following:
   $ cp /bin/more /tmp/mymore
   $ sudo chown root /tmp/mymore
   $ sudo chmod 4700 /tmp/mymore

   Basically, the above commands turns /tmp/mymore into a Set-UID program. Right now, because the permission is set to 4700, other users cannot execute the program. The superuser uses another command (now shown) to grant the execution permission only to Alice. We are not assuming that Alice is completely trusted. It is OK if Alice can only read other people's files, but it is not OK if Alice can gain any privilege beyond that, such as writing to other people's files. Read the manual of the **"more"** program and find out what Alice can do to gain more privilege.

4. Assume that you have a file that you would allow other users to read, only if a user's ID is smaller than 1000. Describe how you can actually achieve this.

5. What is the difference between environment variables and shell variables?

6. The followings are two different ways to print out environment variables. Describe their differences:
   $ /usr/bin/env
   $ /usr/bin/strings /proc/$$/environ

7. In Linux, many environment variables are ignored by the dynamic linker if the program to be executed is a Set-UID program. Two such examples are LD PRELOAD and LD LIBRARY PATH. Read the manual of ld-linux (https://linux.die.net/man/8/ld-linux) and explain why the following environment variables are also ignored:

   • LD AUDIT

   • LD DEBUG OUTPUT

Section 2

15 Points each

1. 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 (see the manual 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. Compile and run the following program, and describe your observation.

The program "myprintenv.c", can be found in the "Assignment1_Files.zip"; it can be compiled using "gcc myprintenv.c", which will generate a binary called a.out. Run it and save the output into a file using "a.out > file".

```c
#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();          ①
      exit(0);
    default:  /* parent process */
      //printenv();        ②
      exit(0);
  }
}
```

Step 2. Now comment out the printenv() statement in the child process case (Line ①), and uncomment the printenv() statement in the parent process case (Line ②). Compile and run the code again, and describe your observation. Save the output in another file.

Step 3. Compare the difference of these two files using the *diff* command. Describe your conclusion

2. 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 program loaded. Essentially, execve() runs the new program inside the calling process. We are interested in what happens to the environment variables; are they automatically inherited by the new program?

Step 1. Compile and run the following program, and describe your observation.

The program "myenv.c" can be found in the "Assignment1_Files.zip"; it can be compiled using "gcc myenv.c", This program simply executes a program called /usr/bin/env, which prints out the environment variables of the current process.

```c
#include <unistd.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() in Line ① to the following and describe your observation.
execve("/usr/bin/env", argv, environ);

Step 3. Describe your conclusion regarding how the new program gets its environment variable.