Independent Skill Development Mission

Independent Skill Development NXT Mission
EMPOWERING FUTURES, SHAPING TOMORROW

ISDM (INDEPENDENT SKILL DEVELOPMENT MISSION

LINUX FILE SYSTEM SECURITY

CHAPTER 1: INTRODUCTION TO LINUX FILE SYSTEM SECURITY

The **Linux file system** is a core component of the operating system that stores and organizes files efficiently. Ensuring **file system security** is essential to **protect sensitive data, prevent unauthorized access, and maintain system integrity**. Linux provides multiple mechanisms to **control access, enforce security policies, and monitor file system activities**.

Key Aspects of Linux File System Security

- **File Permissions and Ownership** – Controls access to files and directories.
- **Access Control Lists (ACLs)** – Provides more granular permission control.
- **Immutable File Attributes** – Prevents accidental modifications or deletions.
- **Disk Encryption** – Protects sensitive data at rest.
- **Audit and Logging** – Monitors unauthorized file access attempts.

By implementing **proper file system security**, Linux users and administrators can **prevent unauthorized data breaches, protect system integrity, and ensure compliance with security standards**.

CHAPTER 2: UNDERSTANDING FILE PERMISSIONS AND OWNERSHIP

1. Linux File Permissions

Linux uses a permission-based system to control **who can read, write, and execute files**. Each file and directory has three sets of permissions:

- **Owner** (u) – The user who created the file.
- **Group** (g) – A set of users who share access to the file.
- **Others** (o) – All other users on the system.

The permissions are represented in **symbolic (rwx) or numeric (chmod) formats**.

2. Viewing File Permissions

To check file permissions, use:

```
ls -l
```

Example output:

```
-rw-r--r--  1 alice  users  1024 Jan 1 12:00 file.txt
```

- rw- → Owner can **read and write**.
- r-- → Group members can **only read**.
- r-- → Others can **only read**.

3. Changing File Permissions (chmod)

To modify file permissions:

```
chmod 750 file.txt
```

This means:

- **Owner**: Read, write, execute (rwx).
- **Group**: Read and execute (r-x).
- **Others**: No access (---).

To apply permissions recursively:

```
chmod -R 755 /var/www/
```

4. Changing File Ownership (chown)

To change the **owner** of a file:

```
sudo chown bob file.txt
```

To change both **owner and group**:

```
sudo chown bob:developers file.txt
```

Chapter 3: Advanced Access Control with ACLs

1. What is ACL (Access Control List)?

The standard permission model (chmod) can be **limited** in certain cases. **ACLs** allow administrators to grant permissions to **specific users or groups** beyond the traditional model.

2. Viewing ACL Permissions

To check if a file has ACLs applied:

```
getfacl file.txt
```

Example output:

```
# file: file.txt
# owner: alice

# group: users

user::rw-

user:bob:r--

group::r--

mask::r--

other::---
```

3. Setting ACL Permissions

To **grant read access** to user bob:

```
setfacl -m u:bob:r file.txt
```

To **grant full access** to a group:

```
setfacl -m g:developers:rwx project/
```

To **remove ACL permissions**:

```
setfacl -x u:bob file.txt
```

4. Making ACL Changes Persistent

To apply ACL settings to **new files in a directory**, use:

```
setfacl -m d:u:bob:rwX /data/
```

This ensures that **future files** in /data/ inherit the same ACL settings.

Chapter 4: Securing Files with Immutable Attributes

1. What Are File Attributes?

Apart from standard permissions, Linux provides **special attributes** that add **extra security layers** to files.

2. Making a File Immutable

An **immutable file** cannot be modified, renamed, or deleted (even by root).

```
sudo chattr +i config.conf
```

To verify the attribute:

```
lsattr config.conf
```

To remove immutability:

```
sudo chattr -i config.conf
```

3. Preventing File Deletion

To **protect a log file from accidental deletion**:

```
sudo chattr +a /var/log/secure.log
```

This allows the file to be **appended** but not deleted.

Chapter 5: Encrypting Files and Partitions for Security

1. Why Use Encryption?

Encryption protects sensitive data from **unauthorized access** in case of system compromise.

2. Encrypting Files with gpg

To encrypt a file:

```
gpg -c confidential.txt
```

This creates confidential.txt.gpg, requiring a **password** to decrypt.

To decrypt:

```
gpg confidential.txt.gpg
```

3. Encrypting a Partition with LUKS

To encrypt a partition /dev/sdb1:

```
sudo cryptsetup luksFormat /dev/sdb1
```

To unlock it:

```
sudo cryptsetup luksOpen /dev/sdb1 encrypted_disk
```

4. Mounting an Encrypted Drive

```
sudo mount /dev/mapper/encrypted_disk /mnt/secure
```

This ensures that **data remains encrypted when not mounted**.

CHAPTER 6: MONITORING FILE ACCESS AND SECURITY AUDITING

1. Logging File Access Attempts

To **track access to a sensitive file**, use auditd:

```
sudo auditctl -w /etc/passwd -p wa -k password_changes
```

To check logs:

```
sudo ausearch -k password_changes
```

2. Detecting Unauthorized Changes with tripwire

Tripwire helps **detect changes to critical files**.

To install:

```
sudo apt install tripwire -y  # Debian/Ubuntu
```

Initialize the database:

```
sudo tripwire --init
```

Run a check:

```
sudo tripwire --check
```

Case Study – Securing a Shared Linux Server

Scenario:

A **university IT department** provides Linux servers for students and staff. They need to:

1. **Protect critical system files** from modification.
2. **Allow students to access shared folders** securely.
3. **Prevent unauthorized deletion of research data.**

Solution:

1. **Restrict system files** using chattr +i /etc/passwd.
2. **Use ACLs** to grant read-only access to students:
3. setfacl -m g:students:r /research_data/
4. **Enable auditing** for unauthorized file modifications:
5. sudo auditctl -w /var/www/html -p wa -k web_files

Outcome:

- **Unauthorized modifications are prevented.**

- **Students can access but not modify important files.**
- **File changes are monitored for security compliance.**

CHAPTER 7: EXERCISE

1. **Set file permissions (chmod) for a directory so only the owner has full access.**
2. **Apply ACL permissions to allow user1 read access to confidential.txt.**
3. **Use chattr to make a configuration file immutable.**
4. **Encrypt a file using gpg and decrypt it.**
5. **Enable auditing to monitor access to /etc/shadow.**

CONCLUSION

Linux **file system security** is crucial for protecting **sensitive data, preventing unauthorized modifications, and ensuring compliance with security standards**

Understanding SELinux and AppArmor

Chapter 1: Introduction to SELinux and AppArmor

What Are SELinux and AppArmor?

Security-Enhanced Linux (SELinux) and **AppArmor (Application Armor)** are two security frameworks in Linux that provide **mandatory access control (MAC)**, which is stricter than traditional discretionary access control (DAC).

Why Are SELinux and AppArmor Important?

- **Enhance system security** by restricting applications to their required permissions.
- **Prevent unauthorized access** and reduce the impact of compromised applications.
- **Protect system integrity** by enforcing security policies beyond file permissions and user roles.

SELinux vs. AppArmor

Feature	**SELinux**	**AppArmor**
Developed By	NSA	Canonical (Ubuntu)
Security Model	Label-based	Profile-based
Configuration Complexity	Complex	Easier
Flexibility	High	Moderate

Default In	RHEL, CentOS, Fedora	Ubuntu, Debian

This chapter will provide **detailed explanations, examples, and case studies** on how **SELinux and AppArmor** work, how to configure them, and how to troubleshoot issues.

Chapter 2: Understanding SELinux (Security-Enhanced Linux)

1. What is SELinux?

SELinux is a **mandatory access control (MAC) system** that assigns **security labels** to files, processes, and users. Unlike traditional Linux permissions, **SELinux policies control what actions processes can perform, even for root users**.

2. SELinux Modes

SELinux operates in three modes:

Mode	**Description**
Enforcing	SELinux policies are enforced (default on RHEL-based systems).
Permissive	SELinux logs policy violations but does not enforce them.
Disabled	SELinux is completely turned off.

To check the current mode:

```
getenforce
```

To set SELinux to permissive mode:

sudo setenforce 0

To re-enable enforcing mode:

sudo setenforce 1

CHAPTER 3: CONFIGURING AND MANAGING SELINUX POLICIES

1. Understanding SELinux Contexts

Every file and process in SELinux has **a security label** (context).

To check a file's SELinux context:

ls -Z /var/www/html/

Example output:

-rw-r--r--. root root system_u:object_r:httpd_sys_content_t:so index.html

Part	Meaning
system_u	User (unprivileged system user)
object_r	Role
httpd_sys_content_t	Type (specific to web servers)

2. Changing SELinux Contexts (chcon)

To change a file's SELinux context manually:

sudo chcon -t httpd_sys_content_t /var/www/html/index.html

3. Using SELinux Boolean Settings (setsebool)

Boolean settings allow enabling/disabling specific SELinux security policies.

To check all SELinux Booleans:

```
sudo getsebool -a
```

To enable a Boolean setting (e.g., allowing HTTPD to connect to the network):

```
sudo setsebool -P httpd_can_network_connect on
```

CHAPTER 4: UNDERSTANDING AND MANAGING APPARMOR

1. What is AppArmor?

AppArmor is an alternative **mandatory access control system** that works by **restricting application access using profiles**. Instead of labeling every file like SELinux, AppArmor uses predefined **profiles** that define what an application can and cannot do.

2. Checking AppArmor Status

To check if AppArmor is active:

```
sudo aa-status
```

Example output:

```
apparmor module is loaded.

8 profiles are in enforce mode.
```

3. Enforcing, Complaining, and Disabled Modes in AppArmor

AppArmor operates in three modes:

Mode	**Description**
Enforcing	Blocks and logs policy violations.

Complain	Logs violations but does not block them.
Disabled	AppArmor is turned off.

To set an application profile to **complain mode**:

```
sudo aa-complain /etc/apparmor.d/usr.sbin.apache2
```

To **enforce** the profile:

```
sudo aa-enforce /etc/apparmor.d/usr.sbin.apache2
```

CHAPTER 5: MANAGING APPARMOR PROFILES

1. Listing Available AppArmor Profiles

To list all AppArmor profiles:

```
sudo apparmor_parser -L
```

2. Creating and Modifying Profiles

To create a new AppArmor profile:

```
sudo aa-genprof /usr/bin/customapp
```

Follow the interactive prompts to define access rules for the application.

3. Loading and Unloading Profiles

To manually load an AppArmor profile:

```
sudo apparmor_parser -r /etc/apparmor.d/usr.sbin.apache2
```

To remove an AppArmor profile:

```
sudo apparmor_parser -R /etc/apparmor.d/usr.sbin.apache2
```

Chapter 6: Case Study – Securing a Web Server with SELinux and AppArmor

Scenario:

A company hosts its website on **Apache HTTP Server**, and they want to **secure it against unauthorized access and attacks**.

Solution Using SELinux:

1. **Ensure SELinux is in enforcing mode:**
2. sudo setenforce 1
3. **Check and assign correct SELinux context to web files:**
4. sudo chcon -t httpd_sys_content_t /var/www/html/index.html
5. **Allow Apache to make network connections using Boolean:**
6. sudo setsebool -P httpd_can_network_connect on

Solution Using AppArmor:

1. **Enable AppArmor and check active profiles:**
2. sudo aa-status
3. **Put Apache into enforcing mode:**
4. sudo aa-enforce /etc/apparmor.d/usr.sbin.apache2
5. **Modify Apache profile to allow additional permissions (if needed):**
6. sudo nano /etc/apparmor.d/usr.sbin.apache2

Add:

/var/www/html/ r,

/var/www/html/** rw,

7. **Restart AppArmor to apply changes:**
8. sudo systemctl restart apparmor

Outcome:

- **SELinux prevents unauthorized access to web files.**
- **AppArmor restricts Apache's actions, minimizing security risks.**
- **The web server is now more secure against potential attacks.**

CHAPTER 7: EXERCISE

1. **Check if SELinux is in enforcing mode and list all active contexts.**
2. **Change the SELinux context of /var/www/html/index.html to httpd_sys_content_t.**
3. **Enable Apache networking in SELinux using Booleans.**
4. **List all AppArmor profiles and enforce a profile for Nginx.**
5. **Put an AppArmor profile into complain mode and check logs for violations.**

CONCLUSION

SELinux and AppArmor provide **robust security layers** in Linux, preventing applications from **accessing unauthorized resources** and **minimizing attack surfaces**. While **SELinux uses label-based access control**, **AppArmor applies rule-based security profiles**. Mastering these tools ensures **better system security and compliance with industry standards**.

User Authentication and PAM (Pluggable Authentication Modules)

Chapter 1: Introduction to User Authentication and PAM

What is User Authentication in Linux?

User authentication is the process of **verifying a user's identity** before granting access to a Linux system. Authentication mechanisms ensure that **only authorized users can access system resources**, preventing unauthorized access and security breaches.

What is PAM (Pluggable Authentication Modules)?

PAM (Pluggable Authentication Modules) is a flexible authentication framework in Linux that allows administrators to **configure and customize authentication methods**. PAM provides:

- **Multiple authentication mechanisms** (passwords, biometrics, smart cards).
- **Centralized authentication management** for different services.
- **Enhanced security policies**, such as password complexity rules and account locking.

Why is User Authentication Important?

- Prevents **unauthorized access** to system resources.
- Protects **sensitive information** from security threats.
- Enforces **password policies** to strengthen security.

This chapter covers **user authentication methods, PAM configuration, and security best practices** for Linux authentication.

CHAPTER 2: UNDERSTANDING USER AUTHENTICATION IN LINUX

1. User Accounts in Linux

Linux user accounts are categorized into:

- **Root user (UID 0)** – The superuser with full system control.
- **Regular users (UID 1000+)** – Standard user accounts with limited privileges.
- **System users (UID <1000)** – Accounts used for system processes and services.

To list users:

```
cat /etc/passwd
```

Each line follows this format:

```
username:x:UID:GID:Full Name:/home/username:/bin/bash
```

2. Password Storage and Hashing

User passwords are stored **securely in /etc/shadow** using a **hashed format**.
To view hashed passwords:

```
sudo cat /etc/shadow
```

Example entry:

```
alice:$6$7dGf1sKj$X9G80Jw2P/8uy57D1k3Y6/.:18753:0:99999:7:::
```

- $6$ → **SHA-512 encryption method**
- 7dGf1sKj → **Salt value to prevent hash collisions**

- X9G80Jw2P... → **Hashed password**

3. Creating and Managing Users

To create a new user:

sudo useradd -m -s /bin/bash john

To set a password:

sudo passwd john

To lock a user account:

sudo passwd -l john

To unlock a user:

sudo passwd -u john

CHAPTER 3: INTRODUCTION TO PAM (PLUGGABLE AUTHENTICATION MODULES)

1. What is PAM?

PAM is a **modular authentication system** that integrates with **various services like SSH, sudo, login, and graphical logins**. PAM modules are defined in the /etc/pam.d/ directory.

2. PAM Configuration Files

Each service (e.g., SSH, login, sudo) has a corresponding file in /etc/pam.d/, such as:

ls /etc/pam.d/

Example output:

common-auth common-password common-session sshd sudo login

3. Structure of a PAM Configuration File

A PAM file contains **four control types**:

Control	**Description**
auth	Verifies user identity (e.g., passwords, biometrics).
account	Checks account policies (e.g., expiration, access control).
password	Manages password changes and complexity.
session	Defines actions before/after login (e.g., session limits).

Example PAM rule in /etc/pam.d/sshd:

auth required pam_unix.so

- auth → Authentication module.
- required → Must pass for authentication to succeed.
- pam_unix.so → Uses traditional Linux password authentication.

CHAPTER 4: ENFORCING STRONG AUTHENTICATION POLICIES WITH PAM

1. Enforcing Password Complexity

To enforce strong passwords, modify:

sudo nano /etc/security/pwquality.conf

Set policies:

```
minlen = 12      # Minimum password length
dcredit = -1     # At least one digit required
ucredit = -1     # At least one uppercase letter required
lcredit = -1     # At least one lowercase letter required
ocredit = -1     # At least one special character required
```

To apply the policy:

```
sudo nano /etc/pam.d/common-password
```

Add:

```
password requisite pam_pwquality.so retry=3
```

This forces users to create **strong passwords** with a minimum length and complexity.

---

2. Implementing Account Lockout for Failed Logins

To prevent brute-force attacks, configure account lockout in:

```
sudo nano /etc/pam.d/common-auth
```

Add:

```
auth required pam_tally2.so deny=3 unlock_time=600
```

- deny=3 → **Locks the account after 3 failed attempts**.
- unlock_time=600 → **Unlocks the account after 10 minutes**.

To check failed attempts:

```
pam_tally2 --user john
```

To reset lockout:

```
pam_tally2 --user john --reset
```

---

3. Restricting User Access Based on Time

To allow john to log in only between 9 AM and 5 PM:

```
sudo nano /etc/security/time.conf
```

Add:

```
login;*;john;Al0900-1700
```

This **restricts login to working hours**.

CHAPTER 5: MULTI-FACTOR AUTHENTICATION (MFA) WITH PAM

1. Installing Google Authenticator for MFA

To enable **two-factor authentication (2FA)**:

```
sudo apt install libpam-google-authenticator -y
```

Run the configuration:

```
google-authenticator
```

Answer the prompts and save the **QR code or secret key**.

2. Enforcing MFA for SSH Logins

Edit PAM's SSH authentication file:

```
sudo nano /etc/pam.d/sshd
```

Add:

```
auth required pam_google_authenticator.so
```

Edit the SSH configuration:

```
sudo nano /etc/ssh/sshd_config
```

Set:

```
ChallengeResponseAuthentication yes
```

Restart SSH:

```
sudo systemctl restart ssh
```

Now, SSH logins require **both a password and a one-time verification code**.

Case Study – Implementing Secure Authentication in a Corporate Environment

Scenario:

A company needs to **improve authentication security** by enforcing **strong passwords, failed login protection, and multi-factor authentication**.

Solution:

1. **Enforce strong passwords** using pam_pwquality.so.
2. **Lock accounts** after 3 failed attempts using pam_tally2.so.
3. **Enable two-factor authentication (2FA)** with pam_google_authenticator.so.
4. **Restrict login times** for employees using time.conf.

Outcome:

- **Users cannot set weak passwords.**
- **Accounts are locked after multiple failed login attempts.**
- **2FA prevents unauthorized access.**
- **System security is significantly enhanced.**

CHAPTER 6: EXERCISE

1. **Create a new user (user1) and set a strong password policy.**
2. **Configure account lockout after 3 failed login attempts.**
3. **Enable MFA for SSH using Google Authenticator.**
4. **Restrict a user to login only during office hours.**
5. **Monitor failed authentication attempts using auth.log.**

CONCLUSION

Linux authentication security relies on **user management and PAM configuration**. By enforcing **strong passwords, account lockout policies, and multi-factor authentication**, administrators can **protect system resources from unauthorized access**. Mastering PAM allows **fine-grained control over authentication processes**, ensuring **enhanced security and compliance**.

CONFIGURING FIREWALLS (IPTABLES, UFW)

CHAPTER 1: INTRODUCTION TO FIREWALLS IN LINUX

What is a Firewall?

A **firewall** is a security mechanism that controls **incoming and outgoing network traffic** based on predefined rules. It acts as a barrier between a **trusted internal network** and **untrusted external networks** (e.g., the internet), preventing unauthorized access and ensuring network security.

Types of Firewalls in Linux

Linux offers multiple firewall tools, including:

- **iptables** – A powerful but complex command-line firewall for packet filtering.
- **UFW (Uncomplicated Firewall)** – A user-friendly front-end for managing iptables (default in Ubuntu).
- **firewalld** – A dynamic firewall system used in **RHEL/CentOS**.

Why Are Firewalls Important?

- **Protects against unauthorized access** by filtering network traffic.
- **Prevents cyber attacks**, such as DDoS and port scanning.
- **Regulates outbound connections** to block malicious applications.

This chapter covers **configuring iptables and UFW**, their rules, examples, and case studies.

CHAPTER 2: UNDERSTANDING IPTABLES FIREWALL

1. What is iptables?

iptables is a **packet filtering firewall** that controls network traffic **based on rules applied to different chains**.

2. iptables Chains and Rules

iptables works with three main **chains**:

Chain	Function
INPUT	Controls incoming traffic to the server.
FORWARD	Controls traffic passing through the server.
OUTPUT	Controls outgoing traffic from the server.

3. Checking Current iptables Rules

To list active firewall rules:

```
sudo iptables -L -v
```

To view rules with line numbers:

```
sudo iptables -L --line-numbers
```

4. Allowing and Blocking Traffic with iptables

Allow SSH (Port 22)

```
sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT
```

Allow HTTP and HTTPS Traffic

```
sudo iptables -A INPUT -p tcp --dport 80 -j ACCEPT
```

```
sudo iptables -A INPUT -p tcp --dport 443 -j ACCEPT
```

Block an IP Address

sudo iptables -A INPUT -s 192.168.1.100 -j DROP

Drop All Incoming Traffic (Default Policy)

sudo iptables -P INPUT DROP

5. Saving iptables Rules

To make iptables rules persistent after reboot:

- **On Debian/Ubuntu:**
- sudo apt install iptables-persistent -y
- sudo netfilter-persistent save
- **On RHEL/CentOS:**
- sudo service iptables save

6. Resetting iptables Rules

To delete all rules:

sudo iptables -F

To reset policies:

sudo iptables -P INPUT ACCEPT

sudo iptables -P OUTPUT ACCEPT

sudo iptables -P FORWARD ACCEPT

Chapter 3: Configuring UFW (Uncomplicated Firewall)

1. What is UFW?

UFW (Uncomplicated Firewall) is a **simplified interface** for managing iptables rules. It is the **default firewall** in **Ubuntu**.

2. Checking Firewall Status

To check if UFW is enabled:

```
sudo ufw status
```

If UFW is **inactive**, enable it:

```
sudo ufw enable
```

3. Allowing and Denying Traffic with UFW

Allow SSH Access

```
sudo ufw allow ssh
```

or

```
sudo ufw allow 22/tcp
```

Allow HTTP and HTTPS Traffic

```
sudo ufw allow http
```

```
sudo ufw allow https
```

Block a Specific IP Address

```
sudo ufw deny from 192.168.1.100
```

Allow Specific IP to Access SSH

```
sudo ufw allow from 192.168.1.50 to any port 22
```

4. Setting Default Policies

By default, deny all incoming traffic and allow outgoing traffic:

```
sudo ufw default deny incoming
```

```
sudo ufw default allow outgoing
```

5. Enabling Logging in UFW

To monitor UFW activity:

```
sudo ufw logging on
```

To check logs:

```
sudo tail -f /var/log/ufw.log
```

6. Disabling or Resetting UFW

To disable UFW:

```
sudo ufw disable
```

To reset all rules:

```
sudo ufw reset
```

Chapter 4: Advanced Firewall Configurations

1. Limiting SSH Login Attempts (Brute Force Protection)

To prevent brute-force SSH attacks, limit the number of connection attempts:

With iptables

```
sudo iptables -A INPUT -p tcp --dport 22 -m limit --limit 3/min --limit-burst 5 -j ACCEPT
```

With UFW

```
sudo ufw limit ssh
```

This allows **only 6 SSH connection attempts within 30 minutes**.

---

2. Port Forwarding with iptables

To forward traffic from port **8080 to port 80**:

```
sudo iptables -t nat -A PREROUTING -p tcp --dport 8080 -j REDIRECT --to-port 80
```

---

3. Blocking a Country Using iptables

To block all traffic from a country (e.g., China), use **GeoIP filtering**:

```
sudo iptables -A INPUT -m geoip --src-cc CN -j DROP
```

(Requires the xtables-addons package).

Chapter 5: Case Study – Securing a Web Server with a Firewall

Scenario:

A company hosts a **web server** (Apache) and needs to secure it by:

- **Allowing web traffic (HTTP, HTTPS).**
- **Restricting SSH access to a specific IP.**
- **Blocking unauthorized network scans.**
- **Limiting SSH login attempts to prevent brute-force attacks.**

Solution Using iptables

1. **Allow only HTTP, HTTPS, and SSH from a specific IP:**

2. sudo iptables -A INPUT -p tcp --dport 80 -j ACCEPT
3. sudo iptables -A INPUT -p tcp --dport 443 -j ACCEPT
4. sudo iptables -A INPUT -p tcp --dport 22 -s 203.0.113.5 -j ACCEPT
5. **Drop all unauthorized incoming traffic:**
6. sudo iptables -P INPUT DROP
7. **Limit SSH login attempts:**
8. sudo iptables -A INPUT -p tcp --dport 22 -m limit --limit 3/min --limit-burst 5 -j ACCEPT

Solution Using UFW

1. **Enable UFW and allow essential services:**
2. sudo ufw enable
3. sudo ufw allow http
4. sudo ufw allow https
5. sudo ufw allow from 203.0.113.5 to any port 22
6. **Deny unauthorized SSH connections:**
7. sudo ufw deny ssh
8. **Enable logging to monitor blocked traffic:**
9. sudo ufw logging on

Outcome:

- The **web server is secured**, allowing only essential traffic.
- **Unauthorized SSH login attempts are blocked.**

- **Firewall logs provide insights into potential attacks.**

CHAPTER 6: EXERCISE

1. **Set up an iptables rule to allow only SSH (port 22) and block all other traffic.**
2. **Use UFW to deny access to port 3306 (MySQL) from external networks.**
3. **Enable rate limiting for SSH using UFW.**
4. **Log all dropped packets in iptables.**
5. **Reset the firewall and configure new rules for a secure server setup.**

CONCLUSION

Firewalls are **essential** for securing Linux systems. By configuring **iptables or UFW**, administrators can **control traffic, prevent cyber threats, and protect critical services**.

VPN AND SECURE REMOTE ACCESS

CHAPTER 1: INTRODUCTION TO VPN AND SECURE REMOTE ACCESS

What is a VPN?

A **Virtual Private Network (VPN)** is a secure **tunnel** between two networks or devices over the internet. It encrypts data, ensuring **confidentiality, integrity, and authentication** in remote communications.

Why Use a VPN?

- **Encrypts internet traffic** to prevent data interception.
- **Allows secure remote access** to private networks.
- **Hides IP addresses** for privacy and anonymity.
- **Bypasses geo-restrictions** to access blocked content.

Types of VPNs

VPN Type	**Description**
Remote Access VPN	Allows users to securely connect to a private network from a remote location.
Site-to-Site VPN	Connects entire networks securely over the internet.
SSL VPN	Uses web browsers for secure access without installing a VPN client.
IPSec VPN	Uses the IPSec protocol for secure encryption and authentication.

This chapter covers **VPN protocols, installation, configuration, and security best practices** for **OpenVPN, WireGuard, and IPSec VPNs**.

Chapter 2: Understanding VPN Protocols

1. Common VPN Protocols

Protocol	Description	Encryption
OpenVPN	Open-source VPN protocol using SSL/TLS encryption.	AES-256
WireGuard	Modern, lightweight VPN with strong security.	ChaCha20
IPSec	Encrypts IP packets for secure communication.	AES-256
L2TP/IPSec	Layer 2 Tunneling Protocol with IPSec security.	AES-256
PPTP	Outdated but simple VPN with weak encryption.	MPPE

2. Choosing the Right VPN Protocol

- **OpenVPN** – Best for security and flexibility.
- **WireGuard** – Fastest VPN with modern encryption.
- **IPSec** – Enterprise-grade VPN with strong encryption.
- **L2TP/IPSec** – Used when OpenVPN is blocked.
- **PPTP** – Should be avoided due to weak encryption.

CHAPTER 3: SETTING UP OPENVPN ON A LINUX SERVER

1. Installing OpenVPN

- **On Debian/Ubuntu:**
- sudo apt update
- sudo apt install openvpn easy-rsa -y
- **On CentOS/RHEL:**
- sudo yum install epel-release -y
- sudo yum install openvpn easy-rsa -y

2. Configuring the OpenVPN Server

Create an OpenVPN directory:

mkdir -p /etc/openvpn/server/

cp -r /usr/share/easy-rsa /etc/openvpn/

cd /etc/openvpn/easy-rsa

Initialize the Public Key Infrastructure (PKI):

./easyrsa init-pki

./easyrsa build-ca

./easyrsa gen-req server nopass

./easyrsa sign-req server server

./easyrsa gen-dh

Generate client certificates:

```
./easyrsa gen-req client1 nopass

./easyrsa sign-req client client1
```

3. Creating the OpenVPN Server Configuration File

Edit the OpenVPN config file:

```
sudo nano /etc/openvpn/server/server.conf
```

Add:

```
port 1194

proto udp

dev tun

ca /etc/openvpn/easy-rsa/pki/ca.crt

cert /etc/openvpn/easy-rsa/pki/issued/server.crt

key /etc/openvpn/easy-rsa/pki/private/server.key

dh /etc/openvpn/easy-rsa/pki/dh.pem

server 10.8.0.0 255.255.255.0

persist-key

persist-tun
```

Enable and start the OpenVPN service:

```
sudo systemctl start openvpn@server

sudo systemctl enable openvpn@server
```

4. Configuring Firewall for OpenVPN

Allow VPN traffic:

```
sudo ufw allow 1194/udp
```

Enable IP forwarding:

```
echo "net.ipv4.ip_forward = 1" | sudo tee -a /etc/sysctl.conf
sudo sysctl -p
```

Chapter 4: Setting Up WireGuard VPN

1. Installing WireGuard

- **On Debian/Ubuntu:**
- sudo apt install wireguard -y
- **On CentOS/RHEL:**
- sudo yum install epel-release -y
- sudo yum install wireguard-tools -y

2. Configuring WireGuard Server

Generate key pairs:

```
wg genkey | tee server_private.key | wg pubkey > server_public.key
wg genkey | tee client_private.key | wg pubkey > client_public.key
```

Edit the WireGuard configuration file:

```
sudo nano /etc/wireguard/wg0.conf
```

Add:

```
[Interface]
PrivateKey = <server_private.key>
```

```
Address = 10.0.0.1/24

ListenPort = 51820


[Peer]

PublicKey = <client_public.key>

AllowedIPs = 10.0.0.2/32
```

Enable and start WireGuard:

```
sudo systemctl enable wg-quick@wg0

sudo systemctl start wg-quick@wg0
```

3. Configuring Firewall for WireGuard

Allow traffic on port 51820:

```
sudo ufw allow 51820/udp
```

Chapter 5: Configuring IPSec VPN

1. Installing IPSec (StrongSwan)

- **On Debian/Ubuntu:**
- sudo apt install strongswan -y
- **On CentOS/RHEL:**
- sudo yum install strongswan -y

2. Configuring IPSec VPN Server

Edit the IPSec configuration file:

```
sudo nano /etc/ipsec.conf
```

Add:

```
config setup
   charondebug="ike 2, knl 2, cfg 2"


conn myvpn
   auto=add
   keyexchange=ikev2
   left=203.0.113.1
   leftsubnet=0.0.0.0/0
   right=%any
   rightdns=8.8.8.8
   ike=aes256-sha2_256-modp2048
   esp=aes256-sha2_256
```

Restart IPSec service:

```
sudo systemctl restart strongswan
```

Chapter 6: Case Study – Implementing Secure Remote Access for a Company

Scenario:

A company needs a **secure way for remote employees to connect** to the corporate network while ensuring:

- **Data encryption to protect communications.**
- **Only authorized users can access resources.**
- **Minimal impact on network performance.**

Solution Using OpenVPN

1. **Deploy an OpenVPN server** with strong AES-256 encryption.
2. **Use client authentication certificates** to restrict access.
3. **Configure firewall rules** to allow only VPN traffic.
4. **Enable logging** to monitor access attempts.

Outcome:

- **Remote employees securely access internal resources.**
- **VPN traffic is encrypted to prevent data interception.**
- **Unauthorized users cannot access the network.**

CHAPTER 7: EXERCISE

1. **Set up an OpenVPN server and connect a client.**
2. **Install and configure WireGuard VPN on a Linux server.**
3. **Create an IPSec VPN tunnel for secure communication.**
4. **Modify firewall rules to allow only VPN traffic.**
5. **Test VPN connection stability and log unauthorized attempts.**

CONCLUSION

VPNs provide **secure remote access** by encrypting traffic and **protecting network resources**. By configuring **OpenVPN, WireGuard, and IPSec**, administrators can **ensure privacy, security, and seamless remote connectivity**. 🚀

Intrusion Detection & Prevention Systems (IDS & IPS)

Chapter 1: Introduction to Intrusion Detection & Prevention Systems

What Are IDS and IPS?

Intrusion Detection Systems (**IDS**) and Intrusion Prevention Systems (**IPS**) are security mechanisms that **monitor and analyze network traffic** to detect and prevent unauthorized activities, cyberattacks, or policy violations.

System	Function
IDS (Intrusion Detection System)	Monitors network or host activities for suspicious behavior and alerts administrators.
IPS (Intrusion Prevention System)	Detects and actively blocks malicious activities before they reach the system.

Why Are IDS & IPS Important?

- **Detects security threats** in real time.
- **Prevents unauthorized access** and attacks.
- **Protects sensitive data** from intrusions.
- **Ensures compliance** with security policies.

Types of IDS & IPS

Type	Description

Network-based (NIDS/NIPS)	Monitors traffic at the network level.
Host-based (HIDS/HIPS)	Monitors activity on an individual host or endpoint.
Signature-based	Detects known attack patterns.
Anomaly-based	Identifies deviations from normal behavior.

This chapter covers **IDS & IPS tools such as Snort, Suricata, and OSSEC**, including installation, configuration, and best practices.

Chapter 2: Understanding Network-Based IDS & IPS (NIDS/NIPS)

1. How Network IDS & IPS Work

Network-based IDS/IPS monitors packets at the network level to identify threats. It operates by:

1. **Capturing network traffic.**
2. **Comparing packets against attack signatures or behavioral models.**
3. **Alerting or blocking malicious activities.**

2. Popular NIDS/NIPS Solutions

Tool	**Description**
Snort	Open-source IDS/IPS with signature-based detection.

Suricata	High-performance IDS/IPS with multi-threading.
Zeek (formerly Bro)	IDS with deep network analysis capabilities.

CHAPTER 3: INSTALLING AND CONFIGURING SNORT IDS/IPS

1. Installing Snort

- **On Debian/Ubuntu:**
- sudo apt update
- sudo apt install snort -y
- **On CentOS/RHEL:**
- sudo yum install epel-release -y
- sudo yum install snort -y

2. Configuring Snort

Edit the Snort configuration file:

```
sudo nano /etc/snort/snort.conf
```

Set the **network interface**:

```
ipvar HOME_NET 192.168.1.0/24
```

```
ipvar EXTERNAL_NET any
```

Enable **rule-based detection**:

```
include $RULE_PATH/local.rules
```

3. Running Snort in IDS Mode

```
sudo snort -A console -i eth0 -c /etc/snort/snort.conf -l /var/log/snort/
```

This command starts Snort in **detection mode**, analyzing traffic and logging suspicious activity.

4. Running Snort in IPS Mode

To actively **block** threats, integrate Snort with iptables:

```
sudo iptables -A INPUT -p tcp --dport 80 -j QUEUE
```

Start Snort in **IPS mode**:

```
sudo snort -Q --daq afpacket -i eth0 -c /etc/snort/snort.conf
```

CHAPTER 4: INSTALLING AND CONFIGURING SURICATA IDS/IPS

1. Installing Suricata

- **On Debian/Ubuntu:**
- `sudo apt install suricata -y`
- **On CentOS/RHEL:**
- `sudo yum install suricata -y`

2. Configuring Suricata

Edit the Suricata configuration file:

```
sudo nano /etc/suricata/suricata.yaml
```

Set **home network variables**:

```
HOME_NET: "[192.168.1.0/24]"
```

Enable **logging of detected threats**:

default-log-dir: /var/log/suricata/

3. Running Suricata in IDS Mode

sudo suricata -c /etc/suricata/suricata.yaml -i etho

4. Running Suricata in IPS Mode

To enable Suricata as an IPS:

sudo suricata -c /etc/suricata/suricata.yaml --af-packet -D

5. Viewing Suricata Alerts

Check logs for detected threats:

sudo tail -f /var/log/suricata/fast.log

Chapter 5: Host-Based IDS & IPS (HIDS/HIPS) with OSSEC

1. What is OSSEC?

OSSEC is an open-source **host-based intrusion detection system (HIDS)** that monitors:

- **File integrity changes**.
- **Unauthorized system modifications**.
- **Login attempts and root access logs**.

2. Installing OSSEC

- **On Debian/Ubuntu:**
- sudo apt install ossec-hids-server -y

- **On CentOS/RHEL:**
- sudo yum install ossec-hids-server -y

3. Configuring OSSEC

Edit OSSEC's main configuration file:

```
sudo nano /var/ossec/etc/ossec.conf
```

Enable **log monitoring**:

```
<localfile>
  <log_format>syslog</log_format>
  <location>/var/log/auth.log</location>
</localfile>
```

4. Starting OSSEC

```
sudo systemctl start ossec-hids
sudo systemctl enable ossec-hids
```

5. Checking OSSEC Alerts

To monitor security events:

```
sudo cat /var/ossec/logs/alerts/alerts.log
```

Chapter 6: Case Study – Implementing an IDS/IPS for Enterprise Security

Scenario:

A financial institution needs **real-time threat detection and prevention** to protect customer data from cyber threats.

Solution:

1. **Deploy Snort IDS** for **network-based intrusion detection**.
2. **Use Suricata as an IPS** to actively block malicious traffic.
3. **Install OSSEC HIDS** to **monitor system logs and file integrity**.
4. **Configure centralized logging** to track and analyze security incidents.

Outcome:

- **Improved threat visibility** with real-time alerts.
- **Active protection** against network intrusions.
- **Compliance with security standards (e.g., PCI-DSS, ISO 27001).**

CHAPTER 7: EXERCISE

1. **Install and configure Snort as an IDS.**
2. **Enable Suricata IPS mode and block suspicious traffic.**
3. **Set up OSSEC to monitor /var/log/auth.log for unauthorized login attempts.**
4. **Analyze firewall logs to detect suspicious network traffic.**
5. **Create custom IDS rules to detect specific attack patterns.**

CONCLUSION

IDS & IPS systems are **critical for detecting and preventing cyber threats**. By deploying **Snort, Suricata, and OSSEC**, administrators can **monitor network traffic, block attacks, and secure Linux systems**. 🚀

BEST PRACTICES FOR LINUX HARDENING

CHAPTER 1: INTRODUCTION TO LINUX HARDENING

What is Linux Hardening?

Linux hardening refers to **securing a Linux system** by implementing security best practices to protect it from cyber threats, unauthorized access, and system vulnerabilities. Hardening involves **configuring system settings, reducing attack surfaces, and enforcing access controls**.

Why is Linux Hardening Important?

- **Prevents unauthorized access** and security breaches.
- **Protects sensitive data** and system integrity.
- **Reduces the attack surface** by removing unnecessary services.
- **Ensures compliance** with security standards (e.g., PCI-DSS, ISO 27001, NIST).

This guide covers **essential Linux security hardening practices**, including **user management, firewall configurations, kernel security, file system protection, and auditing tools**.

CHAPTER 2: USER AND ACCESS CONTROL HARDENING

1. Enforce Strong Password Policies

To enforce strong password rules, configure:

sudo nano /etc/security/pwquality.conf

Set:

```
minlen = 12      # Minimum length
dcredit = -1     # At least one digit
ucredit = -1     # At least one uppercase letter
lcredit = -1     # At least one lowercase letter
ocredit = -1     # At least one special character
```

2. Disable Root Login and Use sudo

Prevent direct root access via SSH:

```
sudo nano /etc/ssh/sshd_config
```

Change:

```
PermitRootLogin no
```

Restart SSH:

```
sudo systemctl restart ssh
```

3. Implement Two-Factor Authentication (2FA)

Install **Google Authenticator** for 2FA:

```
sudo apt install libpam-google-authenticator -y
google-authenticator
```

Enable it in PAM:

```
sudo nano /etc/pam.d/sshd
```

Add:

```
auth required pam_google_authenticator.so
```

4. Restrict Login Access Using /etc/security/access.conf

To allow only specific users to log in:

```
sudo nano /etc/security/access.conf
```

Add:

```
-:ALL EXCEPT alice bob:ALL
```

This allows only **alice** and **bob** to log in.

Chapter 3: Service and Process Hardening

1. Disable Unused Services

To list active services:

```
sudo systemctl list-units --type=service --state=running
```

To disable an unused service (e.g., FTP):

```
sudo systemctl disable vsftpd --now
```

2. Restrict Background Processes

Check running processes:

```
ps aux --sort=-%mem
```

Kill an unwanted process:

```
sudo kill -9 <PID>
```

3. Limit SSH Access to Specific Users

Edit SSH config:

```
sudo nano /etc/ssh/sshd_config
```

Add:

```
AllowUsers alice bob
```

Restart SSH:

```
sudo systemctl restart ssh
```

Chapter 4: Network and Firewall Hardening

1. Configure the Firewall (UFW or iptables)

Using UFW (Ubuntu/Debian)

```
sudo ufw default deny incoming

sudo ufw default allow outgoing

sudo ufw allow ssh

sudo ufw enable
```

Using iptables

```
sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT

sudo iptables -A INPUT -p tcp --dport 80 -j ACCEPT

sudo iptables -P INPUT DROP
```

To make rules persistent:

```
sudo apt install iptables-persistent

sudo netfilter-persistent save
```

2. Disable ICMP (Ping) Requests

```
sudo nano /etc/sysctl.conf
```

Add:

```
net.ipv4.icmp_echo_ignore_all = 1
```

Apply changes:

```
sudo sysctl -p
```

3. Enable SSH Rate Limiting

Limit failed SSH attempts:

```
sudo ufw limit ssh
```

or with iptables:

```
sudo iptables -A INPUT -p tcp --dport 22 -m recent --set --name SSH
```

```
sudo iptables -A INPUT -p tcp --dport 22 -m recent --update --seconds 60 --hitcount 5 -j DROP --name SSH
```

Chapter 5: File System and Kernel Hardening

1. Set File Permissions Securely

To prevent unauthorized access:

```
sudo chmod -R 750 /etc
```

```
sudo chmod 700 ~/.ssh
```

To make a file immutable:

```
sudo chattr +i /etc/passwd
```

2. Encrypt Disk Partitions with LUKS

To encrypt a partition /dev/sdb1:

```
sudo cryptsetup luksFormat /dev/sdb1
```

```
sudo cryptsetup luksOpen /dev/sdb1 encrypted_partition
```

```
sudo mkfs.ext4 /dev/mapper/encrypted_partition
```

3. Secure the Bootloader (GRUB)

To prevent unauthorized boot modifications:

```
sudo grub-mkpasswd-pbkdf2
```

Copy the generated hash and add it to:

```
sudo nano /etc/grub.d/40_custom
```

Add:

```
set superusers="admin"

password_pbkdf2 admin <HASH>
```

Update GRUB:

```
sudo update-grub
```

Chapter 6: Intrusion Detection and Logging

1. Enable Logging with auditd

To track unauthorized activities:

```
sudo apt install auditd -y
```

```
sudo auditctl -w /etc/passwd -p wa -k password_changes
```

Check audit logs:

```
sudo ausearch -k password_changes
```

2. Set Up Fail2Ban to Block Brute-Force Attacks

Install Fail2Ban:

sudo apt install fail2ban -y

Configure SSH protection:

sudo nano /etc/fail2ban/jail.local

Add:

[sshd]

enabled = true

bantime = 600

maxretry = 3

Restart Fail2Ban:

sudo systemctl restart fail2ban

3. Enable Automatic Security Updates

For Debian/Ubuntu:

sudo apt install unattended-upgrades -y

sudo dpkg-reconfigure unattended-upgrades

Case Study – Hardening a Linux Server for an Enterprise

Scenario:

A company needs to **secure its Linux web server** against potential attacks while maintaining **performance and availability**.

Solution:

1. **Implement user authentication hardening** with **2FA and SSH restrictions**.
2. **Apply kernel security** by disabling unnecessary modules.
3. **Set up a firewall** with UFW to allow only necessary traffic.
4. **Monitor logs with auditd** and **Fail2Ban** to detect intrusions.
5. **Encrypt sensitive files** and **disable root login** via SSH.

Outcome:

- **Unauthorized users are blocked** from accessing critical files.
- **Reduced risk of brute-force attacks** with account lockout.
- **Encrypted data ensures confidentiality** and compliance with regulations.

CHAPTER 7: EXERCISE

1. **Configure SSH to allow only specific users to log in.**
2. **Set up UFW to block all ports except SSH, HTTP, and HTTPS.**
3. **Encrypt a partition using LUKS and mount it securely.**
4. **Install and configure Fail2Ban to block brute-force attacks.**
5. **Use auditd to monitor changes in /etc/passwd.**

CONCLUSION

Linux hardening is essential for **securing systems against cyber threats**. By **enforcing access controls, configuring firewalls, encrypting data, and monitoring system activity**, administrators can **significantly reduce security risks** and protect their Linux environments. 🚀

Encrypting File Systems (LUKS, GPG)

Chapter 1: Introduction to File System Encryption

What is File System Encryption?

File system encryption is a **security technique** used to protect data from unauthorized access by converting it into an unreadable format. Only authorized users with the **correct decryption key** can access the data.

Why is File Encryption Important?

- **Protects sensitive data** from unauthorized access.
- **Ensures compliance** with security standards (e.g., GDPR, HIPAA).
- **Prevents data theft** if the system is compromised.
- **Enhances privacy** by restricting access to encrypted files.

Common File Encryption Methods

Encryption Method	Description
LUKS (Linux Unified Key Setup)	Encrypts entire disk partitions.
GPG (GNU Privacy Guard)	Encrypts individual files using public-key cryptography.

This guide covers **LUKS for disk encryption** and **GPG for file encryption**, including installation, configuration, and best practices.

Chapter 2: Understanding LUKS (Linux Unified Key Setup)

1. What is LUKS?

LUKS (**Linux Unified Key Setup**) is a **disk encryption standard** that allows users to encrypt entire disk partitions. It uses **passphrase-based access control** and **key management** to ensure data security.

2. Features of LUKS

- **Supports multiple keys for one partition.**
- **Uses strong encryption algorithms (AES, Serpent, Twofish).**
- **Provides key revocation and management options.**
- **Compatible with most Linux distributions.**

Chapter 3: Setting Up LUKS for Full Disk Encryption

1. Installing LUKS

Most Linux distributions **come with LUKS pre-installed**. To verify:

```
sudo cryptsetup --version
```

If not installed:

```
sudo apt install cryptsetup -y   # Debian/Ubuntu
```

```
sudo yum install cryptsetup -y   # CentOS/RHEL
```

2. Encrypting a New Partition Using LUKS

Step 1: Identify the Disk Partition

List available partitions:

```
lsblk
```

Example output:

```
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda      8:0    0  500G  0 disk
├─sda1   8:1    0  100G  0 part /
├─sda2   8:2    0  100G  0 part /home
└─sda3   8:3    0  300G  0 part
```

Choose the partition to encrypt (e.g., /dev/sda3).

Step 2: Initialize LUKS on the Partition

```
sudo cryptsetup luksFormat /dev/sda3
```

- You will be prompted to enter a **passphrase** (which will be required for decryption).

Step 3: Open the Encrypted Partition

```
sudo cryptsetup luksOpen /dev/sda3 my_encrypted_volume
```

Step 4: Format the Encrypted Partition

```
sudo mkfs.ext4 /dev/mapper/my_encrypted_volume
```

Step 5: Mount the Encrypted Partition

Create a mount point:

```
sudo mkdir /mnt/secure
```

Mount the partition:

```
sudo mount /dev/mapper/my_encrypted_volume /mnt/secure
```

Step 6: Automatically Mount LUKS at Boot

To auto-mount after reboot, add an entry in /etc/crypttab:

```
echo "my_encrypted_volume /dev/sda3 none luks" | sudo tee -a /etc/crypttab
```

Then add the mapped device to /etc/fstab:

```
echo "/dev/mapper/my_encrypted_volume /mnt/secure ext4 defaults 0 2" | sudo tee -a /etc/fstab
```

Chapter 4: Managing LUKS Keys and Passwords

1. Adding a New Key to LUKS

LUKS allows multiple passphrases for one encrypted partition. To add a new key:

```
sudo cryptsetup luksAddKey /dev/sda3
```

You will be asked to enter an **existing key first**, then add the new one.

2. Removing a Key from LUKS

To delete a specific key:

```
sudo cryptsetup luksRemoveKey /dev/sda3
```

3. Changing the LUKS Passphrase

```
sudo cryptsetup luksChangeKey /dev/sda3
```

4. Checking LUKS Encryption Status

```
sudo cryptsetup luksDump /dev/sda3
```

Chapter 5: Encrypting Individual Files Using GPG (GNU Privacy Guard)

1. What is GPG?

GPG (**GNU Privacy Guard**) is a tool that uses **public-key cryptography** to encrypt files, ensuring only authorized users can decrypt them.

2. Installing GPG

- **On Debian/Ubuntu:**
- sudo apt install gnupg -y
- **On CentOS/RHEL:**
- sudo yum install gnupg -y

Chapter 6: Encrypting Files with GPG

1. Encrypting a File with a Passphrase

To encrypt a file (secret.txt):

gpg -c secret.txt

This generates secret.txt.gpg.

2. Decrypting a File with a Passphrase

gpg secret.txt.gpg

You will be prompted to enter the **decryption passphrase**.

3. Encrypting a File Using Public-Key Cryptography

To encrypt a file for a specific user:

gpg --output secret.gpg --encrypt --recipient alice@example.com secret.txt

4. Decrypting a File Using Private Key

```
gpg --output secret.txt --decrypt secret.gpg
```

Chapter 7: Case Study – Secure Data Storage for a Financial Institution

Scenario:

A financial institution needs to **secure sensitive customer data** stored on Linux servers while ensuring **only authorized employees can access it**.

Solution:

1. **Encrypt entire partitions with LUKS** to protect customer records.
2. **Use GPG to encrypt financial reports** before sending them via email.
3. **Restrict access to encrypted volumes** using **role-based authentication**.
4. **Monitor encryption logs** to track access attempts.

Outcome:

- **Customer data is encrypted and secure.**
- **Unauthorized users cannot access financial records.**
- **Regulatory compliance is ensured.**

Chapter 8: Exercise

1. **Encrypt a partition using LUKS and mount it securely.**
2. **Create a new LUKS key and remove an old key.**
3. **Encrypt a file using GPG and decrypt it.**
4. **Enable auto-mounting of a LUKS partition at boot.**
5. **Monitor LUKS access logs to detect unauthorized access attempts.**

CONCLUSION

File system encryption with **LUKS and GPG** provides **robust security** for protecting sensitive data. By **encrypting entire partitions and securing individual files**

ASSIGNMENT SOLUTION: IMPLEMENT FIREWALL RULES USING IPTABLES – STEP-BY-STEP GUIDE

Objective

This assignment provides a **step-by-step guide** on configuring firewall rules using **iptables** to **secure a Linux system** by controlling incoming and outgoing network traffic.

STEP 1: UNDERSTANDING IPTABLES FIREWALL

1. What is iptables?

iptables is a **packet filtering firewall** built into the Linux kernel. It allows administrators to **define rules** for managing network traffic by using different chains:

Chain	Function
INPUT	Controls incoming traffic to the system.
FORWARD	Manages traffic passing through the system.
OUTPUT	Controls outgoing traffic from the system.

2. Checking if iptables is Installed

To verify that iptables is installed, run:

```
sudo iptables --version
```

If iptables is not installed:

- **On Debian/Ubuntu:**

- sudo apt install iptables -y
- **On CentOS/RHEL:**
- sudo yum install iptables -y

STEP 2: VIEWING EXISTING IPTABLES RULES

Before making changes, **check current rules**:

sudo iptables -L -v --line-numbers

This displays:

- **Rule numbers**
- **Packets and bytes matched**
- **Current firewall rules**

STEP 3: SETTING DEFAULT FIREWALL POLICIES

By default, we **deny all incoming traffic** and **allow outgoing traffic**:

sudo iptables -P INPUT DROP

sudo iptables -P FORWARD DROP

sudo iptables -P OUTPUT ACCEPT

Explanation:

- **DROP all incoming traffic** unless explicitly allowed.
- **DROP forwarded traffic** (if the system is acting as a router).
- **ACCEPT outgoing traffic** (so the system can send data).

Step 4: Allowing Essential Services

1. Allow SSH Access (Port 22)

To allow remote SSH access:

```
sudo iptables -A INPUT -p tcp --dport 22 -m state --state NEW,ESTABLISHED -j ACCEPT
```

```
sudo iptables -A OUTPUT -p tcp --sport 22 -m state --state ESTABLISHED -j ACCEPT
```

Explanation:

- **Incoming SSH requests (port 22) are allowed.**
- **Outgoing SSH replies are allowed.**

2. Allow Web Traffic (HTTP and HTTPS)

To allow incoming web traffic:

```
sudo iptables -A INPUT -p tcp --dport 80 -j ACCEPT  # Allow HTTP
```

```
sudo iptables -A INPUT -p tcp --dport 443 -j ACCEPT  # Allow HTTPS
```

3. Allow DNS Requests (Port 53)

```
sudo iptables -A INPUT -p udp --dport 53 -j ACCEPT
```

```
sudo iptables -A OUTPUT -p udp --sport 53 -j ACCEPT
```

4. Allow Ping Requests (ICMP)

To allow the system to be pinged:

```
sudo iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT
```

```
sudo iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT
```

To **block ping requests**, replace ACCEPT with DROP.

Step 5: Blocking Unwanted Traffic

1. Block a Specific IP Address

To block an IP (e.g., 192.168.1.100):

```
sudo iptables -A INPUT -s 192.168.1.100 -j DROP
```

2. Block Access to a Specific Port (Example: Port 23 – Telnet)

```
sudo iptables -A INPUT -p tcp --dport 23 -j DROP
```

3. Limit SSH Login Attempts to Prevent Brute-Force Attacks

```
sudo iptables -A INPUT -p tcp --dport 22 -m recent --set --name SSH
```

```
sudo iptables -A INPUT -p tcp --dport 22 -m recent --update --seconds 60 --hitcount 5 -j DROP --name SSH
```

Explanation:

- Allows only **5 SSH login attempts per minute** from an IP.

Step 6: Allowing Established and Related Connections

To ensure ongoing connections are not blocked:

```
sudo iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

This allows:

- Responses from connections the system **already initiated**.

Step 7: Saving and Making iptables Rules Persistent

By default, iptables rules **do not persist after reboot**. To save them:

- **On Debian/Ubuntu:**
- sudo apt install iptables-persistent -y
- sudo netfilter-persistent save
- **On CentOS/RHEL:**
- sudo service iptables save
- sudo systemctl enable iptables

To manually reload rules after reboot:

```
sudo iptables-restore < /etc/iptables/rules.v4
```

Step 8: Deleting and Flushing iptables Rules

1. Delete a Specific Rule (Using Line Numbers)

To list rules with numbers:

```
sudo iptables -L --line-numbers
```

To delete a rule by its number:

```
sudo iptables -D INPUT 3
```

2. Flush All iptables Rules (Reset Firewall)

```
sudo iptables -F
```

```
sudo iptables -X
```

3. Reset Default Policies

sudo iptables -P INPUT ACCEPT

sudo iptables -P FORWARD ACCEPT

sudo iptables -P OUTPUT ACCEPT

Case Study – Secure a Web Server with iptables

Scenario:

A company hosts a **web server** and needs a firewall to:

1. **Allow HTTP, HTTPS, and SSH traffic.**
2. **Block all other incoming traffic.**
3. **Prevent brute-force SSH attacks.**

Solution:

1. **Set default policies:**
2. sudo iptables -P INPUT DROP
3. sudo iptables -P FORWARD DROP
4. sudo iptables -P OUTPUT ACCEPT
5. **Allow SSH only from a specific IP (203.0.113.5):**
6. sudo iptables -A INPUT -p tcp --dport 22 -s 203.0.113.5 -j ACCEPT
7. **Allow HTTP and HTTPS traffic:**
8. sudo iptables -A INPUT -p tcp --dport 80 -j ACCEPT
9. sudo iptables -A INPUT -p tcp --dport 443 -j ACCEPT

10. **Enable connection tracking for active connections:**

11. sudo iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

12. **Save the firewall rules:**

13. sudo netfilter-persistent save

Outcome:

- **Only HTTP, HTTPS, and SSH (from a trusted IP) are allowed.**
- **Brute-force SSH attacks are blocked.**
- **Unauthorized traffic is dropped, enhancing server security.**

Step 9: Exercise

1. **Set default iptables policies to block all incoming traffic and allow outgoing traffic.**
2. **Allow SSH access only from 192.168.1.10.**
3. **Block incoming connections to port 21 (FTP).**
4. **Limit SSH login attempts to 3 attempts per minute.**
5. **Save and reload iptables rules after a reboot.**

Conclusion

By following this guide, you have successfully:

✅ **Implemented iptables firewall rules** to secure a Linux system.

✅ **Allowed essential services while blocking unauthorized traffic.**

✅ **Protected SSH from brute-force attacks.**

✅ **Ensured firewall rules persist after reboot.**

Assignment Solution: Set Up and Configure SELinux for Enhanced Security – Step-by-Step Guide

Objective

This assignment provides a **step-by-step guide** to setting up and configuring **SELinux (Security-Enhanced Linux)** to enhance system security, enforce mandatory access control (MAC), and protect against unauthorized access.

Step 1: Understanding SELinux

1. What is SELinux?

SELinux (**Security-Enhanced Linux**) is a **kernel-level security module** that enforces **mandatory access control (MAC)** to restrict processes and users from accessing unauthorized resources.

2. SELinux Modes

SELinux operates in three modes:

Mode	**Description**
Enforcing	Enforces all SELinux policies (default mode in RHEL-based systems).
Permissive	SELinux logs policy violations but does not enforce them.
Disabled	SELinux is turned off completely.

To check the current SELinux mode:

getenforce

To view SELinux status:

sestatus

Step 2: Installing and Enabling SELinux

1. Checking if SELinux is Installed

To check if SELinux is installed:

rpm -q selinux-policy

If not installed:

- **On Debian/Ubuntu:**
- sudo apt install selinux-utils selinux-basics selinux-policy-default -y
- **On CentOS/RHEL:**
- sudo yum install policycoreutils selinux-policy selinux-policy-targeted -y

2. Enabling SELinux

To enable SELinux, edit the configuration file:

sudo nano /etc/selinux/config

Change:

SELINUX=enforcing

Save and exit, then **reboot the system**:

sudo reboot

After reboot, verify SELinux is **enforcing**:

```
getenforce
```

Step 3: Configuring SELinux Policies

1. Viewing SELinux Contexts

To check the SELinux security context of a file:

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

Example output:

```
-rw-r--r--. root root unconfined_u:object_r:httpd_sys_content_t:so index.html
```

2. Changing SELinux Contexts

To change the SELinux type of a file:

```
sudo chcon -t httpd_sys_content_t /var/www/html/index.html
```

To recursively apply the change to a directory:

```
sudo chcon -R -t httpd_sys_content_t /var/www/html/
```

3. Restoring SELinux Default Contexts

If file contexts are modified incorrectly, restore them:

```
sudo restorecon -Rv /var/www/html/
```

Step 4: Managing SELinux Booleans

SELinux uses **Boolean values** to **toggle permissions** for specific applications.

1. Viewing Available SELinux Booleans

```
sudo getsebool -a
```

2. Modifying a SELinux Boolean

For example, to **allow Apache to access home directories**:

```
sudo setsebool -P httpd_enable_homedirs on
```

3. Checking a Specific SELinux Boolean

```
sudo getsebool httpd_enable_homedirs
```

Step 5: Managing SELinux Rules for Services

1. Allowing Apache Web Server to Use Custom Directories

If a website is stored in /srv/web/, change its SELinux context:

```
sudo semanage fcontext -a -t httpd_sys_content_t "/srv/web(/.*)?"
sudo restorecon -Rv /srv/web/
```

2. Allowing an Application to Use a Specific Port

By default, SELinux restricts services from using **non-standard ports**. To allow Apache on **port 8080**:

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

To verify the change:

```
sudo semanage port -l | grep http
```

Step 6: Troubleshooting SELinux Issues

1. Checking SELinux Logs for Errors

If an application is **blocked by SELinux**, logs can be found in:

```
sudo cat /var/log/audit/audit.log | grep denied
```

2. Using audit2why to Understand SELinux Errors

To analyze why SELinux denied an action:

```
sudo cat /var/log/audit/audit.log | grep denied | audit2why
```

3. Creating a Custom SELinux Policy to Allow an Action

If an application is **blocked by SELinux**, you can create an exception:

```
sudo cat /var/log/audit/audit.log | grep denied | audit2allow -M my_custom_policy
```

```
sudo semodule -i my_custom_policy.pp
```

This creates a **policy module** to allow the blocked action.

Step 7: Temporarily Disabling SELinux for Testing

If an application is **not working due to SELinux**, switch SELinux to **permissive mode** temporarily:

```
sudo setenforce 0
```

To re-enable SELinux enforcement:

```
sudo setenforce 1
```

To permanently set SELinux to **permissive mode**, edit:

```
sudo nano /etc/selinux/config
```

Change:

SELINUX=permissive

Restart the system:

sudo reboot

Case Study – Securing a Web Server with SELinux

Scenario:

A company is hosting a website on an Apache web server but wants to:

1. **Ensure the website files are protected using SELinux.**
2. **Allow Apache to use port 8080.**
3. **Enable logging to track unauthorized access attempts.**

Solution Using SELinux:

1. **Ensure SELinux is in enforcing mode:**
2. sudo setenforce 1
3. **Change the security context of web files:**
4. sudo semanage fcontext -a -t httpd_sys_content_t "/srv/web(/.*)?"
5. sudo restorecon -Rv /srv/web/
6. **Allow Apache to use port 8080:**
7. sudo semanage port -a -t http_port_t -p tcp 8080
8. **Monitor logs for security violations:**
9. sudo tail -f /var/log/audit/audit.log

Outcome:

- **Website files are protected by SELinux policies.**
- **Apache can serve content on a non-standard port securely.**
- **Unauthorized access attempts are logged for auditing.**

STEP 8: EXERCISE

1. **Set SELinux to enforcing mode and verify the status.**
2. **Change the SELinux context of /var/www/html/index.html to httpd_sys_content_t.**
3. **Use semanage to allow a web server to use port 8081.**
4. **Check the audit logs for blocked actions and use audit2why to analyze them.**
5. **Create a custom SELinux policy to allow a new application to run in /opt/myapp/.**

CONCLUSION

By following this guide, you have successfully:

✅ **Installed and enabled SELinux for enhanced security.**
✅ **Configured SELinux policies to control file access.**
✅ **Managed SELinux Booleans to enable required functionalities.**
✅ **Troubleshot SELinux issues using audit logs and audit2allow.**