Lab 04: SETUID & Environment variables

1. Overview

The learning objective of this lab is for students to understand how environment variables affect program and system behaviors. Environment variables are a set of dynamic named values that can affect the way running processes will behave on a computer. They are used by most operating systems, including Unix and Windows. Although environment variables affect program behaviors, how they achieve that is not well understood by many programmers. Therefore, if a program uses environment variables, but the programmer does not know that they are used, the program may have vulnerabilities.

In this lab, students will learn how environment variables work, how they are propagated from parent process to child, and how they affect system/program behaviors. We are particularly interested in how environment variables affect the behavior of SETUID programs, which are usually privileged programs.

SETUID is an important security mechanism in Unix operating systems. When a regular program is run, it runs with the privilege of the user executing that program. When a SETUID program is run, it runs with the privilege of the program file owner. For example, if the program's owner is root, then when anyone runs this program, the program gains root's privileges during its execution. SETUID allows us to perform essential tasks, such as changing passwords, but vulnerabilities in SETUID programs can allow an adversary to perform local privilege escalation.

While the SETUID concept is limited to Unix, the problems of dangerous environment variables and local privilege escalation exists on all operating systems. The techniques that we will use to mitigate such vulnerabilities in this assignment can be applied to non-SETUID programs to protect those programs as well. This assignment is based on the SEED Lab project directed by Kevin Du and uses the SEED Lab VM

2. Lab Environment

a) Install VMWare workstation/VirtualBox on your Windows Computer.
b) Download the virtual machine image of a Linux machine for this lab from here.
c) Extract the SEEDUbuntu archive then open it in VMWare, the linux machine will start automatically (login with username: SEED, password:dees).

3. Tasks

3.1 Figure out why passwd command need to be Set-UID program. What would happen if it is not? Copy passwd to your own directory. The copy will not set the Set-UID bit. Run the copied program and observe what happens.

3.2. **Run Set-UID shell programs** in Linux, describe and explain your observations.

(a) Copy /bin/zsh to /tmp and make it a set-root-uid program. Run /tmp/zsh. Will you get root privilege? Please describe your observation.

(b) Instead of copying /bin/zsh, this time, copy /bin/bash to /tmp, make it a set-root-uid program. Run /tmp/bash as a normal user. will you get root privilege? Please describe and explain your observation.

3.3. **The PATH environment variable**. The `system(const char *cmd)` library function can be used to execute a command within a program. The way `system(cmd)` works is to invoke the /bin/sh program, and then let the shell program to execute cmd. Because of the shell program invoked, calling system() within a Set-UID program is extremely dangerous. This is because the actual behavior of the shell program can be

affected by environment variables, such as PATH; these environment variables are under user's control. 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):

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

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

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

Compile the above program, and change its owner to root, 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

3.4 **The difference between system() and execve().** Before you work on this task, please make sure that /bin/sh is point to /bin/zsh.

Background: Bob works for an auditing agency, and he needs to investigate a company for a suspected fraud. For the investigation purpose, Bob needs to be able to read all the files in the company's Unix system; on the other hand, to protect the integrity of the system, Bob should not be able to modify any file. To achieve this goal, Vince, the superuser of the system, wrote a special set-root-uid program(see below), and then gave the executable permission to Bob. This program requires Bob to type a file name at the command line, and then it will run /bin/cat to display the specified file. Since the program is running as a root, it can display any file Bob specifies. However, since the program has no write operations, Vince is very sure that Bob cannot use this special program to modify any file.

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
   char *v[3];
   if(argc < 2)
   {
   printf("Please type a file name.\n");
   return 1;
   }
   v[0] = "/bin/cat"; v[1] = argv[1]; v[2] = NULL;
   char *command = malloc(strlen(v[0]) + strlen(v[1]) + 2);
   sprintf(command, "%s %s", v[0], v[1]);
   system(command);

   // execve(v[0], v, NULL);
  return 0 ;
}
```

(a) Compile the program, make it a root-owned Set-UID program. The program will use system() to invoke the command. If you were Bob, can you compromise the integrity of the system? For example, can you remove any file that is not writable to you?

(b) Comment out the system(command) statement, and uncomment the execve() statement; the program will use execve() to invoke the command. Compile the program, make it a root-owned Set-UID. Do your attacks in task (a) still work? Please describe and explain your observations.

3.5. The LD PRELOAD Environment Variable and Set-UID Programs

In this task, we study how Set-UID programs deal with some of the environment variables. Several environment variables, including LD PRELOAD, LD LIBRARY PATH, and other LD * influence the behavior of dynamic loader/linker. A dynamic loader/linker is the part of an operating system (OS) that loads (from persistent storage to RAM) and links the shared libraries needed by an executable at run time.

In Linux, ld.so or ld-linux.so, are the dynamic loader/linker (each for different types of binary). Among the environment variables that affect their behaviors, LD LIBRARY PATH and LD PRELOAD are the two that we are concerned in this lab. In Linux, LD LIBRARY PATH is a colon-separated set of di- rectories where libraries should be searched for first, before the standard set of directories. LD PRELOAD specifies a list of additional, user-specified, shared libraries to be loaded before all others. In this task, we will only study LD PRELOAD.

Step 1. First, we will see how these environment variables influence the behavior of dynamic loader/linker when running a normal program. Please follow these steps:

1. Let us build a dynamic link library. Create the following program, and name it mylib.c. It basically overrides the sleep() function in libc:

```
#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");
}
```

2. We can compile the above program using the following commands (in the -lc argument, the second character is l):

```
gcc -fPIC -g -c mylib.c

gcc -shared -o libmylib.so.1.0.1 mylib.o –lc
```

3. Now, set the LD PRELOAD environment variable:

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

Finally, compile the following program myprog (put this program in the same directory as libmylib.so.1.0.1):

```
// myprog.c
int main()
{
   sleep(1);
```

```
    return 0;
}
```

Step 2. After you have done the above, please run myprog under the following conditions, and observe what happens.

- Make myprog a regular program, and run it as a normal user.
- Make myprog a Set-UID root program, and run it as a normal user.
- Make myprog a Set-UID root program, export the LD PRELOAD environment variable again in the root account and run it.
- 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.

Step 3. You should be able to observe different behaviors in the scenarios described above, even though you are running the same program. You need to figure out what causes the difference. Environment variables play a role here. Please design an experiment to figure out the main causes, and explain why the behaviors in Step 2 are different. (Hint: the child process may not inherit the LD * environment variables).

3.6 Capability Leaking

To follow the Principle of Least Privilege, Set-UID programs often permanently relinquish their root privileges if such privileges are not needed anymore. Moreover, sometimes, the program needs to hand over its control to the user; in this case, root privileges must be revoked. The setuid() system call can be used to revoke the privileges. According to the manual, "setuid() sets the effective user ID of the calling process. If the effective UID of the caller is root, the real UID and saved set-user-ID are also set". Therefore, if a Set-UID program with effective UID 0 calls setuid(n), the process will become a normal process, with all its UIDs being set to n. When revoking the privilege, one of the common mistakes is capability leaking. The process may have gained some privileged capabilities when it was still privileged; when the privilege is downgraded, if the program does not clean up those capabilities, they may still be accessible by the non-privileged process. In other words, although the effective user ID of the process becomes non-privileged, the process is still privileged because it possesses privileged capabilities. Compile the following program, change its owner to root, and make it a Set-UID program. Run the program as a normal user, and describe what you have observed. Will the file /etc/zzz be modified? Please explain your observation.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

void main()
{
        int fd;
        // Assume that /etc/zzz is an important system file,
        // and it is owned by root with permission 0644.
        // Before running this program, you should creat
        // the file /etc/zzz first.
        fd = open("/etc/zzz", O_RDWR | O_APPEND);
```

```
        if(fd == -1) {
                printf("Cannot open /etc/zzz\n");
                exit(0);
        }
        // Simulate the tasks conducted by the program
        sleep(1);
        // After the task, the root privileges are no longer needed,
        // it's time to relinquish the root privileges permanently.
        setuid(getuid()); // getuid() returns the real uid
        if(fork()) { // In the parent process
                close(fd);
                exit(0);
        } else { // in the child process
                //Now, assume that the child process is compromised, malicious
                //attackers have injected the following statements
                //into this process
                write(fd, "Malicious Data", 14);
                close(fd);
          }
}
```

4. Submission

You need to submit a detailed lab report to describe what you have done and what you have observed. Please provide details using screen shots and code snippets.