Day 6 Task: File Permissions and Access Control Lists

**Linux file permissions and ownership** are fundamental concepts that control who can access, modify, or execute files and directories in a Linux-based operating system. Understanding these concepts is crucial for managing the security and integrity of your system. Let's dive into these concepts:

**File Permissions:**

File permissions define what actions (read, write, execute) can be performed on a file or directory by different user categories (owner, group, others).

**Read (r):** Allows viewing the contents of a file or listing the contents of a directory.

**Write (w):** Permits modifying the file or adding/removing files within a directory.

**Execute (x):** Allows executing a file (for scripts or programs) or accessing a directory (if x permission is not set on a directory, you can't cd into it).

Permissions are represented using a 3-character string (e.g., "rw-r--r--" for a file), where the first three characters are for the owner, the next three are for the group, and the last three are for others. You can set and change permissions using the chmod command.

**Ownership:**

Ownership defines who owns a file or directory and which group is associated with it.

**Owner:** The user who created the file or directory is the owner. Owners can change permissions, change ownership, or delete the file.

**Group:** Every file or directory is associated with a group. Group members have the permissions defined for the group. Group ownership allows multiple users to share access to files.

**Others:** Users who are not the owner and not in the associated group fall into the "others" category.

You can view the ownership of files using the ls -l command, and you can change ownership using the chown command.

**Permission Representation:**

In the output of ls -l, file permissions are displayed as a 10-character string, including the type of the file and the permissions for owner, group, and others. For example, "drwxr-xr-x" indicates a directory with read, write, and execute permissions for the owner and read and execute permissions for group and others.

The first character represents the file type ('-' for regular file, 'd' for directory, and more).

The next three characters represent owner permissions.

The next three characters represent group permissions.

The last three characters represent others permissions.

**Numeric Representation:**

File permissions can also be represented numerically, where each permission is assigned a value: **read (4), write (2), and execute (1).** You can calculate the numeric representation by adding the values for each permission. For example, "rw-r--r--" is represented as 644 for a regular file.

Numeric representation is often used with the chmod command, where you set permissions using a 3-digit number.

Understanding and correctly managing file permissions and ownership are essential for maintaining the security and integrity of a Linux system. Care should be taken to grant the necessary permissions while restricting unnecessary access to files and directories.

**Tasks:**

1. Create a simple file and do ls -ltr to see the details of the files refer to Notes

Each of the three permissions are assigned to three defined categories of users. The categories are:

owner — The owner of the file or application.

"chown" is used to change the ownership permission of a file or directory.

group — The group that owns the file or application.

"chgrp" is used to change the group permission of a file or directory.

others — All users with access to the system. (outised the users are in a group)

"chmod" is used to change the other users permissions of a file or directory.

**Solution:**

**ubuntu@DevOps**:**~**$ touch simplefile.txt

**ubuntu@DevOps**:**~**$ ls -ltr simplefile.txt

-rw-rw-r-- 1 ubuntu ubuntu 0 Nov 7 22:45 simplefile.txt

|. |. | |. |. |

own-grp-other Fl-typ user group

As a task, change the user permissions of the file and note the changes after ls -ltr

**ubuntu@DevOps**:**~**$ chmod 700 simplefile.txt

**ubuntu@DevOps**:**~**$ ls -ltr simplefile.txt

**-rwx------ 1 ubuntu ubuntu 0 Nov 7 22:45 simplefile.txt**

1. Read about ACL and try out the commands getfacl and setfacl

**Access Control Lists (ACLs)** in Linux are a more flexible and fine-grained way of managing file and directory permissions compared to the standard Linux file permission system. ACLs provide additional access control options beyond the traditional owner, group, and others permissions. They allow you to define permissions for specific users and groups on a per-file or per-directory basis.

Here are the key aspects of ACLs in Linux:

**Traditional Linux Permissions vs. ACLs:**

Traditional Linux file permissions are represented by a 3x3 matrix (owner, group, others), which can limit access to files and directories based on these three categories. However, this system can be restrictive when you need more granular control.

**Basic ACL Terminology:**

ACL Entry: Each ACL entry is composed of three components: a user or group, a permission set (read, write, execute), and a permission mask.

User ACL: Specifies permissions for a specific user.

Group ACL: Specifies permissions for a specific group.

Default ACL: Applies to newly created files or directories within a directory. It's a way to set default permissions.

Setting ACLs:

To set an ACL on a file or directory, you can use the setfacl command:

**ubuntu@DevOps**:**~**$ setfacl -m u:tom:rw ./simplefile.txt

**ubuntu@DevOps**:**~**$ getfacl simplefile.txt

# file: simplefile.txt

# owner: ubuntu

# group: ubuntu

user::rwx

user:tom:rw-

group::---

mask::rw-

other::---

u:username: Specify the user you want to grant permissions to.

permissions: Set the desired permissions.

You can also use g:groupname for group ACLs, and you can set default ACLs with the -d flag.

Viewing ACLs:

To view the ACL of a file or directory, you can use the getfacl command:

**ubuntu@DevOps:~$ getfacl simplefile.txt**

# file: simplefile.txt

# owner: ubuntu

# group: ubuntu

user::rwx

group::---

other::---

This will display the ACL entries, including user and group permissions.

**Example:**

Here's an example of using ACLs to grant read and write permissions to a specific user on a directory:

**ubuntu@DevOps**:**~**$ setfacl -m u:tom:rw ./simplefile.txt

**ubuntu@DevOps**:**~**$ getfacl simplefile.txt

# file: simplefile.txt

# owner: ubuntu

# group: ubuntu

user::rwx

user:tom:rw-

group::---

mask::rw-

other::---

This allows the **user "tom" to read and write files within the directory,** in addition to the directory's standard owner, group, and others permissions.

ACLs are particularly useful in scenarios where you need to give specific users or groups different levels of access within a directory or on specific files. They provide more flexibility and control, but it's essential to understand how they interact with standard Linux permissions to ensure the desired access control. Additionally, not all file systems support ACLs, so be aware of the limitations and compatibility of your file system.