Set-UID Program Vulnerability Lab

Due: Feb 18 2020, 11:00 PM (EST)

1 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.

1.1 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.
Late submissions are accepted with 50% grading penalty within 24 hours of the due. Submissions that are late for more than 24 hours will NOT be accepted.
Please submit your solution in PDF or image on `https://www.gradescope.com/courses/88159`, using Entry Code: **9J2VYB**. And kindly choose the right page for your answer to every question.

Collaboration Policy
You can optionally form study groups consisting of up to 3 person per group (including yourself).
Everybody should write individually by themselves, and submit the lab reports separately. DO NOT copy each other's lab reports, or show your lab reports to anyone other than the course staff, or read other students' lab reports.

1.2 Environment

Please download the customized version of the Seed Labs from Google Drive. This is the same VM wwe used for the Set-UID lab
Use this link for steps to setup the VM.

- User ID: seed
- Password: dees
- Root User ID: root
- Password: seedubuntu

2 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. **Question 1:** Please provide the names and NetIDs of your collaborator (up to 2). If you finished the lab alone, write None.

2. **Question 2:** `Set-UID` commands:

   (a) Please learn and use the following commands and match them to their appropritate descriptions:

i. `passwd`	i. Change login shell
ii. `chsh`	ii. Execute a command as another user
iii. `su`	iii. Modify a user's password
iv. `sudo`	iv. Print user and group information
v. `id`	v. Change user ID or become superuser

   (b) Why do commands `i - iv` need to be `Set-UID` ? What happens if they are not?

   (c) Please copy these commands(`i - iv`) to your own directory; the copies will not be `Set-UID` programs. Run the copied programs, and report the new observations.

3. (Setup for the rest of the tasks) The`/bin/bash` has certain built-in protection that prevent the abuse of the `Set-UID` mechanism. It does not allow you to get the root privilege by running the `Set-UID` program as it ignores the use of `Set-UID` bit option. 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;
}
```

Compile the program with

```
$ su
# gcc -o system system.c
# chmod u+s system
# exit
```

Create a directory and create a program `ls.c` as follows:

```
#include<stdio.h>
int main(){
    printf("You can't see your directories
    but I can see you\n");
    system("xeyes");
    return 0;
}
```

Compile the program with `$ gcc -o ls ls.c`

Question 3: Describe and explain your observations for the following questions with screenshots of successful execution.

(a) Can you let the `Set-UID` program (owned by root) run your ls.c code instead of `/bin/ls`?

(b) Now copy `/bin/sh` to your current directory as `/[your dir]/ls` and run the above code. What do you observe? Is your code running with the root privilege? (Hint: Use a command from Question 1a)

(c) Now, change `/bin/sh` so it points back to `/bin/bash`, and repeat the above attack. Can you still get the root privilege?

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.

```c
#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 ;
}
```

Question 4: Compile the code and create a file called topSecret in the following way:

```
$ sudo su
# gcc -o readOnly readOnly.c
# chmod u+s readOnly
# echo "very confidential information" > topSecret
# chown root:root topSecret
# chmod 700 topSecret
# exit
```

(a) Set $q = 0$ in the above 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? More specifically, find a file name that will help you gain access to the file. (Hint: Echoing might help. Remember that `system()` actually invokes `/bin/sh`, and then runs the command within the shell environment. Here let us try a different attack like injection. 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.

3 Acknowledgements

This document was modified from the SEED lab instruction. The original copyright notice is

This PDF was created on 2020-02-16 20:48:26Z.