Name: Karthikeyan G
Roll No: CB.SC.P2CYS24008

Securing Coding Lab Experiment – 5
Set-UID

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.

passwd (Change User Password)

- **Set-UID Reason**: Modifies the system's /etc/shadow or /etc/passwd files, which store password information and are only writable by root.
- **Without Set-UID**: Users will not be able to change their own passwords because they lack the permissions to modify these sensitive files.

chsh (Change User's Default Shell)

- **Set-UID Reason**: Changes the login shell in /etc/passwd, which requires root privileges to modify.
- **Without Set-UID**: Users cannot change their login shell, as they don’t have write access to /etc/passwd.

su (Switch User)

- **Set-UID Reason**: Allows a user to switch to another user's account, including root, which requires authentication and privilege elevation.
- **Without Set-UID**: Users won't be able to switch to other users, particularly root, because the command won’t have the necessary permissions to create new processes as other users.

sudo (Execute Command as Another User, Typically Root)

- **Set-UID Reason**: Temporarily grants users root privileges to run specific commands.
- **Without Set-UID**: The command will fail to provide elevated access, as it cannot escalate the privileges required to execute other commands as root.

Set-UID allows commands like passwd, chsh, su, and sudo to run with root privileges, enabling necessary system changes securely. By using the Set-UID mechanism, Linux ensures that these programs can be safely executed by non-privileged users while still maintaining the system’s security and integrity.

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:

```
┌──(root㉿kali)-[/home/karthikeyan]
└─# cp /bin/zsh /tmp/

┌──(root㉿kali)-[/home/karthikeyan]
└─# chmod 4755 /tmp/zsh

┌──(root㉿kali)-[/home/karthikeyan]
└─# su karthikeyan
┌──(karthikeyan㉿kali)-[~]
└─$ /tmp/zsh
kali#
```

```
┌──(karthikeyan㉿kali)-[~]
└─$ ls -l /tmp/zsh
-rwsr-xr-x 1 root root 882152 Sep 29 17:58 /tmp/zsh
```

running /tmp/zsh will give me root privileges because the Set-UID bit (4755) allows the shell to run with root privileges, even when executed by a normal user.

(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

```
┌──(karthikeyan㉿kali)-[~]
└─$ sudo -i
[sudo] password for karthikeyan:
┌──(root㉿kali)-[~]
└─# cp /bin/zsh /tmp/

┌──(root㉿kali)-[~]
└─# chmod 4755 /tmp/bash

┌──(root㉿kali)-[~]
└─# su karthikeyan
┌──(karthikeyan㉿kali)-[/root]
└─$ /tmp/bash
bash-5.2$
```

```
┌──(karthikeyan㉿kali)-[/root]
└─$ ls -l /tmp/bash
-rwsr-xr-x 1 root root 1277936 Sep 29 18:03 /tmp/bash
```

Observation:

When running /tmp/bash after setting the Set-UID bit (4755), the command executed successfully, but the user remained the same “karthikeyan” as verified by the whoami command. No root privileges were obtained.

3.(Setup for the rest of the tasks) As you can find out from the previous task, /bin/bash has certainbuilt-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 linkto /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

```
┌──(root㉿kali)-[/home/karthikeyan]
└─# cd /bin

┌──(root㉿kali)-[/bin]
└─# rm sh

┌──(root㉿kali)-[/bin]
└─# ln -s zsh sh

┌──(root㉿kali)-[/bin]
└─# ls -l |grep zsh
lrwxrwxrwx 1 root root              3 Sep 14 03:40 rzsh -> zsh
lrwxrwxrwx 1 root root              3 Sep 29 18:15 sh -> zsh
-rwxr-xr-x 1 root root         882152 Sep 14 03:40 zsh
-rwxr-xr-x 1 root root            852 Jan  9  2024 zsh5
```

Observation: After changing the default shell to /bin/zsh, Set-UID can now be tested with a shell that may not have the same built-in protections as bash.

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:

```
#include<stdio.h>
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.

```
┌──(root㉿kali)-[/home/karthikeyan]
└─# echo -e '#!/bin/bash|nwhoami' >/home/karthikeyan/ls

┌──(root㉿kali)-[/home/karthikeyan]
└─# chmod +x /home/karthikeyan/ls

┌──(root㉿kali)-[/home/karthikeyan]
└─#  export PATH=/home/karthikeyan:$PATH
```

```
┌──(root㉿kali)-[/home/karthikeyan/Desktop]
└─# gcc -o setuid setuid.c
setuid.c: In function ‘main’:
setuid.c:4:1: warning: implicit declaration of function ‘system’ [-Wimplicit-functi
on-declaration]
    4 | system("ls");
      | ^~~~~~

┌──(root㉿kali)-[/home/karthikeyan/Desktop]
└─# chmod u+s setuid


┌──(root㉿kali)-[/home/karthikeyan/Desktop]
└─# ./setuid
zsh:1: /home/karthikeyan/ls: bad interpreter: /bin/bash|nwhoami: no such file or di
rectory
setuid  setuid.c

┌──(root㉿kali)-[/home/karthikeyan/Desktop]
```

It showing no such file or directory

(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

```
┌──(root㉿kali)-[/home/karthikeyan/Desktop]
└─# sudo ln -sf /bin/bash /bin/sh

┌──(root㉿kali)-[/home/karthikeyan/Desktop]
└─# ls -l /bin/sh
zsh: /home/karthikeyan/ls: bad interpreter: /bin/bash|nwhoami: no such file or dire
ctory
lrwxrwxrwx 1 root root 9 Sep 29 18:59 /bin/sh -> /bin/bash

┌──(root㉿kali)-[/home/karthikeyan/Desktop]
└─# ./setuid
sh: line 1: /home/karthikeyan/ls: cannot execute: required file not found
```

Observation: After linking /bin/sh to /bin/bash, showing me like error.

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. Laboratory for Computer Security Education 3
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 ;
}
```

(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).

```
┌──(karthikeyan㉿kali)-[~/Desktop]
└─$ sudo su
[sudo] password for karthikeyan:
┌──(root㉿kali)-[/home/karthikeyan/Desktop]
└─# gcc bob.c
bob.c: In function ‘main’:
bob.c:19:6: warning: implicit declaration of function ‘execve’ [-Wimplicit-function-decla
ration]
   19 | else execve(v[0], v, 0);
      |      ^~~~~~

┌──(root㉿kali)-[/home/karthikeyan/Desktop]
└─# chmod u+s bob

┌──(root㉿kali)-[/home/karthikeyan/Desktop]
└─# ls -l |grep bob
-rwsr-xr-x 1 root root 17928 Sep 29 19:04 bob
-rw-r--r-- 1 root root   450 Sep 29 19:03 bob.c

┌──(root㉿kali)-[/home/karthikeyan/Desktop]
└─# touch file

┌──(root㉿kali)-[/home/karthikeyan/Desktop]
└─# ls -l |grep file
-rw-r--r-- 1 root root     0 Sep 29 19:08 file
```

```
┌──(root㉿kali)-[/home/karthikeyan/Desktop]
└─# ls -alps
total 68
 4 drwxr-xr-x  2 karthikeyan karthikeyan  4096 Sep 29 19:08 ./
 4 drwx------ 18 karthikeyan karthikeyan  4096 Sep 29 19:06 ../
16 -rwxr-xr-x  1 root        root        16208 Sep 29 19:08 a.out
 0 lrwxrwxrwx  1 root        root            9 Sep 29 19:01 bash -> /bin/bash
20 -rwsr-xr-x  1 root        root        17928 Sep 29 19:04 bob
 4 -rw-r--r--  1 root        root          450 Sep 29 19:03 bob.c
 0 -rw-r--r--  1 root        root            0 Sep 29 19:08 file
16 -rwsr-xr-x  1 root        root        16008 Sep 29 18:36 setuid
 4 -rw-r--r--  1 root        root          222 Sep 29 18:36 setuid.c
 0 lrwxrwxrwx  1 root        root            7 Sep 29 19:01 sh -> /bin/sh
```

Observation:

After setting q = 0 and using system (), i were unable to manipulate the command execution by passing a string that includes a shell command (file;mv file file_new).

(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.

```
┌──(root㉿kali)-[/home/karthikeyan/Desktop]
└─# gcc bob.c -o bob
bob.c: In function ‘main’:
bob.c:19:6: warning: implicit declaration of function ‘execve’ [-Wimplicit-function-declaration]
   19 | else execve(v[0], v, 0);
      |      ^~~~~

┌──(root㉿kali)-[/home/karthikeyan/Desktop]
└─# chmod u+s bob

┌──(root㉿kali)-[/home/karthikeyan/Desktop]
└─# exit

┌──(karthikeyan㉿kali)-[~/Desktop]
└─$ ./bob "file:my file file_new"
/bin/cat: 'file:my': No such file or directory
/bin/cat: file_new: No such file or directory

┌──(karthikeyan㉿kali)-[~/Desktop]
└─$ ls -l
total 56
-rwxr-xr-x 1 root root 16208 Sep 29 19:08 a.out
lrwxrwxrwx 1 root root     9 Sep 29 19:01 bash -> /bin/bash
-rwsr-xr-x 1 root root 16208 Sep 29 19:16 bob
-rw-r--r-- 1 root root   450 Sep 29 19:03 bob.c
-rw-r--r-- 1 root root     0 Sep 29 19:08 file
-rwsr-xr-x 1 root root 16008 Sep 29 18:36 setuid
-rw-r--r-- 1 root root   222 Sep 29 18:36 setuid.c
lrwxrwxrwx 1 root root     7 Sep 29 19:01 sh -> /bin/sh
```

Observation:

When q = 1, the program uses execve() instead of system(), which directly invokes /bin/cat without invoking a shell. Therefore, the string file;mv file file_new is not interpreted as two separate commands.

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 `, not one; in the -lc argment, the second character is `):

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

```
┌──(karthikeyan㉿kali)-[~/Desktop/lab5]
└─$ ls-1
ls-1: command not found

┌──(karthikeyan㉿kali)-[~/Desktop/lab5]
└─$ ls -l
total 8
-rw-rw-r-- 1 karthikeyan karthikeyan 74 Sep 29 19:22 mylib.c
-rw-rw-r-- 1 karthikeyan karthikeyan 68 Sep 29 19:24 myprog.c

┌──(karthikeyan㉿kali)-[~/Desktop/lab5]
└─$ gcc -fPIC -g -c mylib.c

┌──(karthikeyan㉿kali)-[~/Desktop/lab5]
└─$ gcc -shared -Wl,-soname,libmylib.so.1 \
-o libmylib.so.1.0.1 mylib.o -lc

┌──(karthikeyan㉿kali)-[~/Desktop/lab5]
└─$ ls -l
total 32
-rwxrwxr-x 1 karthikeyan karthikeyan 16400 Sep 29 19:25 libmylib.so.1.0.1
-rw-rw-r-- 1 karthikeyan karthikeyan    74 Sep 29 19:22 mylib.c
-rw-rw-r-- 1 karthikeyan karthikeyan  3320 Sep 29 19:24 mylib.o
-rw-rw-r-- 1 karthikeyan karthikeyan    68 Sep 29 19:24 myprog.c
```

(c) Now, set the LD PRELOAD environment variable:

export LD_PRELOAD=./libmylib.so.1.0.1

```
┌──(karthikeyan㉿kali)-[~/Desktop/lab5]
└─$ export LD_PRELOAD=./libmylib.so.1.0.1

┌──(karthikeyan㉿kali)-[~/Desktop/lab5]
└─$ echo $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):

```c
/* 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).

Make myprog a Regular Program, Run as a Normal User:

```
┌──(karthikeyan㉿kali)-[~/Desktop/lab5]
└─$ gcc -g -o myprog myprog.c
myprog.c: In function ‘main’:
myprog.c:5:1: warning: implicit declaration of function ‘sleep’ [-Wimplicit-function-declaration]
    5 | sleep(1);
      | ^~~~~

┌──(karthikeyan㉿kali)-[~/Desktop/lab5]
└─$ ./myprog
I am not sleeping!
```

Since myprog is not a Set-UID program, LD_PRELOAD works normally, and the dynamic library is loaded as expected.

Make myprog a Set-UID Root Program, Run as a Normal User:

```
┌──(root㉿kali)-[/home/karthikeyan/Desktop/lab5]
└─# gcc -g -o myprog myprog.c
myprog.c: In function ‘main’:
myprog.c:5:1: warning: implicit declaration of function ‘sleep’ [-Wimplicit-function-declaration]
    5 | sleep(1);
      | ^~~~~

┌──(root㉿kali)-[/home/karthikeyan/Desktop/lab5]
└─# chmod u+s myprog

┌──(root㉿kali)-[/home/karthikeyan/Desktop/lab5]
└─# exit

┌──(karthikeyan㉿kali)-[~/Desktop/lab5]
└─$ ls -l
total 52
-rwxrwxr-x 1 karthikeyan karthikeyan 16400 Sep 29 19:25 libmylib.so.1.0.1
-rw-rw-r-- 1 karthikeyan karthikeyan    74 Sep 29 19:22 mylib.c
-rw-rw-r-- 1 karthikeyan karthikeyan  3320 Sep 29 19:24 mylib.o
-rwsr-xr-x 1 root        root        17032 Sep 29 19:31 myprog
-rw-rw-r-- 1 karthikeyan karthikeyan    68 Sep 29 19:24 myprog.c
```

The program runs without using the overridden sleep() function. No custom message is printed. When a Set-UID root program is executed, the runtime linker (ld.so) ignores the LD_PRELOAD environment variable for security reasons, as allowing arbitrary code to be loaded would be a major security risk.

Make myprog a Set-UID Root Program, Run as root:

```
┌──(root㉿kali)-[/home/karthikeyan/Desktop/lab5]
└─# gcc -g -o myprog myprog.c
myprog.c: In function ‘main’:
myprog.c:5:1: warning: implicit declaration of function ‘sleep’ [-Wimplicit-function-declaration]
    5 | sleep(1);
      | ^~~~~

┌──(root㉿kali)-[/home/karthikeyan/Desktop/lab5]
└─# chmod 4755 myprog

┌──(root㉿kali)-[/home/karthikeyan/Desktop/lab5]
└─# ./myprog
I am not sleeping!
```

Make myprog **a Set-UID** user1 **Program, Run as Another User**:

```
┌──(karthikeyan㉿kali)-[~/Desktop/lab5]
└─$ sudo chown cys24008 myprog

┌──(karthikeyan㉿kali)-[~/Desktop/lab5]
└─$ sudo chmod u+s myprog

┌──(karthikeyan㉿kali)-[~/Desktop/lab5]
└─$ su cys24008
Password:
┌──(cys24008㉿kali)-[/home/karthikeyan/Desktop/lab5]
└─$ ./myprog

┌──(cys24008㉿kali)-[/home/karthikeyan/Desktop/lab5]
└─$
```

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;
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);
}
}
```

```
┌──(root㉿kali)-[/home/karthikeyan/Desktop/lab5]
└─# vim test.c

zsh: suspended  vim test.c

┌──(root㉿kali)-[/home/karthikeyan/Desktop/lab5]
└─# ls -1
libmylib.so.1.0.1
mylib.c
mylib.o
myprog
myprog.c
test
test.c

┌──(root㉿kali)-[/home/karthikeyan/Desktop/lab5]
└─# gcc -o test test.c

┌──(root㉿kali)-[/home/karthikeyan/Desktop/lab5]
└─# ls -l
total 72
-rwxrwxr-x 1 karthikeyan karthikeyan 16400 Sep 29 19:25 libmylib.so.1.0.1
-rw-rw-r-- 1 karthikeyan karthikeyan    74 Sep 29 19:22 mylib.c
-rw-rw-r-- 1 karthikeyan karthikeyan  3320 Sep 29 19:24 mylib.o
-rwsr-xr-x 1 cys24008    root        17032 Sep 29 19:35 myprog
-rw-rw-r-- 1 karthikeyan karthikeyan    68 Sep 29 19:24 myprog.c
-rwxr-xr-x 1 root        root        16304 Sep 29 20:49 test
-rw-r--r-- 1 root        root         1159 Sep 29 20:42 test.c
```

```
┌──(root㉿kali)-[/home/karthikeyan/Desktop/lab5]
└─# gcc test.c -o test

┌──(root㉿kali)-[/home/karthikeyan/Desktop/lab5]
└─# chmod 4755 test

┌──(root㉿kali)-[/home/karthikeyan/Desktop/lab5]
└─# ./test
Child process running, but cannot modify /etc/zzz
```

Even after dropping root privileges using setuid(), the child process cannot write to the file /etc/zzz and cannot modified