Set-UID Program Vulnerability Lab

Lab Description

`Set-UID` is an important security mechanism in Unix operating systems. When a `Set-UID` program is run, 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 unfortunately, it is also the culprit of many bad things. Therefore, the objective of this lab is two-fold: (1) Appreciate its good side: understand why `Set-UID` is needed and how it is implemented. (2) Be aware of its bad side: understand its potential security problems.

Lab Tasks

This is an exploration lab. Your main task is to "play" with the `Set-UID` mechanism in `Linux`, and write a lab report to describe your discoveries. You are required to accomplish the following tasks in `Linux`:

1. Figure out why `"passwd"`, `"chsh"`, `"su"`, and `"sudo"` commands need to be `Set-UID` programs. What will happen if they are not? If you are not familiar with these programs, you should first learn what they can do by reading their manuals. Please copy these commands to your own directory; the copies will not be `Set-UID` programs. Run the copied programs, and observe what happens.

2. 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 observation. If you cannot find `/bin/zsh` in your operating system, please use the following command to install it:

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

     ```
     $ su
       Password: (enter root password)
     # yum install zsh
     ```

   - For `Ubuntu`

     ```
     $ su
       Password: (enter root password)
     # apt-get install zsh
     ```

   (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. (Setup for the rest of the tasks) As you can find out from the previous task, `/bin/bash` has certain built-in protection that prevent the abuse of the `Set-UID` mechanism. To see the life before 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`. The following instructions describe how to change the default shell to `zsh`.

```
$ su
  Password: (enter root password)
# cd /bin
# rm sh
# ln -s zsh sh
```

4. **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 `ls` command, rather than the absolute path:

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

   (a) Can you let this `Set-UID` program (owned by root) run your code instead of `/bin/ls`? 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 still get the root privilege? Describe and explain your observations.

5. **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 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] = 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 to 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.

6. **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.

(a) 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");
}
```

(b) We can compile the above program using the following commands (in the `-Wl` argument, the third character is ℓ, not one; in the `-lc` argment, the second character is ℓ):

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

(c) Now, set the `LD_PRELOAD` environment variable:

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

(d) 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;
}
```

Please run `myprog` under the following conditions, and observe what happens. Based on your observations, tell us 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).

7. **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 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\n", 15);
    close (fd);
  }
}
```

Submission

You need to submit a detailed lab report 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.