COMSM0049: Lab 8 – Access Control

Vagrant

```
Vagrant.configure("2") do |config|
  config.vm.box = "generic/debian10"
  config.vm.synced_folder ".", "/vagrant"
  config.vm.provision "shell", inline: "sudo apt-get update --yes"
  config.vm.provision "shell", inline: "sudo apt-get install --yes strace nasm"
end
```

*ls command is used to display the permissions of the files. See **man ls** if you need to know more about it. Now, let's see what **ls -l /bin/ls** displays. Try to understand what the permissions are and which user it belongs to and what is the hard link?

* Now try to find the permission for sudo

  Hint: man sudo

Task 1: Creating new users and checking the default permission

Use **man** to find out the difference between :

- adduser
- useradd

1. Create three new users: Alice, Bob and Charlie and set passwords for each user.
2. Log in as Alice, create a file in Alice's home directory and check its default permissions.
3. Create a group named admins and add bob to it. *man groupadd*
4. Change the group's ownership of the file from alice to admins. *man chown*
5. Change the admins group permissions to read and write for the file you have created. *man chmod*

Question:

1. Can bob modify the file's content? Will charlie be able to modify it? Why?
2. Now add Charlie to the admin's group. Can he modify the file now?

Permission	For a file	For a directory
read (r)	allowed to view file contents	allowed to view directory contents
write (w)	allowed to write to file	allowed to remove or add new files to directory
execute (x)	allowed to execute file	allowed to access files in the directory

Task 2 : Special Permissions

In addition to the regular POSIX permissions of read, write and execute there are 3 special permissions. They hold the same place value as the regular permissions and are:

```

SETUID - set user ID on execute

SETGID - set group ID on execute

StickyBit - puts the directory in sticky mode

```

The SETUID and SETGID permissions allow users and groups who are not the owner or group of a file to execute that file as though they were. When the Sticky Bit is set on a directory, only that directory's owner or root can delete or rename the directory's files.

*Note: Use the octal representation as well *

Permission	Octal Value
read (r)	4
write (w)	2
execute (x)	1
No permission	0

1. Log in as Alice and create a folder called alicefolder. Set the SETUID, SETGID and Sticky Bit permissions, list the permissions. What do you notice?
2. Remove execute for all, but leave the special permissions, what changes do you notice?
3. Assign the minimum group permissions for alicefolder so that users from other groups can browse the folder and read the files within it.
4. Change the owner and group for alicefolder to bob
5. Log is as charlie and create a folder called charlie_likes_to_share. Set the permissions for this folder in such a way that charlie can share the files with alice and bob. This means that alice and bob can browse the folder, read the content of any files, but cannot modify, rename or delete any other files than their own. Have you noticed a limitation during that process?
6. As charlie, create a file sensitivedata.txt and write a line of text in this file. Set no permissions for group and other for this file. Switch user to alice, and try to read the data from this file using nano. Alice has permission to run nano, but not to read the file. So when nano attempts to read the file a “permission denied” error message will be displayed. However, if you set the SUID bit on the nano, alice is granted access to the file. How does it work? The UNIX system doesn’t think bugs is reading file via nano, it thinks “root” is the user and hence the access is granted. Test this and use ps to monitor what is going on.

Hint: man nano

Task 3 : Linux Access Control Lists

 **Hints **:

- The getfacl command displays the file name, owner, group and the existing ACL for a file.

- The setfacl command sets ACLs of files and directories.

The -m option adds or modifies one or more rules in a file or folder's ACL.

- The -b option removes rules in a file or folder's ACL.

Questions:

1. Create 2 additional users abbott and costello. Create a group called actors and add both abbott and costello to it.
2. Create a folder called important-files in the home folder of the user costello. Display the ACL of important-files.
   *At the moment, are there any differences between using ls –la and getfacl? *
3. Now, login as abbott and try to create a file called abbott.txt in the costello-files directory. Did it work? Why?
   Please remove the sticky bit configuration you have added during task 2 and try again. Does it work now?
   If not try to add a rule to the ACL of the folder costello-files that gives to the actors group read, write and execute permissions to that folder.
   Try to create the abbott.txt file again and what do you observe
4. Remove the permissions you have added previously. Try to change the rules between the two users and see how you can play around/affect the permissions.

Extra Task :

**Writing a SUID program in C **

accessmysecret.c

```C
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <errno.h>
int main()
{
 printf(" UID GID \n"
 "Real %d Real %d \n"
 "Effective %d Effective %d \n",
 getuid (), getgid (),
```

```
 geteuid(), getegid());
 FILE *fp = fopen("mysecret", "r");
 if (fp == NULL) {
printf("Error: Could not open file");
exit(EXIT_FAILURE);
 }
 char c;
 while ((c=getc(fp)) != EOF) {
 putchar(c);
 }
 putchar('\n');
 return EXIT_SUCCESS;
}
```

- Now you are going to create a SUID program, to grant access to the contents of your "mysecret" file to anyone who runs the program, without sharing direct access to the file.
  'cat > ~/mysecret'
- Make sure the file is only accessible by the owner :
  'ls -la' should show "rw----------" for that file.
- Create a C program by making a new file "accessmysecret.c"
  ' vi accessmysecret.c'
- Remember, vi is modal. Press "i" to enter insert mode, then enter the code:
- Save your changes and quit (Esc, ":wq").
- Compile the program (which uses the C code to create an executable):
  'gcc accessmysecrets.c -o accessmysecrets'
- Set the permissions for the file (using chmod) to setuid:
  'chmod u+s accessmysecrets'
- Check the permissions include SUID:
  'ls -l accessmysecrets'
- Run the program:
  './accessmysecrets'
- Note that the program outputs its real and effective identity.
- Change to another user, and execute the program:
  '/home/yourusername/accessmysecrets'
- Note that the effective ID is that of the owner of the program. You should also see the contents of the mysecret file, even though you don't have access to the secrets file directly.

1. Think about the security of this solution. How secure is it? Would it be safe for root to be the owner of this program? Why not?
2. Switch to another user and use the SUID accessmysecrets program to get read access to any one of the owner user's files!

Hint:

1. there is a security problem with this code.
2. think about hard links.

Challenge :

- Modify the program to correct the above vulnerability.
- Modify the program so that only the first line of the mysecrets file is displayed to others.
- Modify the program so that the script checks the UID and only continues for a specific user (for example, if the user is root).

**Hint **: "man getuid"