Exercise 1: Setting up File Structure and User space The objective of this exercise is to setup the file hierarchy structure and the users that are required for the exercises in this section. The su command is used to switch users.

1. Login as root (password = "enter 2006")

2. Use useradd command to create two new users user1 and user2 as follows:

a. useradd user1 -g users -p user1

b. useradd user2 -g users -p user2

```
root@kali:~# useradd user1 -g users -p user1
root@kali:~# useradd user2 -g users -p user2
```

3. Check user information with the id command. Note the uid, gid for each output.

a. id user1

```
root@kali:~# id user1
uid=1004(user1) gid=100(users) groups=100(users)
```

b. id user2

```
root@kali:~# id user2
uid=1005(user2) gid=100(users) groups=100(users)
```

d. id

```
root@kali:~# id
uid=0(root) gid=0(root) groups=0(root)
```

4. Create directory structure

a. mkdir /test

b. mkdir /test/temp

```
root@kali:~# mkdir /test
root@kali:~# mkdir /test/temp
```

5. Switch user roles as user1 and then back to root using the su command

a. whoami

b. su user1

c. su OR su root (password = "enter 2005")

```
root@kali:~# whoami
root
root@kali:~# su user1
$ su
Password:
```

5. Create a new file as root user and change group ownership as well as user ownership of the file.

a. touch /home/user2/HelloWorld

b. ls –l /home/user2/HelloWorld (observe owner and group)

root is the owner, group root is the owner

c. chgrp users /home/user2/HelloWorld

d. chown user2:users /home/user2/HelloWorld

e. ls –l /home/user2/HelloWorld (observe owner and group)

user2 is the owner and group users is the owner

```
root@kali:~# touch /home/user2/HelloWorld1
root@kali:~# ls -l /home/user2/HelloWorld1
-rw-r--r-- 1 root root 0 Feb 21 01:01 /home/user2/HelloWorld1
root@kali:~# chgrp users /home/user2/HelloWorld1
root@kali:~# chown user2:users /home/user2/HelloWorld1
root@kali:~# ls -l /home/user2/HelloWorld1
-rw-r--r-- 1 user2 users 0 Feb 21 01:01 /home/user2/HelloWorld1
```

Exercise 2: Differences in File and Folder Permissions The objective of the following exercises would be to see the differences in file and folder permissions. The chmod command will be used to change file and directory permission to demonstrate the slight differences in permissions for files and directories.

1. Observe the result of ls and cd commands

a. cd /

```
root@kali:~# cd /
```

b. ls –l

```
root@kali:/# ls -l
total 72
lrwxrwxrwx   1 root root     7 Nov 25 12:24 bin → usr/bin
drwxr-xr-x   3 root root  4096 Nov 25 13:41 boot
drwxr-xr-x  17 root root  3360 Feb 21 00:38 dev
drwxr-xr-x 160 root root 12288 Feb 21 00:49 etc
drwxr-xr-x   6 root root  4096 Jan 27 03:11 home
lrwxrwxrwx   1 root root    35 Nov 25 12:24 initrd.img → boot/initrd.img-5
.3.0-kali2-686-pae
lrwxrwxrwx   1 root root    35 Nov 25 12:24 initrd.img.old → boot/initrd.i
mg-5.3.0-kali2-686-pae
lrwxrwxrwx   1 root root     7 Nov 25 12:24 lib → usr/lib
lrwxrwxrwx   1 root root     9 Nov 25 12:24 lib64 → usr/lib64
lrwxrwxrwx   1 root root    10 Nov 25 12:24 libx32 → usr/libx32
drwx------   2 root root 16384 Nov 25 12:24 lost+found
drwxr-xr-x   3 root root  4096 Nov 25 07:24 media
drwxr-xr-x   2 root root  4096 Nov 25 07:24 mnt
drwxr-xr-x   2 root root  4096 Nov 25 07:24 opt
dr-xr-xr-x 186 root root     0 Feb 21 00:38 proc
drwxr-xr-x  16 root root  4096 Feb 21 00:39 root
drwxr-xr-x  37 root root   920 Feb 21 00:39 run
lrwxrwxrwx   1 root root     8 Nov 25 12:24 sbin → usr/sbin
drwxr-xr-x   3 root root  4096 Nov 25 12:24 srv
dr-xr-xr-x  13 root root     0 Feb 21 00:38 sys
drwxr-xr-x   3 root root  4096 Feb 21 00:50 test
drwxrwxrwt  15 root root  4096 Feb 21 00:39 tmp
drwxr-xr-x  13 root root  4096 Nov 25 12:38 usr
drwxr-xr-x  12 root root  4096 Nov 25 12:39 var
lrwxrwxrwx   1 root root    32 Nov 25 12:39 vmlinuz → boot/vmlinuz-5.3.0-k
ali2-686-pae
lrwxrwxrwx   1 root root    32 Nov 25 12:39 vmlinuz.old → boot/vmlinuz-5.3
.0-kali2-686-pae
```

c. ls -al /home

```
root@kali:/# ls -al /home
total 24
drwxr-xr-x  6 root  root  4096 Jan 27 03:11 .
drwxr-xr-x 19 root  root  4096 Feb 21 00:49 ..
drwxr-xr-x  2  1000  1000 4096 Jan 27 03:08 user1
drwxr-xr-x  2  1001  1001 4096 Feb 21 01:01 user2
drwxr-xr-x  2 user3 user3 4096 Jan 27 03:11 user3
drwxr-xr-x  2 user4 user4 4096 Jan 27 03:11 user4
```

d. What are the directory permissions for user1, user2 and test directories?

```
drwxr-xr-x  2  1000  1000 4096 Jan 27 03:08 user1
drwxr-xr-x  2  1001  1001 4096 Feb 21 01:01 user2
```

```
drwxr-xr-x   3 root root  4096 Feb 21 00:50 test
```

User1 can read write execute the file in directory while users group and others can read and execute.

User2 can read write execute the file in directory while users group and others can read and execute.

Root can read write execute the file in directory

e. Switch to user1 using su user1

```
root@kali:/# su user1
```

f. ls -al /home/user2 (Can you list directory?)

```
$ ls -al /home/user2
total 24
drwxr-xr-x 2  1001  1001 4096 Feb 21 01:01 .
drwxr-xr-x 6 root  root  4096 Jan 27 03:11 ..
-rw-r--r-- 1  1001  1001  220 Jan 27 03:10 .bash_logout
-rw-r--r-- 1  1001  1001 3391 Jan 27 03:10 .bashrc
-rw-r--r-- 1  1001  1001 3526 Jan 27 03:10 .bashrc.original
-rw-r--r-- 1 user2 users    0 Feb 21 00:53 HelloWorld
-rw-r--r-- 1 user2 users    0 Feb 21 01:01 HelloWorld1
-rw-r--r-- 1  1001  1001  807 Jan 27 03:10 .profile
```

yes

g. cd /home/user2 (Can you change directory?)

```
$ cd /home/user2
```

2. Change directory permissions of user2 directory and try again as user1.

a. su root

```
$ su root
Password:
```

b. chmod 740 /home/user2

```
root@kali:/home/user2# chmod 740 /home/user2
```

c. Repeat steps 1e to 1g (Can you list or change directory?)

```
root@kali:/home/user2# su user1
$ ls -al /home/user2
ls: cannot open directory '/home/user2': Permission denied
$ cd /home/user2
sh: 2: cd: can't cd to /home/user2
```

No, user1 does not have permission to list directory

d. su root

```
$ su root
Password:
```

e. chmod 750 /home/user2

```
root@kali:/home/user2# chmod 750 /home/user2
```

f. Repeat steps 1e to 1g (Can you list or change directory?)

```
root@kali:/home/user2# su user1
$ ls -al /home/user2
ls: cannot open directory '/home/user2': Permission denied
$ cd /home/user2
sh: 2: cd: can't cd to /home/user2
```

No user1 does not have the permission to list directory.

g. touch /home/user2/hello12.txt (Can you create new file?)

```
$ touch /home/user2/hello12.txt
touch: cannot touch '/home/user2/hello12.txt': Permission denied
```

User1 does not have the permission to create a new file.

h. su root

```
$ su root
Password:
```

i. chmod 770 /home/user2

```
root@kali:/home/user2# chmod 770 /home/user2
```

j. su user1

```
root@kali:/home/user2# su user1
```

k. Repeat step 2g. (Can you create new file?)

```
$ touch /home/user2/hello12.txt
touch: cannot touch '/home/user2/hello12.txt': Permission denied
```

No, user1 does not have permission to create a file

l. ls –l /home/user2

```
$ ls -l /home/user2
ls: cannot open directory '/home/user2': Permission denied
```

Exercise 3: Default file permissions and Group access control Whenever a new file is created using C program, permissions can be assigned to it. Whatever the permissions are, UNIX system allows the user to filter out unwanted permissions by default. This default setting can be set by the user using the umask command. It is a system call that is also recognized by the shell. The command takes the

permissions set during creation of file and performs a bitwise AND to the bitwise negation of mask value. Some common umask values are 077 (only user has permissions), 022 (only owner can write), 002 (only owner and group members can write), etc.

1. In a terminal window, make sure you are a root user. If not the root user, then switch back to root user (use your password to switch.

```
$ su root
Password:
```

2. Use umask command to check the current mask permission and assign a new mask.

a. umask

```
root@kali:/home/user2# umask
0022
```

b. What is the current mask? How is it interpreted? (try umask –S or the man pages)

```
root@kali:/home/user2# umask -S
u=rwx,g=rx,o=rx
```

Current mask for owner is read write and execute, group is read and execute, others is read and execute

c. cd /test

```
root@kali:/home/user2# cd /test
```

d. touch testmask1

```
root@kali:/test# touch testmask1
```

e. ls -al

```
root@kali:/test# ls -al
total 12
drwxr-xr-x  3 root root 4096 Feb 21 02:00 .
drwxr-xr-x 19 root root 4096 Feb 21 00:49 ..
drwxr-xr-x  2 root root 4096 Feb 21 00:50 temp
-rw-r--r--  1 root root    0 Feb 21 02:00 testmask1
```

f. What are the permissions of the file testmask1

Owner: read+write

Group: read

Other: read

g. umask 0077

```
root@kali:/test# umask 0077
```

h. touch testmask2

```
root@kali:/test# touch testmask2
```

i. Now what are the permissions of the file testmask2

```
-rw-------  1 root root    0 Feb 21 02:01 testmask2
```

Only owners has the permission read and write

3. What is the effect of setting mask value to 0000?

```
root@kali:/test# umask 0000
root@kali:/test# touch testmask3
root@kali:/test# ls -al
total 12
drwxr-xr-x  3 root root 4096 Feb 21 02:09 .
drwxr-xr-x 19 root root 4096 Feb 21 00:49 ..
drwxr-xr-x  2 root root 4096 Feb 21 00:50 temp
-rw-r--r--  1 root root    0 Feb 21 02:00 testmask1
-rw-------  1 root root    0 Feb 21 02:01 testmask2
-rw-rw-rw-  1 root root    0 Feb 21 02:09 testmask3
```

All users(owner, group member, others) can read and write.

4. The risks of setting the extra bits will be covered in Exercise 4 which shows that extra bits should not be set, in general. What should be the umask value to ensure that the “extra bits” can never be set (ie. assigned ‘1’ value)?

The umask should be 4 digits with 0 in front ensure that there will have no extra space for extra bits. Like 0077.

Exercise 4: setuid bit, setgid bit and sticky bit As explained in the background above, the highest three bits of the permission bits represent the setuid bit, setgid bit and the sticky bit. If the setuid bit is set then the uid will always be set to the owner of the file during execution. If the setuid bit is not set then the uid will be the user who executes the process. Similarly, if the setgid bit is set then the gid will be set to the group that owns the file during execution. If the setgid bit is not set then the gid will be the group that executes the process. The sticky bit is set to keep processes in the main memory. In the following exercise, the objective is to demonstrate how processes are affected when the setuid bit is set. The exercise must be begun with root privileges.

a. which touch

```
root@kali:/test# which touch
/usr/bin/touch
```

b. ls –l /bin/touch

```
root@kali:/test# ls -l /bin/touch
-rwxr-xr-x 1 root root 104292 Feb 28  2019 /bin/touch
```

c. chmod 4755 /bin/touch

```
root@kali:/test# chmod 4755 /bin/touch
```

d. ls –l /bin/touch

```
root@kali:/test# ls -l /bin/touch
-rwsr-xr-x 1 root root 104292 Feb 28  2019 /bin/touch
```

e. ls –l /home/user2

```
root@kali:/test# ls -l /home/user2
total 0
-rw-r--r-- 1 user2 users 0 Feb 21 00:53 HelloWorld
-rw-r--r-- 1 user2 users 0 Feb 21 01:01 HelloWorld1
```

f. chmod 700 /home/user2/HelloWorld

```
root@kali:/test# chmod 700 /home/user2/HelloWorld1
```

g. ls –l /home/user2 (observe timestamp and permissions)

```
root@kali:/test# ls -l /home/user2
total 0
-rw-r--r-- 1 user2 users 0 Feb 21 00:53 HelloWorld
-rwx------ 1 user2 users 0 Feb 21 01:01 HelloWorld1
```

Only owner has the permissions on read write and execute.

h. su user1

```
root@kali:/test# su user1
```

j. touch /home/user2/HelloWorld

```
$ touch /home/user2/HelloWorld1
```

j. ls –l /home/user2 (observe timestamp)

```
$ ls -l /home/user2
ls: cannot open directory '/home/user2': Permission denied
```

Permission denied by the directory.

k. su root

```
$ su root
Password:
```

l. chmod 0755 /bin/touch

```
root@kali:/test# chmod 0755 /bin/touch
```

m. su user1

```
root@kali:/test# su user1
```

n. touch /home/user2/HelloWorld (Why permission denied?)

```
$ touch /home/user2/HelloWorld1
touch: cannot touch '/home/user2/HelloWorld1': Permission denied
```

```
$ ls -l /bin/touch
-rwxr-xr-x 1 root root 104292 Feb 28  2019 /bin/touch
```

Because other users does not have any permission on write but read and execute.

Exercise 5: Using setuid bit to “misuse” system The setting of setuid bit can have serious security implications in the access control system. The objective of this exercise is to demonstrate the risks involved in using the setuid bit. There is a user secret_user who has created a secret file called “secret.txt” in the folder /temp to which only that user has access. Your job is to modify system settings such that you have access to the secret file without the owner knowing about it. The constraint is that you cannot change access rights to the file and you cannot spy being the root user. Hint: The owner of the file has all access permissions on the file implicitly. And you already know the root password.

```
root@kali:/# touch /test/temp/secret.txt
root@kali:/# ls -l /test/temp/secret.txt
-rw-rw-rw- 1 root root 0 Feb 21 02:49 /test/temp/secret.txt
root@kali:/# chgrp users /home/secret_user/secret.txt
chgrp: cannot access '/home/secret_user/secret.txt': No such file or direct
ory
root@kali:/# chgrp users /test/temp/secret.txt
root@kali:/# chown secret_user:users /test/temp/secret/txt
chown: cannot access '/test/temp/secret/txt': No such file or directory
root@kali:/# chown secret_user:users /test/temp/secret.txt
root@kali:/# ls -l /test/temp/secret.txt
-rw-rw-rw- 1 secret_user users 0 Feb 21 02:49 /test/temp/secret.txt
root@kali:/#
```

```
root@kali:/# su secret_user
$ chmod 0700 secret.txt
chmod: cannot access 'secret.txt': No such file or directory
$ su root
Password:
root@kali:/# su secret_user
$ chmod 0700 /test/temp/secret.txt
$ su user1
Password:
su: Authentication failure
$ su root
Password:
root@kali:/# touch /test/temp/secret.txt
root@kali:/# chmod 4755 /bin/touch
root@kali:/# touch /test/temp/secret.txt
root@kali:/# ls -l /test/temp/secret.txt
-rwx------ 1 secret_user users 0 Feb 21 02:56 /test/temp/secret.txt
root@kali:/#
```