CHAPTER - 1

SSH (Secure Shell):

SSH is a protocol used for secure remote sessions and communication between a client and server.

Default Port: SSH typically operates on port 22.

Example: If my office is located in the city and the office server contains the files and directories I need for my work, I don't need to physically travel to the office.
Instead, I can remotely access the server from home to work with those files and directories.
Various tools and packages are available for remote sessions, such as RDP for Windows, PuTTY, and Telnet.

SSH-KEYGEN:

SSH-KEYGEN is a tool used to generate a pair of cryptographic keys: a public key and a private key.
Analogy: This is similar to WhatsApp's end-to-end encryption.
The public key is used to encrypt a message, turning it into cipher text or a "salted" address, while the private key is used to decrypt that cipher text back into the original message.
Cracking this type of encryption is extremely difficult, taking an estimated 100 to 500 years.

Logging in with SSH:
Initially, when logging into a machine using SSH, you are prompted to enter a password for authentication.
However, after setting up SSH key-based authentication, you won't be asked for a password when logging in.

How SSH Works:

1.First, we generate a pair of keys (public and private) on our local machine.
2.We then share our public key with the server for identification.
3.The server uses this public key to identify our credentials and sends back an authentication message securely encrypted.
4.This encrypted message can only be opened with our private key.
5.After successful authentication, a secure tunnel is established.
6.All data transmitted through this tunnel is encrypted, ensuring security during transmission.
7.Once the SSH key-based authentication is set up, future logins will not require a password, as the private key automatically provides the necessary authentication.

Three methods of SSH-keygen:

i)Authentication with passphrase
ii)Authentication without passphrase
iii)Authentication with custom Keypairs

Algorithm for SSH – SHA Secure Hash Algorithm (SHA-256)

Commands:

[student@serverb]#ssh-keygen —> It create a two keys in local machine public and private

[student@serverb]#ssh-copy-id -i .ssh/id_rsa.pub student@servera → copy our public key and send to target machine servera

[student@serverb]#ssh student@servera —> Now its login without asking password

[student@servera]# ctrl + d = logout

[student@serverb]# ls .ssh
id_rsa.pub id_rsa

[student@serverb]#rm -f .ssh/id_rsa.pub —> Delete the keys from a ssh directory

[student@serverb]#rm -f .ssh/id_rsa

[student@serverb]#ssh student@servera —> Now your prompted asking a password for login.

—---

CHAPTER - 3

1.What is users?
In Linux, a "user" refers to an account that can log into the system and access various resources. Each user has a unique username, a user ID (UID), and is part of one or more

USER AND GROUP MANAGEMENT:

1.What is users?
In Linux, a "user" refers to an account that can log into the system and access various resources. Each user has a unique username, a user ID (UID), and is part of one or more groups.

1. Types of Users

a.Root User: The superuser with UID 0, having unrestricted access to the system. Can perform any administrative tasks.
b.Regular Users: Normal user accounts with a unique UID (typically starting from 1000). These users have limited permissions based on their groups and file ownership.
c.System Users: Accounts used by system processes and services, typically with UIDs below 1000. These accounts are not meant for interactive logins.

Analogy: In an organization, there are many employees, each with unique responsibilities.
It's not feasible to allocate a separate server for every employee, so instead, multiple user accounts are created on a single server.

A user in a Linux system represents an individual or a role with specific permissions and access rights.
For instance, you might have one user account for the morning shift and another for the night shift,
allowing different people to work on the same machine at different times without interrupting each other's work.

As a system administrator, your role is to monitor, manage, and configure these users, ensuring that they have the appropriate access to resources and that the system runs smoothly.
By managing users effectively, you can maintain security, ensure continuity of operations, and optimize the use of system resources.
Commands

Configuration files:
#/etc/passwd ---> It holds the user account information
#/etc/shadow ---> It holds the user account password in encrypted method
#/etc/group ---> It holds the group information
#/etc/gshadow ---> It holds the group password in encrypted method

1)How to add the user in linux

#useradd username --> Used to add the users

2)How to set a password for a user

There is two method
a)passwd username ---> It doesnt show the password in a promt

b)passwd --stdin username ---> It show the password in a promt

In useraccount has a 7 fields like name, password,UID,GID,Label,Home directory,shell access

Ex: employee1:x:1000:1000::/home/employee:/bin/bash ---> 7 fields

3)User Modification

a)How to change the username

```
#usermod -l newname oldname

#usermod -l maries employee1
```

Ex: **maries**:x:1000:1000::/home/employee:/bin/bash ---> 7 fields

b)How to change the user-id for a user

```
#usermod -u UID username

#usermod -u 2000 maries
```

Ex: maries:x:**2000**:1000::/home/employee:/bin/bash ---> 7 fields

c)How to add the label for a user

```
#usermod -c "label" username

#usermod -c "System Administrator" maries
```

Ex: maries:x:2000:1000:**System Administrator**:/home/employee:/bin/bash ---> 7 fields

d)How to Change the home directory for a user

```
#usermod -md homedirectory(/home) username

#usermod -md /home/maries maries
```

Ex: maries:x:2000:1000:System Administrator:**/home/maries**:/bin/bash ---> 7 fields

/root is home directory for a root ---> root files and directories stores under /root directory
/home is home directory for a user ---> user files and directories stores under /home directory

e) Shell access

/bin ---> Essentail Binary commands like (ls,cp,mv...)
/bash ---> Is give login access for a user

/sbin ---> Essential System Binary Commands like (lvcreate, lvextend...)

/nologin ---> Is not give the login access for a user

```
#usermod -s /bin/bash or /sbin/nologin username
```

```
#usermod -s /sbin/nologin maries
```

Ex: maries:x:2000:1000:System Administrator:/home/maries:**/sbin/nologin** ---> it doesnt login to the machine

```
#su - maries
your account is not available
```

```
#usermod -s /bin/bash maries
```

Ex: maries:x:2000:1000:System Administrator:/home/maries:**/bin/bash** ---> it login to the machine

```
#su - maries
[maries@workstation]#
```

Group Management:

What is a Group in Linux?
In Linux, a group is a collection of users that share common permissions or access rights to certain resources, such as files, directories, and devices. Groups help in managing and organizing users, especially in environments where multiple users need to collaborate or share resources.

Each group is identified by a Group ID (GID), and users can be assigned to one or more groups. Every user has a primary group, and they can also belong to supplementary (or secondary) groups.

Why Do We Need Groups?

Instead of assigning permissions to individual users, permissions can be assigned to groups. This allows all members of the group to have the same access rights to files, directories, or other resources. For example, a group called "developers" might have write access to a project's directory, enabling all developers to work on the project without needing individual permissions.

a)How to add or create the group

```
#groupadd groupname
```

#groupadd cse

Ex: cse:x:1677:

b)How to change the group name

#groupmod -n newgroup oldgroup

#groupmod -n cyber cse

Ex: **cybe**r:x:1677:

c)How to set a password for group

#gpasswd cyber

d)How to add the user in a group

#gpasswd -a user1 cyber --->Adding a single user

Ex: cyber:x:1677:**user1**

#gpasswd -M user2,user3,user4 cyber --->Adding Multiple user in a group

Ex: cyber:x:1677:**user1,user2,user3,user4**

e)How to add the user as secondary group

#useradd -G cyber rohith ---> add the new user as secondary group

Ex: cyber:x:1677:user1,user2,user3,user4,**rohith**

#usermod -G cyber maries ---> add the existing user as secondary group

Ex: cyber:x:1677:user1,user2,user3,user4,rohith,**maries**

f)How to remove the user from a group

#gpasswd -d user1 cyber

Ex : cyber:x:1677:user2,user3,user4,rohith,maries

SUDO:Superuser DO

It allows a permitted user to execute a command as the superuser (root) or another user, as specified by the security policy. The sudo command is used to perform tasks that require administrative or root privileges without logging in as the root user.

When a user account is created, it does not automatically have administrative privileges. To grant a user the ability to execute commands with superuser rights, the user must be explicitly added to the sudoers file or to a group with sudo privileges.

Configuration File for `/etc/sudoers`

1. Command:

```
vim /etc/sudoers
```

2. Instructions:
-Open the `/etc/sudoers` file using the `vim` editor.
- In the file, search for the line that contains:

```
root ALL=(ALL) ALL
```

- After this line, add the following configuration to grant specific permissions to the user:

```
username ALL=(ALL) ALL
```

- Replace `username` with the actual username of the user you want to grant `sudo` privileges.

Example:

If you want to give the user `john` full sudo privileges, you would add:

```
john ALL=(ALL) ALL
```

This addition will allow the user `john` to execute any command with root privileges.

Methods:

1. Gaining Full Access
- Syntax:

```
username ALL=(ALL) ALL
```

- Example:

```
maries ALL=(ALL) ALL
```

- **Explanation**: This configuration gives the user ("maries" in this example) full root privileges, allowing them to run any command as any user on the system.

2. Providing Limited Access
- Syntax:

```
maries ALL=(ALL) /usr/sbin/useradd,/usr/sbin/usermod
```

- Explanation: This configuration allows the user to run only specific commands (`useradd` and `usermod`) with root privileges. The user ("maries") can only execute these commands as the root user.

3. Providing Full Access and Limiting Some Commands
- Syntax:

```
maries ALL=(ALL) ALL, !/usr/sbin/userdel, !/usr/sbin/groupdel
```

- Explanation: This setup allows the user ("maries") to run all root commands except for the specified ones (`userdel` and `groupdel`). The exclamation mark `!` is used to exclude these commands.

4. Running Sudo Commands Without a Password
- Syntax:

```
user1 ALL=(ALL) NOPASSWD: ALL
```

- Explanation: This configuration allows the user (`maries`) to run any command with `sudo` without being prompted for a password.

—---

CHAPTER - 4

FILES AND DIRECTORY PERMISSIONS

Linux as a Multi-User System:

- Linux is a multi-user operating system, meaning multiple users can work on the same system simultaneously.
- Each file and directory in your account can be protected or made accessible to other users by adjusting its access permissions.

Role of a System Administrator:

- As a system administrator, you are responsible for managing access to files and directories.
- This involves assigning the appropriate permissions to users to control who can read, write, or execute a particular file.

Analogy:
Consider an organisation with three users and one file:

- Developer: Has read and write permissions (rw) for the file.
- Tester: Has read and execute permissions (rx) for the file.
- Marketer: Has only read permission (r) for the file.

Types of Access Permissions:
Each file and directory has three types of access permissions:

1. Read (r):
   - Allows the user, file owner, or group members to read the contents of a file.

2. Write (w):
   - Allows the user, file owner, or group members to modify, write to, or delete the file.

3. Execute (x):
   - Allows the user, file owner, or group members to execute the file (if it is a script or program).

Assigning Permissions:
Permissions can be assigned using two methods: **Symbolic** and **Numeric**.

1. Symbolic Method:

- Who (u, g, o):
  - `u`: User (file owner)
  - `g`: Group
  - `o`: Others

- What (+, -, =):
  - `+`: Add permission
  - `-`: Remove permission
  - `=`: Set permission

- Which (r, w, x):
  - `r`: Read
  - `w`: Write
  - `x`: Execute

Example:

To assign permissions, use the following format:

chmod u+rwx,g+rx,o+r filename

chmod u-rwx,g-rx,o-r filename

chmod u=rwx,g=rx,o=r filename

This command sets the following permissions on `filename`:
- User (u): Read, write, and execute (rwx)
- Group (g): Read and execute (rx)
- Others (o): Read only (r)

2. Numeric Method:

In the numeric method, permissions are represented by a three-digit number:

- Read (r) = 4
- Write (w) = 2
- Execute (x) = 1

The sum of these numbers determines the permission for each category (user, group, others).

Examples:

- 7 (rwx): 4 (read) + 2 (write) + 1 (execute)
- 5 (r-x): 4 (read) + 0 (write) + 1 (execute)
- 4 (r--): 4 (read) + 0 (write) + 0 (execute)

To set permissions using the numeric method:
ex:
chmod 754 filename

u=7 g=5 x=4

This command sets the following permissions on `filename`:
- 7 (rwx): User has read, write, and execute permissions.
- 5 (r-x): Group has read and execute permissions.
- 4 (r--): Others have read-only permission.

Permission Representation Example:

Using the numeric method, the command:

ex:
chmod 755 filename

Sets:
- User (u): Read, write, and execute (rwx)
- Group (g): Read and execute (r-x)
- Others (o): Read and execute (r-x)

How to change Ownership:

ls -l —> for check the permissions
ls -la —-> for check the file permissions
ls -ld —-> for check the directory permissions

#ls -l filename
-rw-r-xr-- root root filename

Root —> is owner of the file
Root —> is group owner of the file

chown command is used to change the ownership of both user and groups

#chown username filename

-rw-r-xr-- username root filename

#chown :groupname filename
-rw-r-xr-- username groupname filename

chgrp is used to change the group ownership for the file

#chgrp developer filename
-rw-r-xr-- username developer filename

Example :

Step 1: Create a File as Root

First, login as the root user or switch to root using `sudo`:

su - root

Password : redhat

Create a file called `example.txt`:

```
touch /root/example.txt
```

Step 2: Set Permissions for the File

Set the permissions for the file. For example, let's set the following permissions:
- `rwx` for the owner (`root`)
- `r--` for the group
- `---` for others

```
chmod 740 /root/example.txt
```

Step 3: Verify the File's Permissions**

Check the permissions for `example.txt`:

```
ls -l /root/example.txt
```
Output:

```
-rwxr----- 1 root root 0 Aug 31 10:30 /root/example.txt
```

Explanation:
- `rwxr-----`: The owner (root) has read, write, and execute permissions. The group has read permission, and others have no permissions.
- `root root`: The first `root` is the file owner, and the second `root` is the group.

Step 4: Switch to the Student User

Now, switch to the `student` user:

```
su - student
```

Step 5: Try to Access the File as the Student User

As the `student` user, attempt to view the file's contents or check its permissions:

A. Listing the File:
If the student tries to list the file's permissions:

```
ls -l /root/example.txt
```

Output:

```
ls: cannot access '/root/example.txt': Permission denied
```

Since the `student` user doesn't have permission to access files in the `/root` directory, they will receive a "Permission denied" message.

B. Trying to Read the File:

If the student tries to read the file:

```
cat /root/example.txt
```
Output:

```
cat: /root/example.txt: Permission denied
```

Again, since the `student` user does not have read access to the file, they will receive a "Permission denied" error.

- Root User: Created the file `/root/example.txt` and assigned `rwx` permissions to the owner, `r--` to the group, and no permissions to others.
- Student User: Cannot access or read the file because it is located in the `/root` directory, which is restricted to the root user by default.

Special permissions:
Special permissions in Linux, such as `setuid`, `setgid`, and `sticky bit`, provide advanced control over how users and processes interact with files and directories. Here's an explanation of each with examples:

1. Setuid (Set User ID)
When the setuid bit is set on an executable file, it allows users to run that file with the permissions of the file's owner.

Example:
- Let's say we have a program `example.sh` owned by the `root` user.
```
-rwxr-xr-x 1 root root 1234 Aug 31 12:00 example.sh
```

- To set the setuid bit on this file:

```
#chmod u+s example.sh
```

- The permissions now look like this:

```
-rwsr-xr-x 1 root root 1234 Aug 31 12:00 example.sh
```

- Result:When any user runs `example.sh`, it runs with `root` privileges.

2. Setgid (Set Group ID)
When the setgid bit is set on a directory, files created within the directory inherit the directory's group, rather than the user's default group. For executables, it allows users to run the file with the group's privileges.

Example:
- Assume you have a directory called `shared_folder` and want all files created inside it to belong to the `developers` group:

```
#chgrp developers shared_folder
#chmod g+s shared_folder
```

- The directory permissions now look like this:

```
drwxrwsr-x 2 username developers 4096 Aug 31 12:00 shared_folder
```

- Result: Any file created in `shared_folder` will automatically belong to the `developers` group.

3. Sticky Bit
When the sticky bit is set on a directory, only the file's owner, the directory's owner, or the root user can delete or rename files within the directory.

Example:
- Commonly used on directories like `/tmp` where multiple users have write access.

```
#chmod +t /tmp
```

- The directory permissions now look like this:

```
drwxrwxrwt 10 root root 4096 Aug 31 12:00 /tmp
```

- Result:In `/tmp`, users can create and edit their own files, but they cannot delete or rename files owned by others.

- setuid (`u+s`): Runs the file with the permissions of the file owner.
- setgid (`g+s`): Inherit the group of the directory for new files, or run executables with group permissions.
- sticky bit (`+t`): Only the file's owner can delete or rename files in the directory.

Setting and Checking Special Permissions

- To set special permissions using numeric notation:

  - setuid: `chmod 4755 example.sh`
  - setgid: `chmod 2755 shared_folder`
  - sticky bit: `chmod 1777 /tmp`

- To view these special permissions:

ls -ld /path/to/directory_or_file
—---

Chapter - 5

SELINUX

Introduction to SELinux

What is SELinux?

SELinux stands for Security-Enhanced Linux. It is a security module integrated into the Linux kernel that provides an additional layer of security through access controls.

Why Use SELinux?

Traditional Linux permissions (like `rwx` permissions) can control access to files and directories. However, SELinux provides fine-grained access controls, even if someone manages to bypass traditional permissions. It's particularly useful in environments requiring high security, such as servers, to protect against unauthorised access.

1.Modes of Operation

Enforcing: Enforce follows the strict rules it only allows the authorised access and policies and also stored in logs.

Permissive:Permissive allows both authorised and unauthorised access and policies and also stored in logs.

Disabled: SELinux is turned off entirely.

Temporary method:

#getenforce —> to get which mode is enabled

Enforcing

#setenforce 0 —> Set a enforce as a permissive mode

Permissive

#setenforce 1 —> Set a enforce as a enforcing mode

Enforcing

For Permanent method we will edit in the config file of selinux

#vim /etc/selinux/config

SELINUX=enforcing or permissive or disable

Save the file :wq!

Now reboot the machine

#reboot or #init 6

2. Ports

Configuring HTTPD (Web Server) with SELinux and Firewall for Enhanced Security

Scenario:

You are running a web server using the `httpd` service on your system. To secure it, you need to ensure that the service is correctly configured with both SELinux and the firewall, particularly if you plan to run the web server on a non-standard port.

Default HTTP Ports:
- HTTP (Unencrypted Web Traffic): Port 80
- HTTPS (Encrypted Web Traffic): Port 443

Steps to Secure HTTPD with SELinux and Firewall

1. Running HTTPD on a Non-Standard Port (e.g., Port 8080)

1. Change the HTTPD Configuration to Use a New Port
- Edit the HTTPD configuration file to specify the new port.

#dnf install —> this command is used to install packages or service from repos

```
  #dnf install httpd —> Install the httpd services

     #vim /etc/httpd/conf/httpd.conf
```

- Locate the line that defines the `Listen` directive and change it to the desired port, e.g., port 8080:

```
     Listen 8080
```

- Save the file and restart the HTTPD service:

```
     #systemctl restart httpd
```

2. Configure the Firewall to Allow the New Port

- Add the new port (8080 in this example) to the firewall to allow traffic through:

```
     #firewall-cmd --permanent --add-port=8080/tcp
     #firewall-cmd --reload   —> reload firewall that only port will be added
```

- Verify the firewall rules:

```
     #firewall-cmd --list-all
```

3. Configure SELinux to Allow the New Port

- By default, SELinux allows HTTPD to use certain ports. You'll need to add the new port to the allowed list for HTTPD.
- Check the currently allowed ports for HTTPD:

```
     #semanage port -l —> this command is used to list all enable port in selinux so we user grep search particular port from selinux

     #semanage port -l | grep http_port_t
```

- Add the new port (8080) to SELinux for HTTPD:

```
      #semanage port -a -t http_port_t -p tcp 8080
```

- Verify the addition:

```
      #semanage port -l | grep http_port_t
```

2. Importance of Configuring Both Firewall and SELinux

- Firewall:

  - Controls network traffic and can block unauthorized access to the web server.
- SELinux:
  - Enforces security policies, ensuring that the `httpd` service can only bind to ports that are explicitly allowed.

- Why Change the Default Port?
  - Changing the default port from 80 to a non-standard port like 8080 can help avoid automated attacks that target well-known service ports.

Example: Testing the Configuration
- After configuring the server, test it by accessing the web server through the new port. For example:

```
#hostname -i  —> to identified the ip for our machine
```

```
#curl 172.25.250.9:8080/filename
```

 - In a web browser, navigate to
 http://your-server-ip:8080

  - If everything is set up correctly, you should see your web page.

3.Context

SELinux Context

Analogy: Office Building Security

1. The Office Building: Think of SELinux as a security system in a large office building with various restricted areas.

2. Office Zones:
   - Zone 1 (Root Directory): High-security area (e.g., administrative offices).
   - Zone 2 (/var/www): Specific department's area (e.g., web development department).

3. Security Badges (SELinux Labels):
   - Zone 1 Badge: Allows access to administrative offices.
   - Zone 2 Badge: Allows access to the web development department.

4. Access Control:
   - Employees with the Zone 1 badge can't access Zone 2.
   - Only employees with the correct Zone 2 badge can access files there.

Commands and Steps

1. Check SELinux Status:

```
#sestatus
```

- This command shows if SELinux is enabled and its mode (Enforcing, Permissive, or Disabled).

2. Create a File in the Root Directory (Zone 1):

```
#touch /root/testfile
```

Check the SELinux Context of the File**:

```
#ls -Z /root/testfile
```

- Expected Output: `system_u:object_r:root_t:s0`
  - Explanation: The label shows that the file is in the root directory and is subject to strict security policies.

3. Create a File in `/var/www` Directory (Zone 2):

Create and Edit the HTML File

1.Open the Terminal

Navigate to the Directory:
```
#cd /var/www/html
```
Create and Edit an HTML File Using vim:

```
#vim testfile.html
```

Add Content in vim:

Press i to enter **insert mode**.

```
<!DOCTYPE html>

<html>
<head>
   <title>Test Page</title>
</head>
<body>
   <h1>Hello, SELinux!</h1>
   <p>This is a test page to demonstrate file access and SELinux context.</p>
```

```
</body>
</html>
```

Press Esc to exit **insert mode**.

Type :wq and press Enter to **write** and **quit**.

Verify the File Content:

```
cat testfile.html
```

2. Access the File via Web Browser

Open Firefox and go to http://localhost/testfile.html.

Expected Result: The HTML content should be displayed correctly.

Notes: If You add change the port you give ip like this

```
#curl 172.25.250.9:8080/textfile.html
```

- In a web browser, navigate to
http://your-server-ip:8080/textfile.html

- If everything is set up correctly, you should see your web page.

3. Check the Initial SELinux Label

Check the SELinux Context:

```
ls -Z /var/www/html/testfile.html
```

Expected Output: The label should be system_u:object_r:httpd_sys_content_t:s0

Change the Label for the file

```
#cd /var/www/html
```

```
#ls -lZ
```

Output = system_u:object_r:httpd_sys_content_t:s0

Change the label

```
#chcon system_u:object_r:admin_home_t:s0 testfile.html
```

I change the label for the file

```
#ls -lZ
```

Output = system_u:object_r:admin_home_t:s0

Now check the output will show in the firefox or not

```
#curl ip_addr:port/testfile.html
```

Output : forbidden error

In a web browser, navigate to
http://your-server-ip:8080/textfile.html

  - If everything is not set up correctly, you should see your web page label is not correctly assign

Because our mode is enforcing, I change to enforce to permissive

```
#setenforce 0
```

```
#curl ip_addr:port/testfile.html
```

In a web browser, navigate to
http://your-server-ip:8080/textfile.html

  - If everything is set up correctly, you should see your web page.

—---