Linux Permissions are the Discretionary Access Control (DAC) scheme on Linux multi-user operating system to create access control on files and other file-like entities like UNIX sockets etc. In this lab, you will review the basic commands and operations in the Linux.

To realize this lab is required to use the Linux System (Kali) embedded in the Virtual Box Environment previously provided by the Professor.

**Task 1**

**Log into the system with a normal user and open a shell (user=kali, password=kali). Following, navigate to the folder “/” and list them. Who is the owner of the folder /etc?**

$cd /

$ls -l

The owner is root.

**Task 2**

**Enter into the folder “/etc/init.d” and list all the existing files. What is the group and the owner of the file x11-common?**

$cd /etc

$ls -l

The owner is root and group, root.

**Task 3**

**Open the file “x11-common” (use your preference editor), insert some lines and save it. What happens? Justify your answer.**

You cannot save the changes in the file, as you did not have permission to do it.

**Task 4**

**Return to your home directory. Create a directory “lab1” and, inside this folder, the file “work1.sh”. In the file “work1.sh”, add these contents:**

**#/bin/sh**

**ls -all**

$cd /home/mx

$mkdir lab\_dir

$cd lab\_dir

$vim work1.sh

**Task 5**

**Execute the file script “work1.sh”. Is it work? If it is not, why? Fix the problem.**

No, it is not run, because you do not have execution permission.

$ ls -l

$ chmod +x work1.sh

Tasks 6

**Linux has a find command, which is responsible to find files and other information in OS. Using the man files of the command “find”, understand how to make a search into the system. Cite all readable files that exist in the home folder (Tip -> use the option “perm”).**

$ find /home/mx -perm -o+r

**Task 7**

**Using the same approach, find all writable files in the home folder.**

$ find /home/mx -perm -o+w

**Task 8**

**Inside your home directory, create a new folder: “lab2”. Inside the newest folder, create a new file, which name is “sample”. Set its permissions such that only the root user can read (Tip 🡪 read about the “chmod” command).**

$mkdir /home/mx/lab2

$cd /home/mx/lab\_dir2

$chmod 600 sample

**Task 9**

**In the OS, create a new group of users, which name is lab\_admin. Create two new users (admin1 and admin2) and set them into this new group (Tip 🡪 read about groupadd and adduser command).**

$ sudo groupadd lab\_admin

$sudo adduser admin1

$sudo adduser admin2

$sudo adduser admin1 lab\_admin

$sudo adduser admin2 lab\_admin

**Task 10**

**Change the group of the sample file to the ‘lab\_admin’ group and change the owner to be “admin1” 🡪 Tip 🡪 read about *chwon* command*.***

$sudo chown admin1:lab\_admin shell

**Task 11**

**Setuid, which stands for a set user ID on execution, is a special type of file permission in Unix and Unix-like operating systems such as Linux and BSD. It is a security tool that permits users to run certain programs with escalated privileges.**

**When an executable file's setuid permission is set, users may execute that program with a level of access that matches the user who owns the file. For instance, when a user wants to change their password, they run the passwd command. The passwd program is owned by the root account and marked as setuid, so the user is temporarily granted root access for that limited purpose.**

**List the file /usr/bin/passwd. Does the file include the setuid permission set?**

Yes, when viewing a file's permissions with the ls -l command, the setuid permission is displayed as an "s" in the "user execute" bit position. For example:

ls -l /usr/bin/passwd

-rw**s**r-xr-x 1 root 54192 Nov 20 17:03 /usr/bin/passwd

**Task 12**

**Change the permission of work1.sh, making that only the user can read, write, and execute it. Use absolute mode (numeric permissions)**

Set the permission. 7 for user, 0 for group and 0 for others.

$chmod 700 /work1.sh

**Task 13**

**Change the permission of work1.sh, making that any user can read it, the group can read/write to it and the user (owner) can read/write/execute it. Use absolute mode.**

Set the permission. 7 for user, 6 for group and 4 for others.

$chmod 764 /work1.sh

**Task 14**

**Change the permission of work1.sh, making that only the user (owner) can read/write/ execute the file, and the other will have the execution permission. Following, assure that the others, when will execute the file, will do with the privileges of the owner user.**

Set the permission. 7 for user, 1 for group and 1 for others. To make sure it only executes with owner’s user’s privileges, set setuid bit, which means setting value 4.

# chmod 711 work1.sh

#chmod u+s work1.sh

#chmod o+s work1.sh

#chmod o+x work1.sh

**Task 15**

**Change the permission set of work1.sh so that only the user (owner) can read, write, and execute it. Use symbolic mode (letters).**

Set the permission. Depending on the current permissions, the user may have to nullify some of the permissions.

Command:

chmod u+rwx work1.sh

chmod g-rw work1.sh

chmod o-r work1.sh

**Task 16**

**Change the permission set of work1.sh so that any user can read it, the group can read/write to it and the user (owner) can read/write/execute it. Use symbolic mode.**

chmod g+rw work1.sh

chmod o+r work1.sh

**Task 17**

**Change the permission set of work1.sh so that only the user (owner) can read/write/ execute, group, and any user can execute it. However, whenever anyone executes it, it should run with the privileges of the group. Use absolute mode.**

Set the permission. To make sure it only executes with the group’s privileges, set setgid bit.

chmod 2711 work1.sh

Here, 2 is to setguid bit (4 can be used for setuid bit and 1 can be used for sticky bit)

7 is to set read/write/execute permissions for the owner.

1 is to set execute permission for group.

1 is to set execute permission for other users.

**Task 18**

**The sticky bit is a special permission which does not affect individual files. However, at the directory level, it restricts file deletion. Only the owner (and root) of a file can remove the file within that directory. A common example of this is the /tmp directory:**

***[tcarrigan@server article\_submissions]$ ls -ld /tmp/***

***drwxrwxrwt. 15 root root 4096 Sep 22 15:28 /tmp/***

**The permission set is noted by the lowercase t, where the x would normally indicate the execute privilege.**

**Change the permission set of work1.sh so that only the owner can rename or delete this file while maintaining the existing permissions. Use absolute mode.**

Set sticky bit on the file.

chmod 3711 /work/readme.txt

<https://medium.com/swlh/linux-privilege-escalation-in-four-ways-eedb52903b3>

<https://payatu.com/guide-linux-privilege-escalation>

<https://book.hacktricks.xyz/linux-unix/privilege-escalation>

<https://www.beyondtrust.com/blog/entry/how-a-linux-attacker-can-escalate-from-low-level-privileges-to-root>

Part II - Linux Permission Escalation

**Exploiting SUDO rights/user**

If the attacker cannot directly get root access via any other techniques, he might try to compromise any of the users who have SUDO access. Once he has access to any of the sudo users, he can basically execute any commands with root privileges.

Administrators might just allow the users to run a few commands through SUDO and not all of them but even with this configuration, they might introduce vulnerabilities unknowingly which can lead to privilege escalation.

A classic example of this is assigning SUDO rights to the find command so that another user can search for files/logs in the system. While the admin might be unaware that the ‘find’ command contains parameters for command execution, an attacker can execute commands with root privilege.

**A. Prints the commands which we can run as SUDO.**

$sudo -l

**B. Now, you know which command the user can run, using the sudo command. You see that your default user (mx) can perform any command using sudo. You can use for example the find command’s exec parameter for arbitrary code execution.**

$sudo find /home -exec sh -I \;

**Observation:** NEVER give SUDO rights to any of programming language compiler, editor or interpreter. This technique can be used in vi, more, less, perl, ruby, gdb, and others.

$sudo python -c ‘import pty; pty.spawn(‘bin/’bash’);’

How to fix the problem:

* Do not give sudo rights to any program which lets you escape to the shell.
* Never give SUDO rights to vi, more, less, nmap, perl, ruby, python, gdb and others.

**Cracking Passwords**

A couple files of particular interest on [Linux systems](https://null-byte.wonderhowto.com/how-to/linux-basics/) are the **/etc/passwd** and **/etc/shadow** files. The /etc/passwd file contains basic information about each user account on the system, including the root user which has full administrative rights, system service accounts, and actual users. There are seven fields in each line of /etc/passwd. A typical line looks something like this:

msfadmin:x:1000:1000:msfadmin,,,:/home/msfadmin:/bin/bash

The first field is the user's login name. The second field traditionally contained an encrypted password, but nowadays (unless you get extremely lucky) it merely contains the letter "x," to denote that a password has been assigned. If this field is blank, the user does not need to supply a password to log in.

The third field is the user ID, a unique number assigned to the user, followed by the group ID in the fourth field. The fifth field is typically the full name of the user, although this can also be left blank. The sixth field is the user's home directory, and finally, the seventh field is the default shell, usually set to /bin/bash.

The /etc/shadow file contains the encrypted passwords of users on the system. While the /etc/passwd file is typically world-readable, the /etc/shadow is only readable by the root account. The shadow file also contains other information such as password expiration dates. A typical line in /etc/shadow will look like this:

msfadmin:$1$XN10Zj2c$Rt/zzCW3mLtUWA.ihZjA5/:14684:0:99999:7:::

**Challenge 1 – Cracking the Linux Password**

Create 4 new users and set a very strong password to them: user1, user2, user3, and user4. Save the strong password in a file or in a piece of paper.

Since we have achieved root-level access with our [kernel exploit](https://null-byte.wonderhowto.com/how-to/perform-local-privilege-escalation-using-linux-kernel-exploit-0186317/), we can use these files to uncover passwords of other users in the hopes of [pivoting to other systems](https://null-byte.wonderhowto.com/how-to/hack-like-pro-pivot-from-victim-system-own-every-computer-network-0149847/) and [furthering exploitation](https://null-byte.wonderhowto.com/collection/exploit-development/).

The first thing we need to do is copy the contents of /etc/passwd and /etc/shadow into their own text files on our local machine; let's call them passwd.txt and shadow.txt, respectfully.

John the Ripper is a popular password cracking tool that supports many common hash types as well as a useful autodetect feature. It has been around for a while now, and as such, it continues to be one of the strongest and easiest to use crackers available.

*Support Tutorials:*

* <https://linuxhint.com/john_ripper_ubuntu/>
* <https://null-byte.wonderhowto.com/how-to/hack-like-pro-crack-user-passwords-linux-system-0147164/>

**Challenge 2 – Fix the sudo issues**

Following the support tutorials, configure the sudo application for the default Linux user and the users that you create, fixing all issues that enable the sudo exploitation to escalate the user permission.

* <https://tecadmin.net/linux-sudo-privileges-configuration/>
* <https://www.tecmint.com/sudoers-configurations-for-setting-sudo-in-linux/>