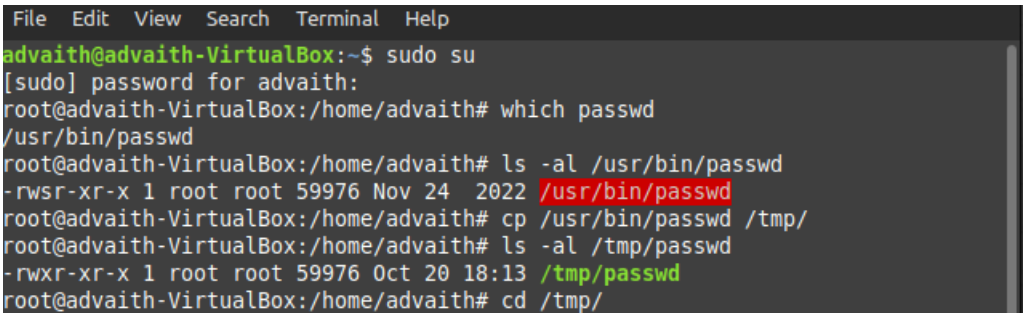
Security Risks of Set-UID Programs in Unix-Based Systems

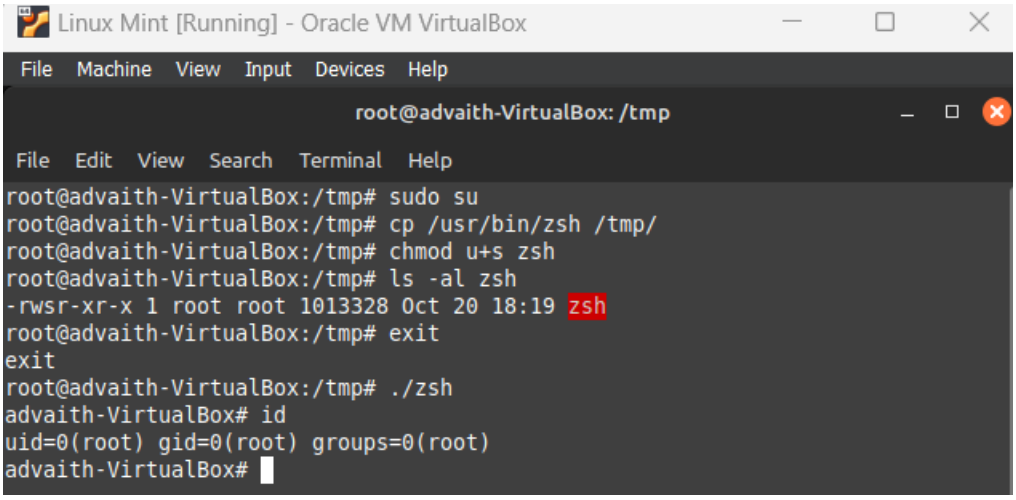
We will figure out why "passwd", "chsh", "su", and "sudo" commands need to be Set-UID programs. What will happen if they are not?



We find that when copying passwd to /tmp/,it lost root's privileges. As for chsh, su and sudo, they are the same.

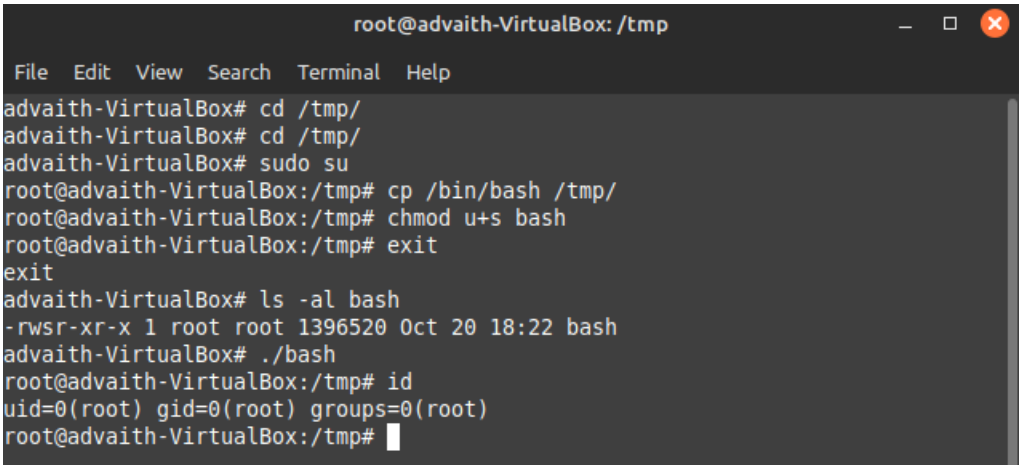
2.. Run Set-UID shell programs in Linux, and describe and explain your observations.

(a) Login as root, copy /bin/zsh to /rmp, 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.



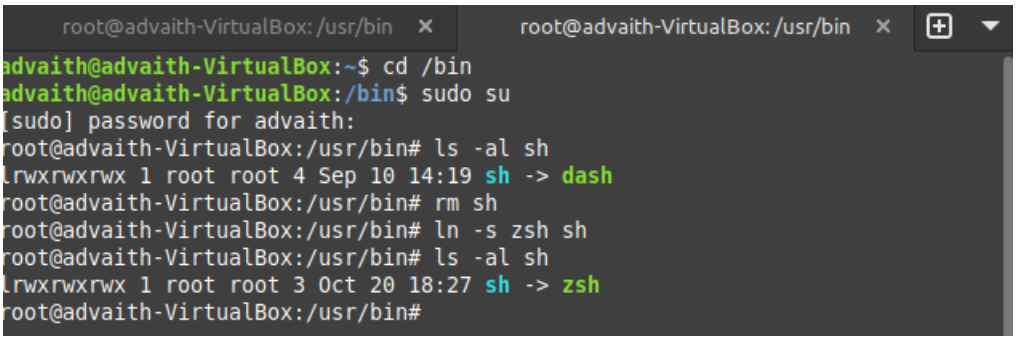
So now normal user get root privilege.

(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.



Since we do the same operating, zsh can get root privilege, but bash can't.

3. 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.

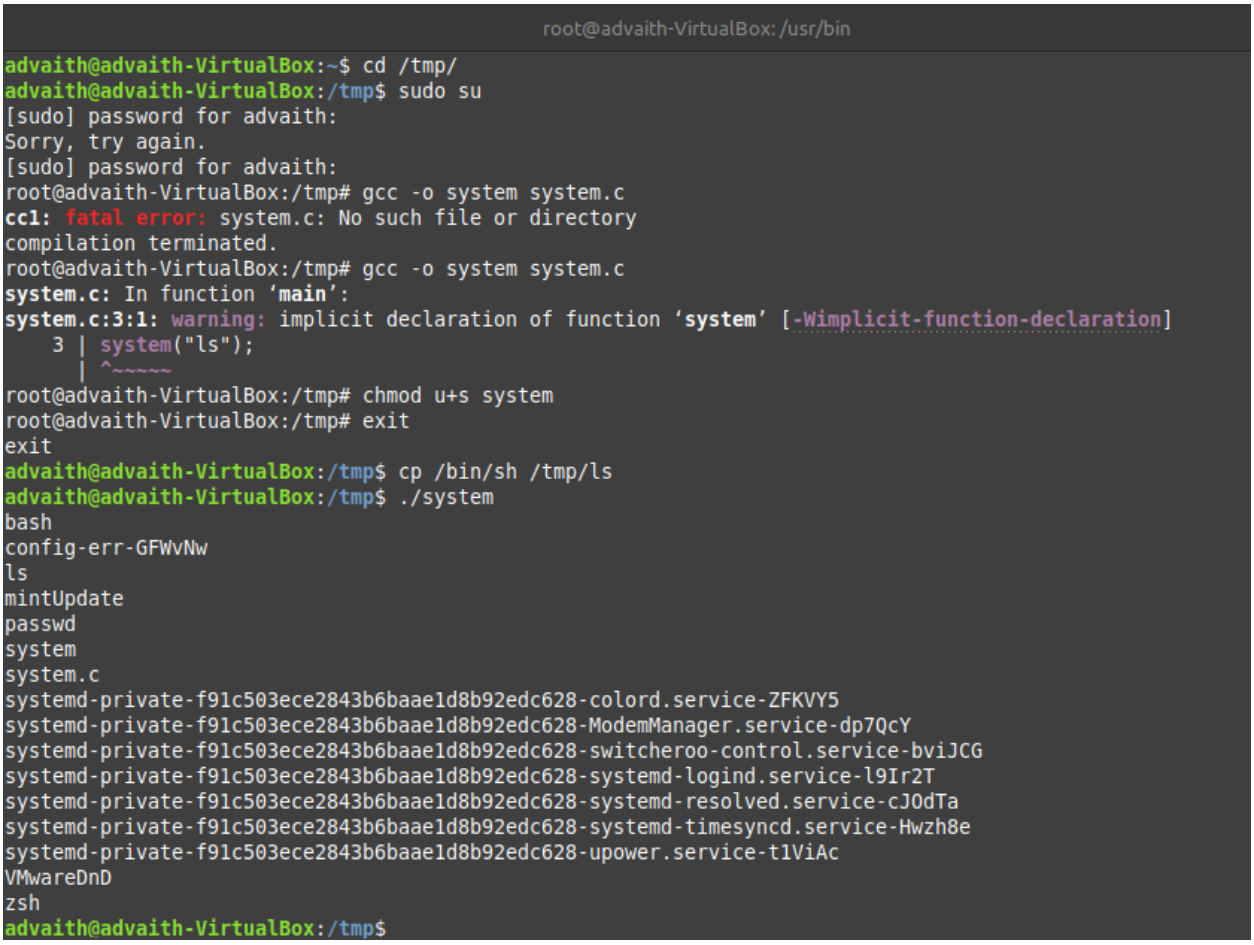


4.. The PATH environment variable. The system(const char *cmd) library function can be used to execute a command within a program. The way system(cmd) works is to invoke the /bin/sh program, and then let the shell program to execute cmd. Because of the shell program invoked, calling system() within a 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. 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

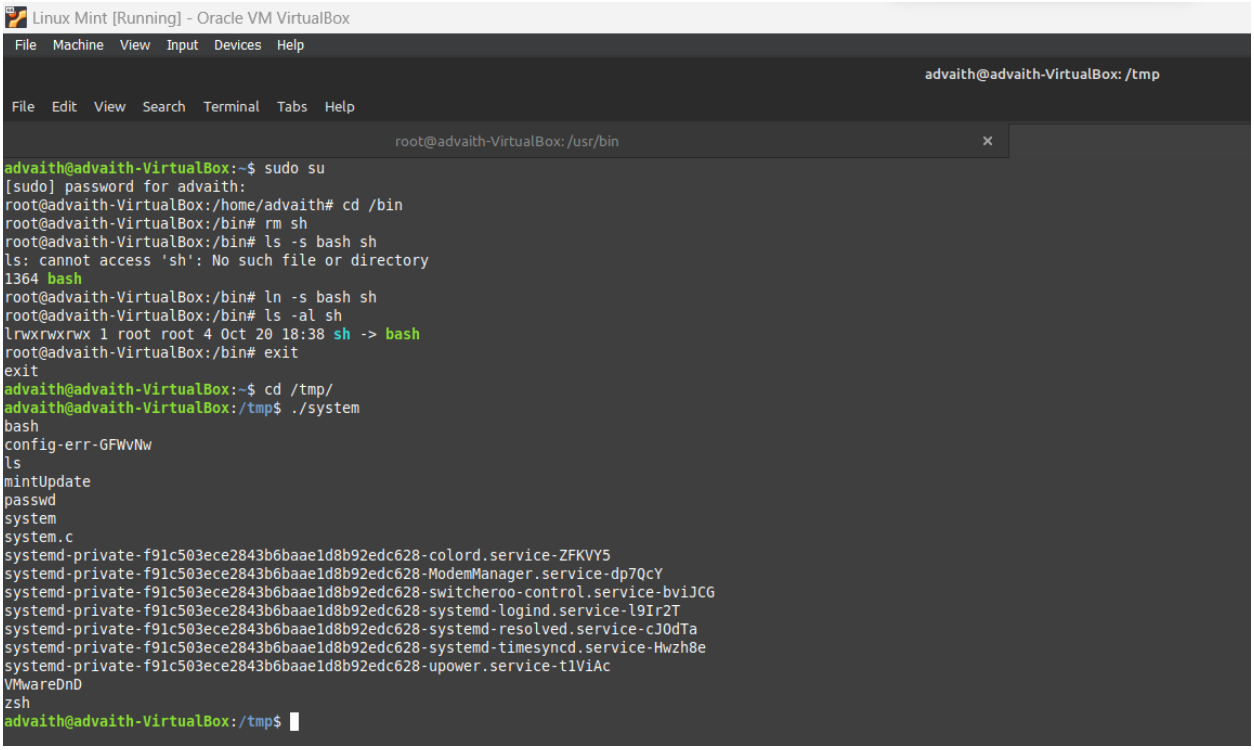
```
int main()
{
    system("ls");
    return 0;
}
```

(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.:



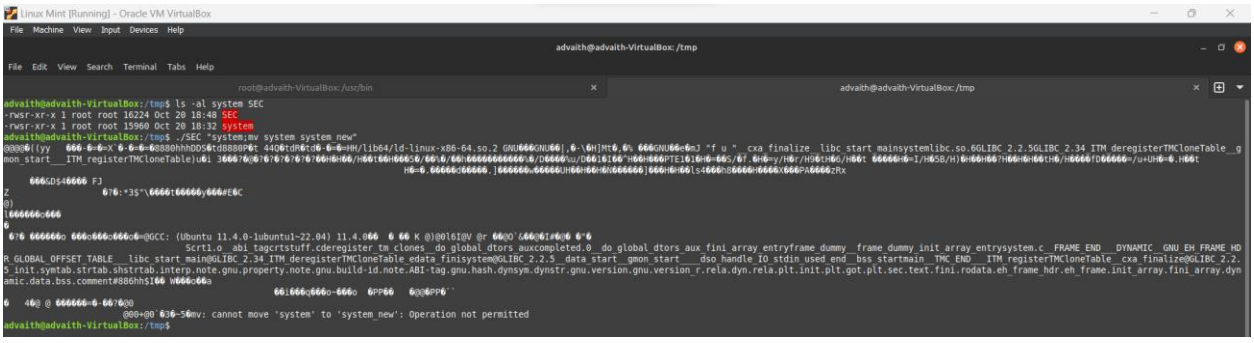
It can have root previlege, copy /bin/sh to /tmp with new name ls.(make sure sh -> zsh). Then set PATH to current directory /tmp, compile and run system program and we will get root previlege.

(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.



We can't get root privilege.

5.. The difference between system() and execve(). Before you work on this task, please make sure that /bin/sh is point to /bin/zsh. Background: Bob works for an auditing agency, and he needs to investigete 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.



The SEC file is not safe, Bob can read, write or move files which only root user can run.

(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.

```
advaith@advaith-VirtualBox:/tmp$ sudo su
root@advaith-VirtualBox:/tmp# gcc -o SEC2 SEC.c
SEC.c: In function ‘main’:
SEC.c:21:8: warning: implicit declaration of function ‘execve’ [-Wimplicit-function-declaration]
   21 |   else execve(v[0], v, 0);
      |        ^~~~~
root@advaith-VirtualBox:/tmp# chmod u+s SEC2
root@advaith-VirtualBox:/tmp# ./SEC2 "system;mv system system_new2
> "
/bin/cat: 'system;mv system system_new2'$'\n': No such file or directory
root@advaith-VirtualBox:/tmp# ls system*
system  system.c

systemd-private-f91c503ece2843b6baae1d8b92edc628-colord.service-ZFKVY5:
tmp

systemd-private-f91c503ece2843b6baae1d8b92edc628-ModemManager.service-dp7QcY:
tmp

systemd-private-f91c503ece2843b6baae1d8b92edc628-switcheroo-control.service-bviJCG:
tmp

systemd-private-f91c503ece2843b6baae1d8b92edc628-systemd-logind.service-l9Ir2T:
tmp

systemd-private-f91c503ece2843b6baae1d8b92edc628-systemd-resolved.service-cJOdTa:
tmp

systemd-private-f91c503ece2843b6baae1d8b92edc628-systemd-timesyncd.service-Hwzh8e:
tmp

systemd-private-f91c503ece2843b6baae1d8b92edc628-upower.service-t1ViAc:
tmp
root@advaith-VirtualBox:/tmp#
```

When modify q to 1, the attack can't make sense. The reason why the before attack effectively is becausesystem() call /bin/sh, which links zsh. After running cat file with root privilege, it runs mv file file_new. But when q = 1, execve() will regard file;mv file file_new2 as a folder name, so system will prompt there have no the file.

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 buid a dynamic link library. Create the following program, and name it mylib.c. It basically overrides the sleep() function in libc:

```c
#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 argument, 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):

```c
int main()
{
   sleep(1);
   return 0;
}
```

// myprog.c

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.

<1> Make myprog a regular program, and run it as a normal user.

```
seed@ubuntu:/tmp$ export LD_PRELOAD=./libmylib.so.1.0.1
seed@ubuntu:/tmp$ echo $LD_PRELOAD
./libmylib.so.1.0.1
seed@ubuntu:/tmp$ gcc -o myprog myprog.c
seed@ubuntu:/tmp$ ./myprog
```

```
I am not sleeping!
```

<2> Make myprog a Set-UID root program, and run it as a normal user.

```
seed@ubuntu:/tmp# sudo su
root@ubuntu:/tmp# export LD_PRELOAD=./libmylib.so.1.0.1
root@ubuntu:/tmp# gcc -o myprog myprog.c
root@ubuntu:/tmp# chmod u+s myprog
root@ubuntu:/tmp# exit
exit
seed@ubuntu:/tmp$ ./myprog
seed@ubuntu:/tmp$
```

In this situation, it will ignore LD_PRELOAD environment variable and use the system's default sleep() function. So sleep() function will not be overrided.

<3> Make myprog a Set-UID root program, and run it in the root account.

```
seed@ubuntu:/tmp$ sudo su
root@ubuntu:/tmp# export LD_PRELOAD=./libmylib.so.1.0.1
root@ubuntu:/tmp# gcc -o myprog myprog.c
root@ubuntu:/tmp# chmod u+s myprog
root@ubuntu:/tmp# ./myprog
I am not sleeping!
```

In this situation, it will use LD_PRELOAD environment variable and override sleep() function.

<4> 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).

```
seed@ubuntu:/tmp$ sudo su
root@ubuntu:/tmp# useradd -d /usr/user1 -m user1
root@ubuntu:/tmp# su user1
$ export LD_PRELOAD=./libmylib.so.1.0.1
$ gcc -o myprog myprog.c
$ chmod u+s myprog
$ su seed
Password:
seed@ubuntu:/tmp$ ./myprog
```

```
seed@ubuntu:/tmp$
```

In this situation, it will not override sleep() function.

From the four formal situation, we know that only a user run the program created by himself, LD_PRELOAD environment valiable can be used and sleep() function can be overrided.

7.. Relinquishing privileges and cleanup. To be more secure, Set-UID programs usually call setuid() system call to permancently 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", 14);
        close(fd);
      }
}
```

The result:

```
seed@ubuntu:/etc$ sudo su
root@ubuntu:/etc# gcc -o test test.c
root@ubuntu:/etc# chmod u+s test
root@ubuntu:/etc# exit
exit
seed@ubuntu:/etc$ ls -al zzz test
-rwsr-xr-x 1 root root 7453 Aug  8 07:49 test
-rw-r--r-- 1 root root   51 Aug  8 07:47 zzz
seed@ubuntu:/etc$ ./test
seed@ubuntu:/etc$ cat zzz
Malicious Data
```