Project III
CS 6238: Secure Computer Systems
Exploring Set-UID
Due Date: October 22, 2020 at 11:59 pm EST

Acknowledgement: This project is based on a lab developed by Professor Wenliang Du at Syracuse University. Professor Du has kindly allowed us to use this lab (see copyright notice below for his lab development).

Project Objectives

Set-UID is an important mechanism in Unix operating systems. When a Set-UID program is run, it assumes the program owner's privileges. For example, if the program's owner is root, when anyone having execute permission runs this program, the program gains the root's privileges during its execution. Set-UID allows us to do many useful things, but unfortunately, it can also be exploited in a number of ways. Therefore, the objective of this project is two-fold:

1. Appreciate Set-UID's good side by understanding why Set-UID is needed and how it is implemented
2. Be aware of its bad side by understanding potential security problems it can lead to when used improperly.

One quick point to keep in mind as you work on various tasks - If a question asks whether your program is running as a root, it assumes that you will run the program as a seed user, not root (we all know a program will run with root privileges when launched in the root terminal).

Project Tasks

This is an exploration lab. Your main task is to experiment with the Set-UID mechanism in Linux and write a lab report that describes your findings. You will be provided a VM to complete the tasks. The VM will be shared on Canvas and mirror links will be made available. The password for the “seed” user is “dees” and the root password is “seedubuntu”. You are required to accomplish the following tasks in Linux:

1. **Task One (10 points)**

(a) Why do "passwd", "chsh", "su", and "sudo" commands need to be Set-UID programs. (4 points)
(b) What will happen if they are not? (If you are not familiar with these programs, you should first learn what they do by reading their manual descriptions). (3 points)
(c) Copy these command binary files to your own directory; the copies will not be Set-UID programs. Run the copied programs, observe the results, and describe what you see. (3 points)

2. **Task Two (15 points)**: Run Set-UID shell programs in Linux and describe and explain your observations.

(a) Login as root, copy /bin/zsh to /tmp, and make it a set-root-uid program with permission 4755. Then login as a normal user and run /tmp/zsh. Will you get root privilege? (Please describe your observations).

Note: If you cannot find /bin/zsh in your operating system, please use the following commands to install it:
For Ubuntu:

```
sudo apt-get install zsh
```

Note: in our pre-built Ubuntu VM image, zsh is already installed.

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

Setup for remaining tasks: As you will find out from the previous task, /bin/bash has certain built-in protection that prevents the abuse of the Set-UID mechanism. To see what could be done prior to such a protection scheme was implemented, we are going to use a different shell program called /bin/zsh. In some Linux distributions (such as Fedora and Ubuntu), /bin/sh is actually a symbolic link to /bin/bash. To use zsh, we need to link /bin/sh to /bin/zsh.

Execute the following commands to change the default shell to zsh:

`su` *(Enter root password)*

```
cd /bin
rm sh
ln -s zsh sh
```

3. **Task Three (20 points) 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 by invoking the /bin/sh program, and then letting the shell program execute *cmd*. Because of the invoked shell program, 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.

The 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;
}
```

(a) Can you run the above program with Set-UID (owned by root) instead of /bin/ls to list files? If you can, is your code running with the root privilege? Describe and explain your observations.

(b) Now, change /bin/sh so it points back to /bin/bash and repeat the above attack. Can you get root privilege? Describe and explain your observations.

4. **Task Four (20 points) The difference between *system()* and *execve()***. *Before you work on this task, please make sure that /bin/sh is pointed to /bin/zsh.*

Background: Bob works for an auditing agency, and he needs to investigate a company for a suspected fraud. For the investigation purposes, 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, Charlie, the sysadmin, wrote a special set-root-uid program (see below), and then gave execute 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 root, it can display any file Bob specifies. However, since the program has no write operations, Charlie 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] = 0;


  /* Set q = 0 for Question a, and q = 1 for Question b */

  int q = 0;
  if (q == 0)
  {
    char *command = malloc(strlen(v[0]) + strlen(v[1]) + 2);
    sprintf(command, "%s %s", v[0], v[1]);
    system(command);
  }
  else execve(v[0], v, 0);
  return 0 ;
}
```

a) Set q = 0 in the program. This way, the program will use *system()* to invoke the command.
   - Is this program safe?
   - If you were Bob, can you compromise the integrity of the system? For example, can you remove any file that is not writable by you?

(Hint: remember that *system(*) actually invokes /bin/sh, and then runs the command within the shell environment. We have tried the environment variable in the previous task; here let us try a different attack. Please pay attention to the special characters used in a normal shell environment).

b) Set q = 1 in the program. This way, the program will use *execve()* to invoke the command. Do your attacks in task (a) still work? Please describe and explain your observations.

5. **Task Five (20 points) The LD PRELOAD environment variable**

To make sure Set-UID programs are safe from the manipulation of the LD PRELOAD environment variable, the runtime linker (ld.so) will ignore this environment variable if the program is a Set-UID root program, except for some conditions. We will figure out what these conditions are in this task.

(step 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)
{
  printf("I am not sleeping!\n");
}
```

(step 2) We can compile the above program using the following commands:

```
gcc -fPIC -g -c mylib.c
gcc -shared -W1,-soname,libmylib.so.1 -o libmylib.so.1.0.1 mylib.o -lc
```

(step 3) Now, set the LD PRELOAD environment variable using the following command:

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

(step 4) Compile the following program (put this program in the same directory as libmylib.so.1.0.1):

```
/* myprog.c */
int main()
{
  sleep(1);
  return 0;
}
```

(step 5) Please run myprog under the following conditions and observe what happens. Based on your observations, describe when the runtime linker will ignore the LD PRELOAD environment variable, and explain why.

- 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 and run it in the root account.
- Make myprog a Set-UID user1 program (i.e., the owner is user1, which is another user account), and run it as a different user (not-root user).

6. Task Six (15 points) Relinquishing privileges and cleanup

To be more secure, Set-UID programs usually call *setuid()* system call to permanently relinquish their root privileges. However, sometimes, this is not enough. Compile the following program and make the program a set-root-uid program. Run it in a normal user account and describe what you have observed. Will the file /etc/zzz be modified? Please explain your observations.

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

void main()
{
  int
  fd;
  fd = open("/etc/zzz", O_RDWR | O_APPEND);

/* Assume that /etc/zzz is an important system file, and it is owned by root
with permission 0644 */
/* 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);
  }
}
```

Submission: You need to submit a detailed lab report describing what you have done for each task and what you have observed. You are also required to write explanations for your observations.