Name : Varada Gurjar **Course : 225** **Date : 2 Feb 2015**

5. (15 points) The difference between system() and execve().

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).

Answer 5 a)

Consider a root-access file containing secure data in /bin

root@seed-desktop:/bin# **cat /bin/SecureFile.txt**

THIS IS A RESTRICTED ACCESS FILE. DO NOT DUPLICATE

This file should be viewable by Bob (using cat) but not duplicated , say to his local folder using the boundary program given in the question 5.
However invoking the command as below (here Bob is 'seed') ,

seed@seed-desktop:/bin$ **./entry_program "/bin/SecureFile.txt; cp /bin/SecureFile.txt /home/seed/Desktop/"**

Bob is able to

- view the /bin/SecureFile.txt
- copy the Secure file to his desktop

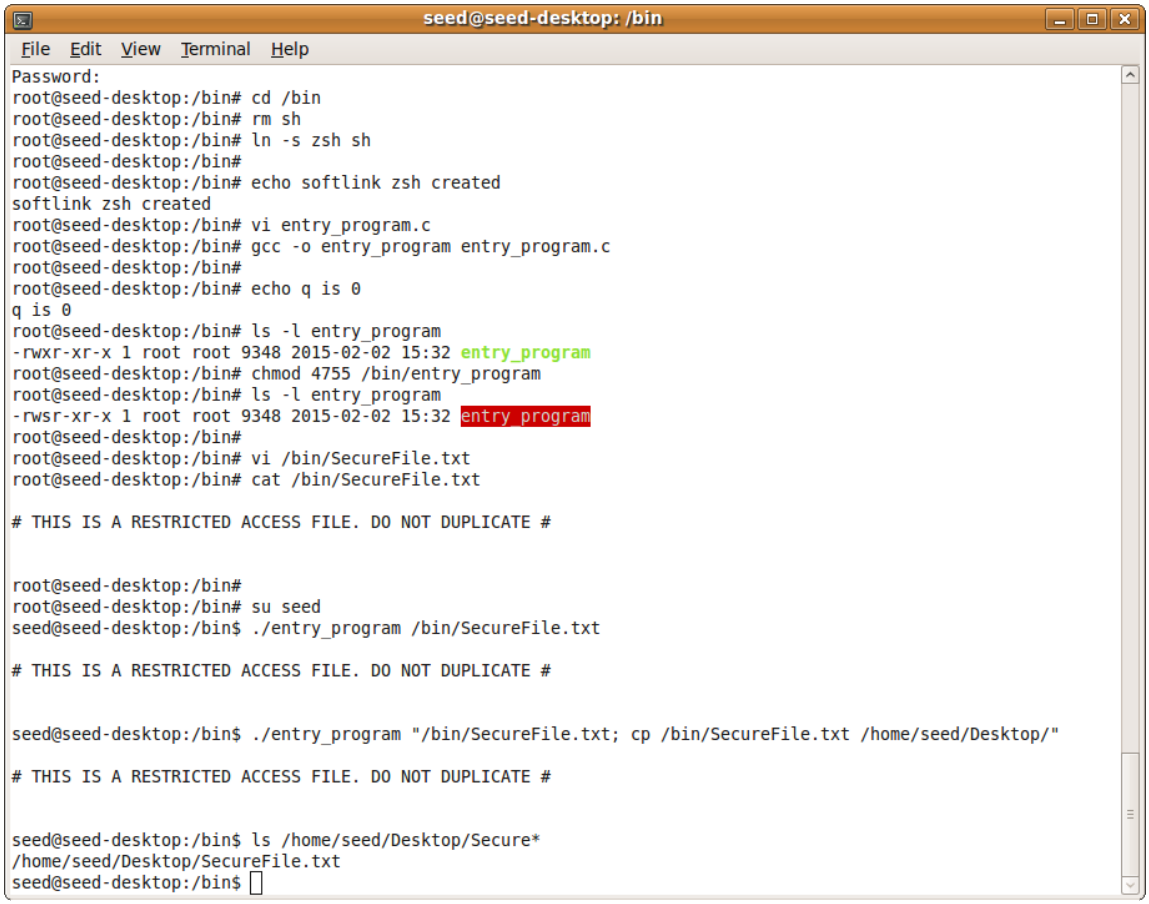
by invoking two commands in system(command) **using the special character ;**

The reason the special character ; works to execute two commands in system is because system() function invokes the command passed to it from a separate shell . Since the boundary company program Bob accesses is a Set-UID root program and the ; semicolon separated commands run from a different shell, it is like running cp <securefile> <local folderpath> from a seed privilege shell.

Hence Bob is able to compromise the integrity of the system using special characters.

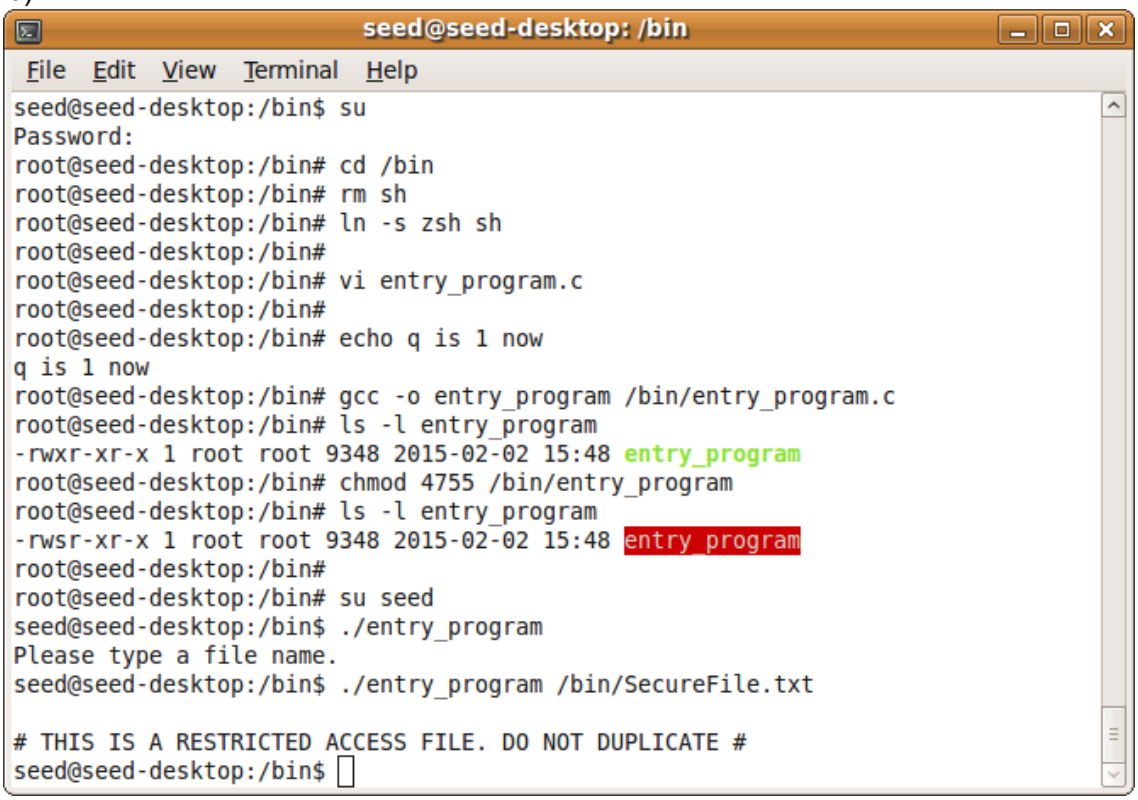
```
seed@seed-desktop:/bin$ su
Password:
root@seed-desktop:/bin# cd /bin
root@seed-desktop:/bin# rm sh
root@seed-desktop:/bin# ln -s zsh sh
root@seed-desktop:/bin#
root@seed-desktop:/bin# echo softlink zsh created
softlink zsh created
root@seed-desktop:/bin# vi entry_program.c
root@seed-desktop:/bin# gcc -o entry_program entry_program.c          //The boundary program Bob
root@seed-desktop:/bin#                                               //accesses is entry_program.c
root@seed-desktop:/bin# echo q is 0
q is 0
root@seed-desktop:/bin# ls -l entry_program
-rwxr-xr-x 1 root root 9348 2015-02-02 15:32 entry_program
root@seed-desktop:/bin# chmod 4755 /bin/entry_program
root@seed-desktop:/bin# ls -l entry_program
-rwsr-xr-x 1 root root 9348 2015-02-02 15:32 entry_program     //entry_program is Set-UID root program
root@seed-desktop:/bin#
root@seed-desktop:/bin# vi /bin/SecureFile.txt
root@seed-desktop:/bin# su seed
seed@seed-desktop:/bin$ ./entry_program "/bin/SecureFile.txt; cp /bin/SecureFile.txt /home/seed/Desktop/"

# THIS IS A RESTRICTED ACCESS FILE. DO NOT DUPLICATE #
```

seed@seed-desktop:/bin$ **ls /home/seed/Desktop/Secure***
/home/seed/Desktop/SecureFile.txt **// Bob has successfully viewed and duplicated**
seed@seed-desktop:/bin$ **// the Secure file.**



5 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.

Answer 5 b)



Setting q=1 in the code of the entry_program the call is made to the execve() function.

int execve(const char *path, char *const argv[], char *const envp[]);

The execve() loads and executes a new program from within the entry_program. Unlike system() here , it does not open new shell with 'seed' privilege and run the command.
execve() loads the new program at the 'path' with the 'argv' as input for the new program . It loads the new program from current program. It does not consider any special character delimiter ,like ; as in the previous case.
Incase execve (/bin/cat , <filename> ; <rootcommand>, 0) is invoked, it will try to call the /bin/cat program passing the whole " <filename> ; <rootcommand>" as an argument.
The /bin/cat will echo no file found.

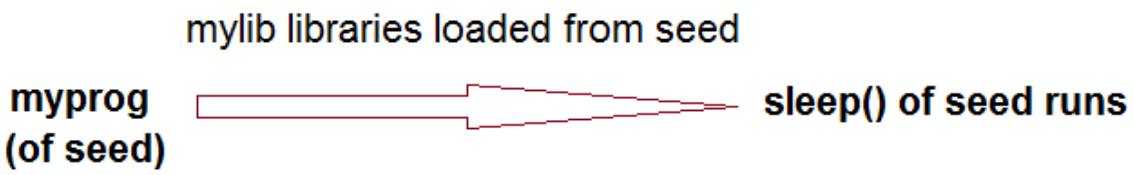
As execve ignores ; the previous attack will not work in the case of q = 1 case.

---

6. 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.

Case 1)

```
root@seed-desktop:/bin# su seed
seed@seed-desktop:/bin$
seed@seed-desktop:/bin$ echo CASE 1
CASE 1
seed@seed-desktop:/bin$ cd /home/seed
seed@seed-desktop:~$ pwd
/home/seed
seed@seed-desktop:~$ vi myprog.c                    //Regular program as a normal user
seed@seed-desktop:~$ gcc -o myprog myprog.c
seed@seed-desktop:~$
seed@seed-desktop:~$ vi 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:~$
seed@seed-desktop:~$ export LD_PRELOAD=./libmylib.so.1.0.1
seed@seed-desktop:~$
seed@seed-desktop:~$ ./myprog
I am not sleeping!
```

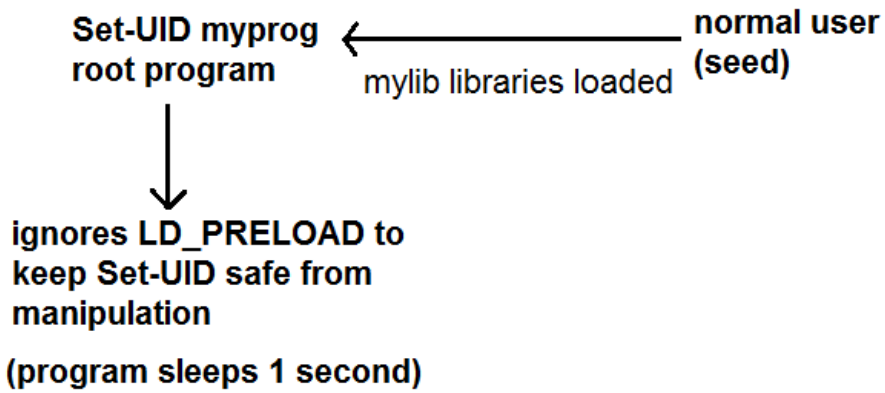


The 'seed' user creates myprog program, loads mylib libraries runs myprog. Local sleep() version runs.
Hence the output of myprog is " I am not sleeping".

Case 2)

```
root@seed-desktop:/home/seed# echo CASE 2
CASE 2
root@seed-desktop:/home/seed# cd /bin
root@seed-desktop:/bin# vi myprog.c
```

```
root@seed-desktop:/bin# gcc -o myprog /bin/myprog.c
root@seed-desktop:/bin# ls -l /bin/myprog
-rwxr-xr-x 1 root root 9147 2015-02-02 16:10 /bin/myprog          // myprog is Set-UID root program
root@seed-desktop:/bin# chmod 4755 /bin/myprog                    // run by normal user
root@seed-desktop:/bin# ls -l /bin/myprog
-rwsr-xr-x 1 root root 9147 2015-02-02 16:10 /bin/myprog
root@seed-desktop:/bin# su seed
seed@seed-desktop:/bin$ export LD_PRELOAD=./libmylib.so.1.0.1
seed@seed-desktop:/bin$ ./myprog
seed@seed-desktop:/bin$                                           // program sleeps 1 second.
```



Set-UID myprog of the root is called by normal user 'seed' . The local mylib libraries of user 'seed' are loaded. However Set-UID mechanism ignores the LD_PRELOAD environment variable value to be safe from manipulation and running of normal user libraries.

Hence the program does sleep of one second.

Case 3)

```
root@seed-desktop:/home/seed# echo CASE 3
CASE 3
root@seed-desktop:/home/seed# ls -l /bin/myprog
-rwsr-xr-x 1 root root 9147 2015-02-02 16:10 /bin/myprog          // myprog is Set-UID root program
root@seed-desktop:/home/seed# export LD_PRELOAD=./libmylib.so.1.0.1   //run by root itself
root@seed-desktop:/home/seed#
root@seed-desktop:/home/seed# /bin/myprog
I am not sleeping!
```

Root creates Set-UID program myprog .Root loads normal user libraries mylib and runs myprog. As real user id of the user running my prog is the same as root, there is no attempt to manipulate the Set-UID mechanism. Root runs the mylib and not a normal user(seed) trying to manipulate the Set-UID mechanism.

Hence the LD_PRELOAD variable is not ignored and the mylib library runs the printf().

Case 4)

```
root@seed-desktop:/home/seed# echo CASE 4
CASE 4
root@seed-desktop:/home/seed# sudo adduser varada
Adding user `varada' ...
Adding new group `varada' (1002) ...
Adding new user `varada' (1002) with group `varada' ...
Creating home directory `/home/varada' ...
Copying files from `/etc/skel' ...
```

```
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for varada
Enter the new value, or press ENTER for the default
    Full Name []:
    Room Number []:
    Work Phone []:
    Home Phone []:
    Other []:
Is the information correct? [Y/n] y
root@seed-desktop:/home/seed# rm /bin/myprog*                    // user1 here is varada, added.
root@seed-desktop:/home/seed# su varada
varada@seed-desktop:/home/seed$
varada@seed-desktop:/home/seed$ cd /home/varada/
varada@seed-desktop:~$ vi myprog.c
ERROR: ld.so: object './libmylib.so.1.0.1' from LD_PRELOAD cannot be
preloaded: ignored.
varada@seed-desktop:~$ vi myprog.c                    //  'varada' creates myprog Set-UID
varada@seed-desktop:~$ gcc -o myprog /home/varada/myprog.c          // program . 'seed' runs it
varada@seed-desktop:~$ ls -l /home/varada/myprog
-rwxr-xr-x 1 varada varada 9147 2015-02-02 16:32 /home/varada/myprog
varada@seed-desktop:~$ chmod 4755 /home/varada/myprog
varada@seed-desktop:~$ ls -l /home/varada/myprog
-rwsr-xr-x 1 varada varada 9147 2015-02-02 16:32 /home/varada/myprog
varada@seed-desktop:~$
varada@seed-desktop:~$ su seed
Password:
seed@seed-desktop:/home/varada$ export LD_PRELOAD=./libmylib.so.1.0.1
seed@seed-desktop:/home/varada$ /home/varada/myprog
seed@seed-desktop:/home/varada$                    //program sleeps 1 second
```

Set-UID myprog of 'varada' is called by normal user 'seed' . The local mylib libraries of user 'seed' are loaded. However Set-UID mechanism ignores the LD_PRELOAD environment variable value to be safe from manipulation and running of normal user libraries.

Hence the program does sleep of one second.

7.

Answer 7)

Consider name of the program in question 7 as 'rootfile' being run by a normal user 'seed'.
In rootfile the /etc/zzz file of the root is opened. This has been done with root privileges as rootfile is Set-UID allowing root privilege to 'seed'.

Then the **setuid()** function is called. It is checked that the effective user id is that of root, and sets the getuid() or the real user id (of seed) to all three process ids. **In other words setuid(getuid) drops privileges of seed from root to a normal user . The further commands will run with seed privilege.**

The fork() function is invoked with 'seed' privilege and both the parent and child processes will run with seed privilege. Depending now on the order of execution either 'Malicious Data'is written into /etc/zzz or not.

I modified the program in question 7 to print pid and ppid of the processes. This will also help determine order of parent and child program execution after fork.

```
main()
{
……
…….
int pid;                                    // To view the process id of parent and child.
pid = fork();
     if (pid) { /* In the parent process */
           printf ( "Parent : Parent's PID: %d\n", getpid());
           printf ( "Parent : Child's PID: %d\n", pid);
       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 */
       printf ( "Child : Child's PID: %d\n", getpid());
        printf ( "Child : Parent's PID: %d\n", getppid());
……..
}
```

This rootfile is run by seed user. Commands as below :

```
root@seed-desktop:/bin# whoami
root
root@seed-desktop:/bin# vi rootfile.c
root@seed-desktop:/bin# gcc -o rootfile rootfile.c
root@seed-desktop:/bin# ls -l /bin/rootfile
-rwxr-xr-x 1 root root 9522 2015-02-02 15:06 /bin/rootfile
root@seed-desktop:/bin# chmod 4755 /bin/rootfile
root@seed-desktop:/bin# ls -l /bin/rootfile                    // root creates rootfile and makes it Set-UID
-rwsr-xr-x 1 root root 9522 2015-02-02 15:06 /bin/rootfile
root@seed-desktop:/bin#
root@seed-desktop:/bin# cd /etc                                // root creates /etc/zzz . zzz is a text file.
root@seed-desktop:/etc# vi zzz                                 //zzz contains "THIS IS AN IMPORTANT FILE.
root@seed-desktop:/etc# chmod 0644 /etc/zzz.txt                // SECURE DATA HERE.."
root@seed-desktop:/etc# su seed
seed@seed-desktop:/etc$ cd /bin
seed@seed-desktop:/bin$ ./rootfile
Child : Child's PID: 18041
Child : Parent's PID: 18040                                    // childs pid and ppid printf
Parent : Parent's PID: 18040
Parent : Child's PID: 18041                                    //parents pid printed
seed@seed-desktop:/bin$ su
Password:
root@seed-desktop:/bin# cat /etc/zzz
# THIS IS AN IMPORTANT FILE. SECURE DATA HERE...


Malicious Data                                                 //malicious data added to zzz by child process
root@seed-desktop:/bin#
```

Observations :

- **As noted from the printf outputs, both parent and child processes execute one after the other. The child process executes first modifying the /etc/zzz file.** If the parent had executed first the /etc/zzz file would have been closed and the rootfile program would exit.

- **It can be observed that both parent and child processes run. However to change order of execution , one process can be made to sleep() making the other process run.**

- **The pid gives process id and parent process id. pid of the parent is same as the ppid of child.** This is also seen in the output.

- fork() returns 0 to child process, and child process id to the parent process.