File Permissions & Security in Linux

Linux is known for its robust security model, which revolves around **file permissions, user roles, and access control mechanisms**. Understanding these principles is essential for maintaining a secure and well-managed system.

1Understanding Linux File Permissions

In Linux, **every file and directory has specific permissions** that define who can read, write, or execute them. These permissions are assigned based on three categories:

User Type	Description
Owner	The user who created the file/directory.
Group	A set of users who share access to the file.
Others	All users who are not the owner or in the group.

Each file in Linux has three main permissions:

Permission	Symbol	Meaning
Read	`r` (4)	View file contents or list directory contents.
Write	`w` (2)	Modify file contents or create/delete files inside a directory.
Execute	`x` (1)	Run a file (if it's a script or program) or enter a directory.

Example of File Permissions

Running the `ls -l` command displays file permissions:

```
bash
CopyEdit
ls -l myfile.txt
```

Example Output:

```
csharp
CopyEdit
-rwxr--r--  1 user group  1024 Feb 7 10:00 myfile.txt
```

Breaking it down:

- **`-rwxr--r--`** → The first character `-` means it's a file (`d` would indicate a directory).
- **Owner (`rwx`)** → Can read, write, and execute the file.
- **Group (`r--`)** → Can only read the file.
- **Others (`r--`)** → Can only read the file.

---

2 Modifying File Permissions in Linux

Permissions can be modified using the **`chmod`** (change mode) command.

Using Symbolic Notation (`+` to add, `-` to remove, `=` to set)

```
bash
CopyEdit
chmod u+x myscript.sh   # Give execute permission to the owner
chmod g-w myfile.txt    # Remove write permission from the group
chmod o=r myfile.txt    # Set others' permission to read-only
```

Using Numeric (Octal) Notation

Each permission type has a corresponding numeric value:

- `r` = **4**, `w` = **2**, `x` = **1**

To set specific permissions, add the values together:

```
bash
CopyEdit
chmod 755 myscript.sh   # rwx for owner, r-x for group, r-x for others
chmod 644 myfile.txt    # rw- for owner, r-- for group, r-- for others
```

---

3 Changing File Ownership and Group

The **`chown`** (change owner) and **`chgrp`** (change group) commands manage ownership.

- **Change file owner:**

  ```
  bash
  CopyEdit
  chown newuser myfile.txt
  ```

- **Change file group:**

  ```
  bash
  CopyEdit
  ```

```
chgrp newgroup myfile.txt
```

- **Change both owner and group:**

```
bash
CopyEdit
chown newuser:newgroup myfile.txt
```

---

4 Special Permissions: SUID, SGID, and Sticky Bit

Linux provides additional **special permissions** for specific security needs:

Special Permission	Symbol	Purpose
SUID (Set User ID)	`s` (in owner execute bit)	Allows a file to be executed with the file owner's privileges.
SGID (Set Group ID)	`s` (in group execute bit)	Inherits the group of the file when executed or created.
Sticky Bit	`t` (in others execute bit)	Prevents others from deleting files in a shared directory.

Applying Special Permissions

- **Set SUID:** `chmod u+s file.sh`
- **Set SGID:** `chmod g+s directory/`
- **Set Sticky Bit:** `chmod o+t /shared-folder/`

---

5 File Security & Access Control in Linux

Using `umask` for Default Permissions

When new files are created, Linux assigns default permissions using the **`umask`** value.

- View current `umask`:

```
bash
CopyEdit
umask
```

- Change default permissions (e.g., `0022` for `rw-r--r--`):

```
bash
CopyEdit
umask 0022
```

---

6 Securing Linux Files with ACL (Access Control List)

ACLs allow more **fine-grained control** over file access beyond standard permissions.

- **Check ACL permissions:**

```
bash
CopyEdit
getfacl myfile.txt
```

- **Grant specific user permission:**

```
bash
CopyEdit
setfacl -m u:username:rwx myfile.txt
```

- **Remove ACL permissions:**

```
bash
CopyEdit
setfacl -x u:username myfile.txt
```

---

7 Linux Security Best Practices

- **Use Strong Passwords & Enable Multi-Factor Authentication (MFA).**
- **Disable Root Login & Use a Non-Root User (`sudo` for Privileges).**
- **Set Proper File Permissions to Limit Unauthorized Access.**
- **Regularly Update Packages & Security Patches.**
- **Enable Firewalls (`ufw`, `iptables`) to Block Unwanted Traffic.**
- **Use SSH Key Authentication Instead of Passwords.**
- **Monitor System Logs (`/var/log/auth.log`, `/var/log/syslog`).**

Conclusion

Linux provides a **strong permission-based security model** that protects files and directories from unauthorized access. By understanding file permissions, ownership, ACLs, and security best practices, system administrators and users can **enhance security and prevent vulnerabilities** in their systems.