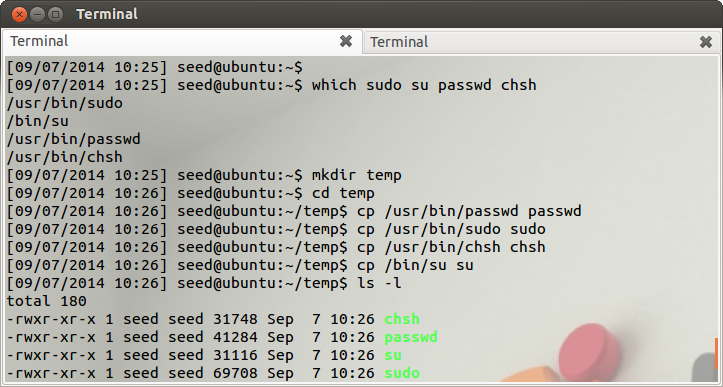
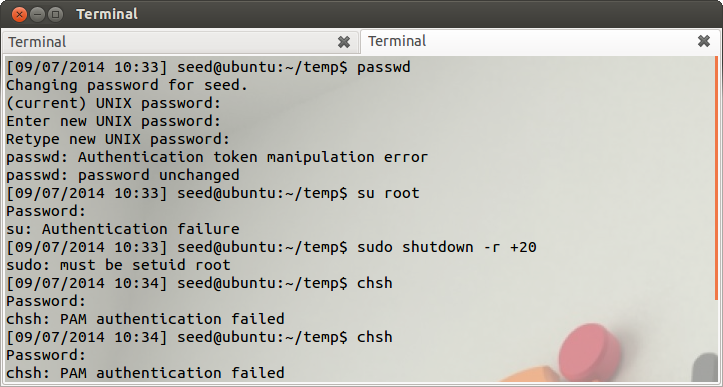
**Set-UID Program Vulnerability Lab Report**

Ankit Kumar

­

**Task 1:** Understanding “passwd”, “chsh”, “su” and “sudo”

****

****

**Observation:**

1.

* 1. passwd is a program or tool on UNIX operating systems that is used to change the user’s password or to change a local accounts password.

1.2. chsh is short form for change shell, it is used to change a login shell. This program modifies

the /etc/passwd file and only allows users to modify their own login shells

1.3. su is short for substitute user, it is used to change the login session’s owner. Su can change

the ownership of a session to any owner but is mostly use to change the owner to root, to

gain access to all commands and files of a system.

1.4. sudo is short for substitute user do, it allows the substituted user to execute commands as  
 the owner of that session, the real and effective uid and gid are set and made to match with

that of the target user in the passwd file. Authorized users are determined by referring   
 to /etc/sudoers file.

1. A Set-UID program is a privileged program as it allows users to gain root privilege when programs are being executed.

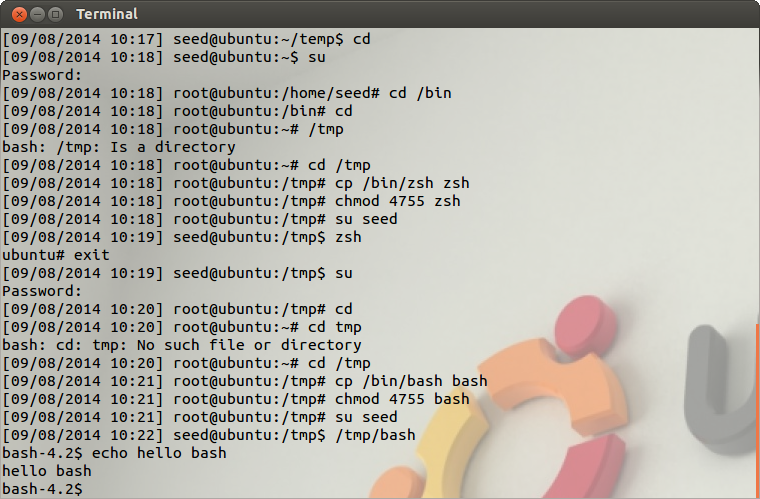
These command need to modify etc/passwd, etc/shadow and other such system config files. These files do not allow the normal user to access or modify them, only the root user is allowed to perform these actions. passwd, chsh, su and sudo command need to be set with SUID to give root permissions to normal user to allow updating of the protected files. su needs to be root to allow the changing of users on a system, chsh has to change the passwd file which is only writable by root, passwd needs to modify the etc/passwd file which needs root access and sudo needs to access /etc/sudoers file to allow substitute user to run a program with that users permissions.

1. These files are located in the following locations:
   1. passwd - /usr/bin
   2. sudo - /usr/bin
   3. su - /bin
   4. chsh - /usr/bin

**Explanation:**

Copying the programs to the folder “temp” is successful but on running these commands from the present working directory (pwd). Authentication errors are displayed for all of the commands.

**Task 2:**



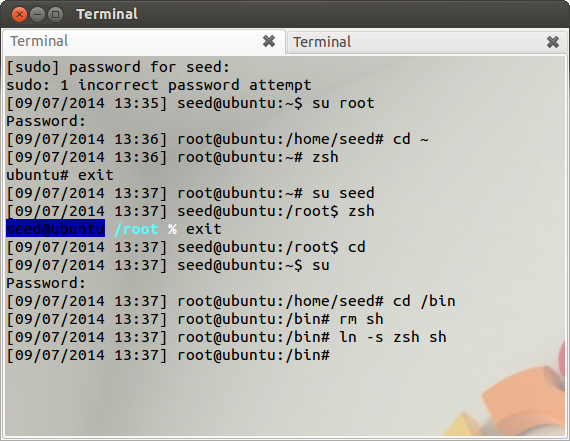
**Observation:**

1. On copying zsh from the bin directory to the tmp directory and making it a setuid program and then executing zsh from the tmp directory, root privileges are allowed. We know this because we get “ubuntu#” prompt where ‘#’ signifies root privilege. Any further commands will be executed as root.
2. On copying bash from the bin directory to the tmp directory and making it a setuid program and then executing bash from the tmp directory, root privileges are NOT allowed. We know this because we get “bash-4.2$” prompt as opposed a ‘#’ prompt which signifies rot access. Any further commands will be executed with the permissions of a regular user.

**Explanation:**

1. zsh doesn’t have any protective mechanism and hence can be exploited by simply setting the setuid bit using chmod 4755 command on the copied file
2. Bash has some inherent protective mechanisms that do not allow for root access using SetUID.

**Task 3:**

****

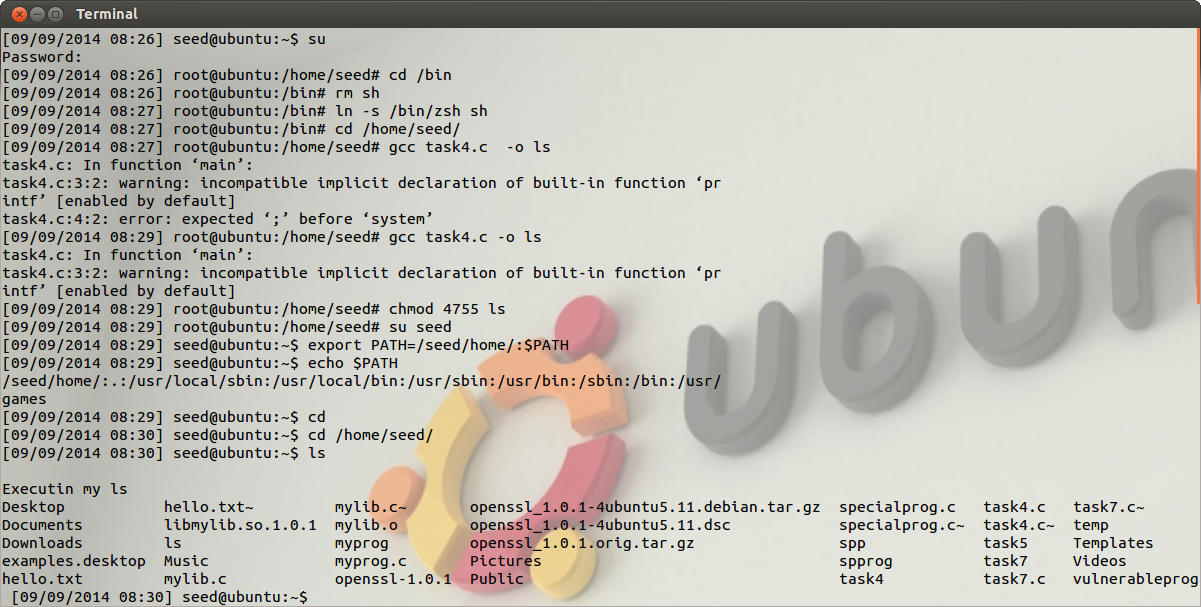
**Observation:**

Since /bin/sh is a symbolic link to /bin/bash and we need to use zsh to discover the vulnerabilities of Set-UID program, we remove sh and instead make links between zsh and sh. So, now sh links to zsh in any /bin/sh calls.

**Explanation:**

To be able to discuss the vulnerabilities of Set-UID we need to use zsh shell as opposed to bash, because bash has security measures that prevents misuse of Set-UID

**Task 4:**

****

****

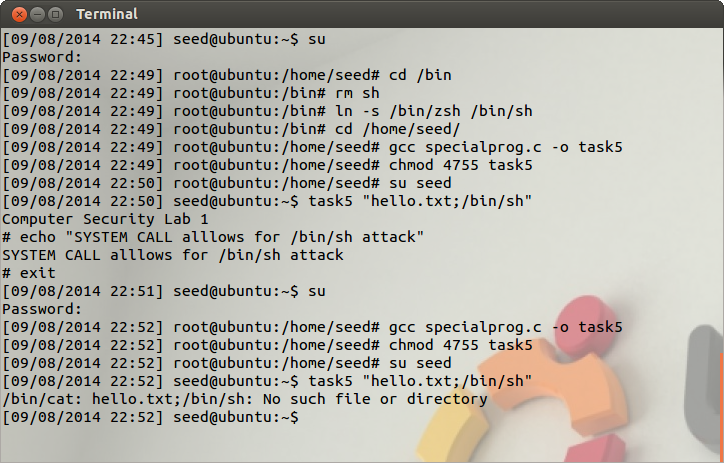
**Observation:**

1. This program owned by root can be executed and run instead of /bin/ls by exporting the path of the directory ahead of all the other paths where ls program is searched for by the shell. export PATH=/home/seed/:$PATH adds the directory with the fake ls program ahead of the /bin/ls file which allows us to execute this program in place of the regular ls program.
2. On changing the symbolic link sh to bash and adding the directory of fake ls ahead of the PATH variable, even bash is executing the fake ls program. *This is a very surprising observation as bash has built-in protection to avoid exactly this kind of a situation.*

**Explanation:**

1. export PATH=/home/seed/:$PATH adds the directory with the fake ls program ahead of the /bin/ls file which allows us to execute this program in place of the regular /bin/ls program.
2. export PATH=/home/seed/:$PATH adds the directory with the fake ls program ahead of the /bin/ls file which allows us to execute this program in place of the regular /bin/ls program in bash as well, this is surprising as bash is expected to drop the set-uid when it sees that the user id is not 0 but the program has root priviliges.

**Task 5:**



**Observation:**

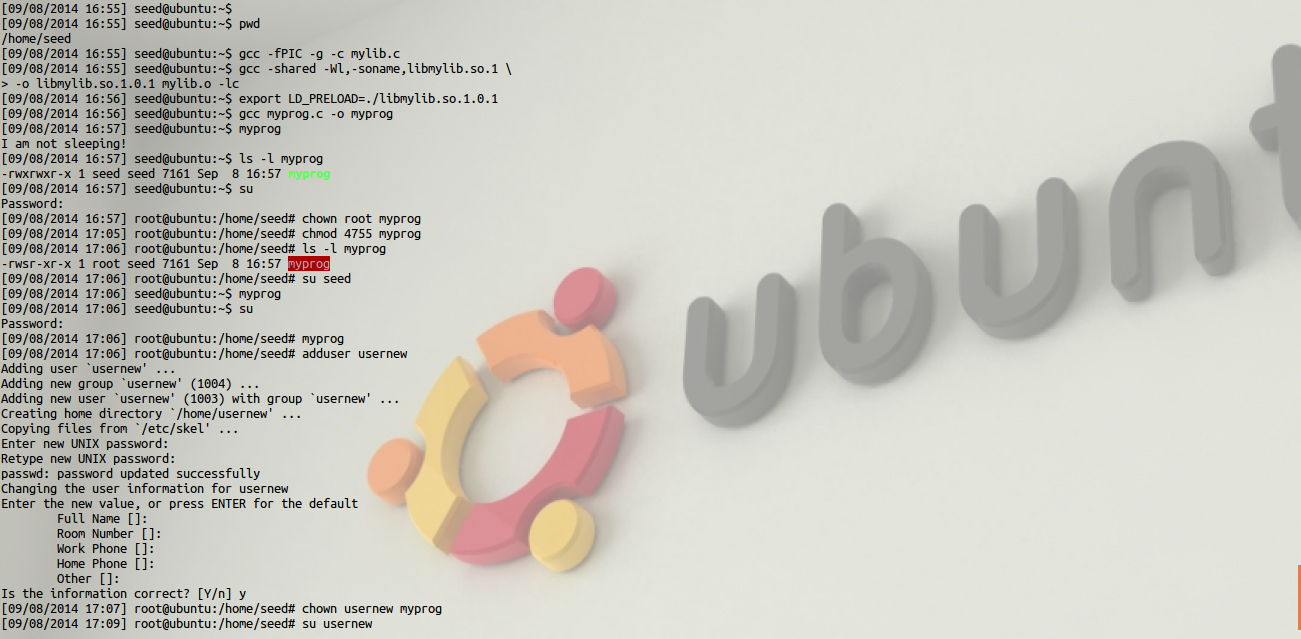
1. (q=0 in the vulnerable program) The program uses system to invoke the command. But more than one command can be executed by the use of delimiter ’;’, so once the command has executed /bin/cat hello.txt in the example provided above, another command “/bin/sh” can be executed which will give us full access to the shell and allow us to execute any command.
2. (q=1 in the vulnerable program) The program does not allow the execution of the /bin/sh file when execve is used to execute the /bin/cat command.

**Explanation:**

1. The system call to /bin/cat is done through the shell where sh is symbolic link to /bin/zsh, so when delimiter is used then next command can be input thus allowing Bob to gain access to the shell which allows him to execute any command with the privileges of the user that executed the program
2. The execve is a library that makes a call to the absolute program without making the use of /bin/sh by calling execve(argv[0], argv[1],0) where the argv[1] is fit directly into the argv[0] program. The execve call does not use system to make a call to the program /bin/cat.

System executes this call by first calling /bin/sh which executes the /bin/cat program, so now when Bob puts the command task5 hello.txt; /bin/sh he will get the output that there is no such file or directory.

**Task 6:**





**Observation:**

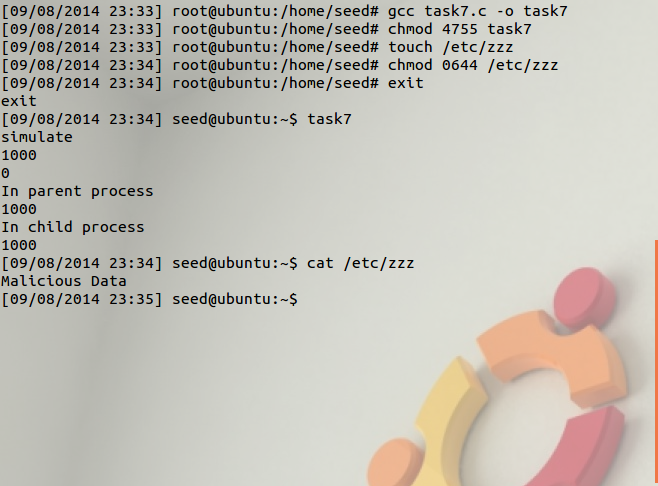
1. Make myprog a regular program and run it as a normal user, the attackers library is used instead of the default library, thus making the attack successful.
2. Make myprog a Set-UID root program, and run it as a normal user, the attacker’s pre-loaded library is not used, thus the attack fails.
3. Make myprog a Set-UID root program, and run it in the root account, the runtime linker will ignore the LD\_PRELOAD variable.
4. Make myprog a Set-UID usernew program, and run it as a normal user, the runtime linker will ignore the LD\_PRELOAD variable.

**Explanation:**

If the environment variable LD PRELOAD is set to libmylib.so.1.0.1, the sleep() in the standard libc will not be invoked; the overloaded sleep() from out library will be loaded instead, and the result will be the “I am not sleeping!” statement being printed to the screen.

The runtime linker/loader (ld.so) is responsible for prevention against attacks where the LD\_PRELOAD variable can be manipulated, if the program is a Set-UID root program and the real UID is not zero, it ignores the environment variable(LD\_PRELOAD).

**Task 7:**

****

**Observation:**

The program is run and the simulated process is executed. The file in /etc/zzz is modified and the Malicious Data entry is made in the zzz file as shown by executing the cat command on /etc/zzz file.

**Explanation:**

The program calls setuid() system call to permanently relinquish root privileges, but this is done in the parent process and not the child process, the child process still has the privileges and will thus be able to edit the zzz file even though the parent process has relinquished its privileges.