Independent Skill Development Mission

Independent Skill Development Mission NXT
EMPOWERING FUTURES, SHAPING TOMORROW

ISDM (INDEPENDENT SKILL DEVELOPMENT MISSION

NETWORK CONFIGURATION IN UNIX/LINUX

CHAPTER 1: INTRODUCTION TO NETWORK CONFIGURATION IN UNIX/LINUX

What is Network Configuration?

Network configuration in UNIX/Linux involves setting up and managing network interfaces, IP addresses, routing, and network services. Proper network configuration ensures seamless communication between systems, allowing users and applications to **access the internet, share files, and communicate within a network**.

Network configuration is essential for:

- **Connecting systems to the internet or a local network**
- **Managing static and dynamic IP addresses**
- **Configuring firewalls and security settings**
- **Setting up network services such as DNS, DHCP, and VPNs**

Administrators configure networks using tools such as ifconfig, ip, nmcli, and netplan. Understanding network settings helps troubleshoot connectivity issues, optimize performance, and enhance security.

Example: Checking the Current Network Configuration

Use the following command to display network interface details:

```
ip addr show
```

Output:

```
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default
  inet 192.168.1.100/24 brd 192.168.1.255 scope global enp0s3
```

This shows an **active network interface (enp0s3)** with an assigned **IP address (192.168.1.100)**.

Exercise

1. Use ifconfig or ip addr to list network interfaces on your system.
2. Identify the **default gateway** using ip route.

Case Study: Resolving Connectivity Issues in a Corporate Network

A company experiences frequent network disruptions. By **analyzing IP configurations and routing tables**, administrators identify **conflicting IP addresses**, which were causing packet drops. Fixing the issue restores smooth communication.

Chapter 2: Configuring Network Interfaces

Understanding Network Interfaces

A network interface is a virtual or physical connection that allows a UNIX/Linux system to communicate over a network. Common types include:

- **Ethernet (eth0, enpXsX)** – Wired network interfaces.
- **Wi-Fi (wlan0, wlpXsX)** – Wireless interfaces.
- **Loopback (lo)** – A virtual interface for local communication.

Configuring Static and Dynamic IP Addresses

Administrators can assign an **IP address dynamically (DHCP)** or **statically (manual configuration)**.

1. Assigning a Static IP Address (Using ifconfig)

```
sudo ifconfig eth0 192.168.1.150 netmask 255.255.255.0 up
```

This assigns **192.168.1.150** as the static IP for the eth0 interface.

2. Assigning a Static IP Address (Using ip Command)

```
sudo ip addr add 192.168.1.150/24 dev eth0
```

Verify the changes:

```
ip addr show eth0
```

3. Configuring Dynamic IP Address (DHCP)

To obtain an IP address automatically:

```
sudo dhclient eth0
```

Example: Setting Up a Persistent Static IP (Ubuntu/Debian)

Edit the network configuration file /etc/network/interfaces:

```
auto eth0

iface eth0 inet static

    address 192.168.1.150
```

```
  netmask 255.255.255.0

  gateway 192.168.1.1

  dns-nameservers 8.8.8.8
```

Restart networking service:

```
sudo systemctl restart networking
```

Exercise

1. Assign a **temporary** static IP using ifconfig and verify it.
2. Configure a **permanent** static IP by editing /etc/network/interfaces or netplan.

Case Study: Migrating a Server to a New Network

A company relocates its database server to a new **subnet (192.168.2.x)**. By **updating the static IP settings and configuring DNS properly**, the transition is completed **without downtime**.

CHAPTER 3: CONFIGURING NETWORK ROUTES AND GATEWAYS

Understanding Routing in UNIX/Linux

Routing determines how packets travel from one network to another. The **default gateway** directs traffic to external networks, such as the internet.

Checking the Current Routing Table

Use the ip route command to display the routing table:

```
ip route show
```

Example output:

```
default via 192.168.1.1 dev eth0
```

```
192.168.1.0/24 dev eth0 proto kernel scope link src 192.168.1.150
```

This shows:

- The **default gateway is 192.168.1.1** (router).
- Traffic for **192.168.1.x** stays within the local network (eth0).

Adding a Static Route

To add a new route to **192.168.2.0/24** via a gateway:

```
sudo ip route add 192.168.2.0/24 via 192.168.1.254 dev eth0
```

To delete the route:

```
sudo ip route del 192.168.2.0/24
```

Example: Redirecting Traffic Through a Different Gateway

If you have two network interfaces and want to send traffic for **10.0.0.0/8** via eth1:

```
sudo ip route add 10.0.0.0/8 via 192.168.1.100 dev eth1
```

Exercise

1. Identify the **default gateway** and routing table using ip route.
2. Add a static route for a **new subnet** and verify using ip route show.

Case Study: Optimizing Routing for a Multi-Homed Server

A server has **two internet connections (ISP1 and ISP2)**. By configuring **custom routes**, administrators ensure **critical traffic** (database syncs) uses **ISP1**, while regular traffic routes through **ISP2**, improving reliability.

Chapter 4: Managing DNS and Hostname Resolution

Understanding DNS in UNIX/Linux

The **Domain Name System (DNS)** translates human-readable domain names into IP addresses.

Checking Current DNS Settings

```
cat /etc/resolv.conf
```

Example output:

```
nameserver 8.8.8.8

nameserver 8.8.4.4
```

This system uses **Google DNS servers.**

Changing DNS Servers

To set a custom DNS server (1.1.1.1 - Cloudflare DNS):

```
echo "nameserver 1.1.1.1" | sudo tee /etc/resolv.conf
```

Configuring a Permanent DNS Server (Ubuntu/Debian)

Edit /etc/network/interfaces and add:

```
dns-nameservers 8.8.8.8 8.8.4.4
```

Restart networking:

```
sudo systemctl restart networking
```

Example: Testing DNS Resolution

To check if DNS works correctly:

nslookup google.com

or

dig google.com

Exercise

1. Change your system's **DNS settings** to use 1.1.1.1 and verify using nslookup.
2. Identify the DNS server your system is currently using.

Case Study: Resolving Slow Website Loading Times

A company experiences **slow website loading** due to **ISP DNS issues**. By switching to **Google DNS (8.8.8.8) or Cloudflare DNS (1.1.1.1)**, browsing speed significantly improves.

CONCLUSION

This guide covered:

✅ Configuring **static and dynamic IP addresses**.
✅ Setting up **network routes and default gateways**.
✅ Managing **DNS and hostname resolution**.
✅ Using **network tools (ifconfig, ip route, nslookup)** for troubleshooting.

UNDERSTANDING NETWORK PROTOCOLS (TCP/IP, UDP) IN UNIX/LINUX

CHAPTER 1: INTRODUCTION TO NETWORK PROTOCOLS

What are Network Protocols?

Network protocols define the **rules and standards** for data communication between computers over a network. They ensure **data is sent, received, and interpreted correctly** across different devices and platforms.

Two fundamental network protocols in UNIX/Linux are:

1. **TCP/IP (Transmission Control Protocol/Internet Protocol)** – A reliable, connection-oriented protocol.
2. **UDP (User Datagram Protocol)** – A connectionless, fast, and lightweight protocol.

Understanding these protocols is essential for:

- **Network configuration and troubleshooting**
- **Optimizing network performance**
- **Securing network traffic**
- **Developing network-based applications**

Example: Checking Active Network Connections

```
netstat -tunlp
```

This command lists **all active TCP and UDP connections** along with their listening ports and associated processes.

Exercise

1. Run netstat -tulpn to identify active **TCP and UDP** connections on your system.
2. Use ping to check network connectivity between two devices.

Case Study: Diagnosing Network Latency Issues in an Enterprise

A company experiences **slow database performance** due to **high latency** in TCP connections. By analyzing logs and using network monitoring tools, administrators identify **packet loss and optimize TCP configurations**, improving system response time.

CHAPTER 2: UNDERSTANDING THE TCP/IP PROTOCOL SUITE

What is TCP/IP?

TCP/IP (Transmission Control Protocol/Internet Protocol) is the **foundational protocol** for internet communication. It ensures **reliable data transmission** across networks by breaking data into packets, sending them to the destination, and reassembling them.

Key Features of TCP/IP

- **Reliable, connection-oriented communication**
- **Ensures data integrity with error checking and retransmission**
- **Flow control to prevent packet loss or congestion**
- **Uses IP addressing to identify source and destination devices**

TCP/IP Four-Layer Model

Layer	Function	Protocols

Application	Interfaces with software	HTTP, FTP, SSH, DNS
Transport	Ensures end-to-end communication	TCP, UDP
Internet	Routing and addressing	IP, ICMP, ARP
Network Access	Physical data transmission	Ethernet, Wi-Fi

Example: Checking Network Interfaces and IP Configuration

ip addr show

This command lists all **IP addresses and interfaces** in a UNIX/Linux system.

Exercise

1. Identify the **IP address and subnet mask** of your system using ip a.
2. Use traceroute google.com to track the network path to Google's servers.

Case Study: Migrating a Web Application to a New Network

A web hosting company **moves its application servers** to a different subnet. By **updating TCP/IP configurations and routing tables**, they ensure a **smooth migration** without downtime.

Chapter 3: Transmission Control Protocol (TCP)

What is TCP?

TCP (Transmission Control Protocol) is a **connection-oriented protocol** that ensures **reliable, ordered, and error-checked data transfer** between devices.

How TCP Works

1. **Establishes a Connection** (Three-way handshake: SYN, SYN-ACK, ACK)
2. **Transfers Data with Error Checking and Flow Control**
3. **Ensures Reliable Delivery with Retransmission**
4. **Closes the Connection Gracefully**

TCP Three-Way Handshake Process

1. **SYN** → Client requests connection
2. **SYN-ACK** → Server acknowledges request
3. **ACK** → Client confirms connection

```
tcpdump -i etho 'tcp'
```

This captures TCP packets in real time for troubleshooting.

Example: Checking Open TCP Ports

```
sudo netstat -tulnp | grep LISTEN
```

This lists **active TCP listening ports** and services.

Exercise

1. Capture live TCP traffic using tcpdump.
2. Check which TCP ports are open using netstat.

Case Study: Optimizing TCP for High-Performance Web Applications

An online retailer experiences **slow web page loading times** due to **inefficient TCP settings**. By **tuning TCP window size and enabling keepalive**, they **reduce latency** and enhance user experience.

Chapter 4: User Datagram Protocol (UDP)

What is UDP?

UDP (User Datagram Protocol) is a **connectionless, fast, and lightweight protocol** used in applications where speed is more important than reliability.

Key Features of UDP

- **No connection setup** – Data is sent without establishing a connection.
- **Lower latency than TCP** – Ideal for real-time applications.
- **No error checking or retransmission** – Packet loss can occur.

Common Use Cases for UDP

Application	**Protocol**
Live Streaming	RTP (Real-Time Protocol)
VoIP (Voice over IP)	SIP, RTP
Online Gaming	Game servers use UDP for low-latency communication
DNS Queries	UDP is used for fast domain resolution

Example: Checking Open UDP Ports

```
sudo netstat -tulnp | grep udp
```

This lists **UDP services running on the system**.

Testing UDP Connectivity with nc (Netcat)

Start a UDP listener on port 9999:

```
nc -lu 9999
```

Send a message from another machine:

```
echo "Hello UDP" | nc -u <destination_IP> 9999
```

Exercise

1. Identify active UDP services on your system.
2. Test UDP connectivity using nc (Netcat).

Case Study: Using UDP for Video Streaming Optimization

A media company improves **live video streaming performance** by switching from **TCP to UDP**, reducing **buffering and transmission delays** for viewers.

Chapter 5: TCP vs. UDP - When to Use Each

Feature	**TCP**	**UDP**
Connection Type	Connection-oriented	Connectionless
Reliability	Ensures data delivery	No guaranteed delivery
Error Checking	Retransmits lost packets	No retransmission

Speed	Slower, due to handshaking	Faster, no handshaking
Use Case	Web browsing, file transfers	Streaming, VoIP, gaming

Choosing Between TCP and UDP

- **Use TCP when reliability is critical** (e.g., file downloads, emails).
- **Use UDP when speed is essential** (e.g., video conferencing, DNS).

Example: Testing Both Protocols

Test TCP connectivity to port 22 (SSH):

```
nc -zv <server> 22
```

Test UDP connectivity to port 53 (DNS):

```
nc -zu <server> 53
```

Exercise

1. Compare TCP and UDP speeds using iperf.
2. Identify which protocol is used for DNS resolution.

Case Study: Choosing TCP vs. UDP for Financial Transactions

A bank needs **secure and reliable transactions** for **online banking**. By using **TCP**, they **prevent data loss** and ensure **financial security**.

Conclusion

This guide covered:

✅ **Understanding TCP/IP and UDP protocols.**

- ✅ **Configuring TCP connections and analyzing traffic.**
- ✅ **Using UDP for high-speed communication.**
- ✅ **Choosing between TCP and UDP based on application needs.**

Configuring SSH, FTP, SCP, and Telnet in UNIX/Linux

Chapter 1: Introduction to Remote Access and File Transfer Protocols

What are SSH, FTP, SCP, and Telnet?

Remote access and file transfer are essential for managing UNIX/Linux systems. The primary protocols for remote management and file exchange include:

- **SSH (Secure Shell)** – Secure remote access to UNIX/Linux servers.
- **FTP (File Transfer Protocol)** – Transfers files between systems.
- **SCP (Secure Copy Protocol)** – Securely copies files over SSH.
- **Telnet** – Remote access but lacks encryption (mostly deprecated for security reasons).

These protocols enable administrators to **manage systems, transfer files, and execute commands remotely**, making them crucial for network administration and system maintenance.

Example: Checking if SSH is Running on a Linux System

```
sudo systemctl status ssh
```

This verifies if the SSH service is active and running.

Exercise

1. Check if **SSH and FTP services** are installed on your system.

2. Use netstat -tulpn | grep ssh to confirm the SSH port is open.

Case Study: Enabling Secure Remote Access for a Distributed Team

A software company needs **secure remote access** for its global team. By configuring **SSH with key-based authentication**, they ensure encrypted connections, preventing unauthorized access.

Chapter 2: Configuring SSH (Secure Shell)

Installing and Enabling SSH

Most UNIX/Linux distributions include SSH by default. If not installed, use:

```
sudo apt install openssh-server   # Debian-based
```

```
sudo yum install openssh-server   # RHEL-based
```

Enable and start the SSH service:

```
sudo systemctl enable ssh
```

```
sudo systemctl start ssh
```

Connecting to a Remote Server via SSH

To connect to a remote system:

```
ssh username@remote_IP
```

Example:

```
ssh user@192.168.1.100
```

Setting Up Key-Based Authentication

1. **Generate SSH Keys (On Client Machine)**
2. ssh-keygen -t rsa -b 4096
3. **Copy the Public Key to the Server**
4. ssh-copy-id user@192.168.1.100
5. **Disable Password Authentication (On Server)**
   Edit /etc/ssh/sshd_config and set:
6. PasswordAuthentication no

Restart SSH:

sudo systemctl restart ssh

Example: Restricting SSH Access to Specific Users

Edit /etc/ssh/sshd_config and add:

AllowUsers admin user1

Restart SSH:

sudo systemctl restart ssh

Exercise

1. Set up SSH key-based authentication for secure login.
2. Restrict SSH access to a specific user on your system.

Case Study: Securing Remote Access for Cloud Servers

A company manages **AWS cloud servers**. To prevent **brute force attacks**, they disable password authentication and allow SSH access **only from whitelisted IPs**.

CHAPTER 3: CONFIGURING FTP (FILE TRANSFER PROTOCOL)

Installing and Enabling an FTP Server

Install the **vsftpd** (Very Secure FTP Daemon) package:

```
sudo apt install vsftpd   # Debian-based
```

```
sudo yum install vsftpd   # RHEL-based
```

Start and enable the FTP service:

```
sudo systemctl enable vsftpd
```

```
sudo systemctl start vsftpd
```

Configuring FTP Server Settings

Edit the configuration file:

```
sudo nano /etc/vsftpd.conf
```

Ensure the following settings:

```
anonymous_enable=NO

local_enable=YES

write_enable=YES

chroot_local_user=YES
```

Restart the FTP server:

```
sudo systemctl restart vsftpd
```

Creating FTP Users

1. **Create a New FTP User**
2. sudo useradd -m ftpuser

3. sudo passwd ftpuser
4. **Set File Permissions for FTP Directory**
5. sudo chown ftpuser:ftpuser /home/ftpuser

Connecting to FTP Server

- **From a UNIX/Linux system:**
- ftp 192.168.1.100
- **From a web browser:**
- ftp://192.168.1.100

Example: Checking Active FTP Connections

netstat -tulpn | grep vsftpd

Exercise

1. Create a new FTP user and restrict access to their home directory.
2. Use an FTP client like FileZilla to transfer files.

Case Study: Setting Up an FTP Server for Internal File Sharing

A company needs a **secure file-sharing system** for internal teams. By setting up an **FTP server with user authentication**, they enable efficient file transfers without external access.

Chapter 4: Configuring SCP (Secure Copy Protocol)

What is SCP?

SCP is a **secure alternative to FTP** that allows **file transfers over SSH**. It provides encryption, making it more secure than traditional FTP.

Using SCP to Transfer Files

1. **Copy a File to a Remote Server**
2. scp filename user@remote_IP:/destination/path

Example:

```
scp report.pdf user@192.168.1.100:/home/user/
```

3. **Copy a File from a Remote Server**
4. scp user@remote_IP:/path/to/file local_directory

Example:

```
scp user@192.168.1.100:/home/user/report.pdf .
```

5. **Copy an Entire Directory**
6. scp -r local_directory user@remote_IP:/remote/directory

Example: Automating Secure File Transfers with SCP

Schedule a daily SCP backup using cron:

```
0 2 * * * scp /backup/*.tar.gz user@backup_server:/mnt/storage
```

This transfers backup files **daily at 2 AM** to a remote backup server.

Exercise

1. Transfer a file securely using SCP.
2. Automate a scheduled SCP backup using cron.

Case Study: Automating Secure Data Backups for a Remote Server

A research lab **automates daily data backups** using **SCP and cron jobs**, ensuring secure storage on an external server.

Chapter 5: Configuring Telnet (Deprecated)

What is Telnet?

Telnet is a **remote login protocol** similar to SSH, but it **does not encrypt traffic**, making it insecure. It is mostly **deprecated** but sometimes used for **legacy systems**.

Installing and Enabling Telnet

```
sudo apt install telnetd   # Debian-based
```

```
sudo yum install telnet-server   # RHEL-based
```

Start the Telnet service:

```
sudo systemctl enable telnet
```

```
sudo systemctl start telnet
```

Connecting to a Remote Server via Telnet

```
telnet remote_IP
```

Example: Restricting Telnet Access to Specific Users

Edit /etc/hosts.allow and add:

```
telnetd: 192.168.1.0/24
```

Restart Telnet service:

```
sudo systemctl restart telnet
```

Exercise

1. Connect to a remote Telnet server (if enabled).
2. Identify active Telnet connections using netstat -tulpn.

Case Study: Replacing Telnet with SSH for Security

A university migrates from **Telnet to SSH** to prevent **password sniffing and unauthorized access**, significantly improving security.

Conclusion

This guide covered:

✅ **Configuring SSH, FTP, SCP, and Telnet for remote access and file transfers.**
✅ **Implementing secure authentication with SSH keys.**
✅ **Using SCP for encrypted file transfers.**
✅ **Setting up FTP servers for internal file sharing.**

IMPLEMENTING FIREWALLS AND SECURITY POLICIES IN UNIX/LINUX

CHAPTER 1: INTRODUCTION TO FIREWALLS AND SECURITY POLICIES

What is a Firewall?

A **firewall** is a security mechanism that **monitors and controls incoming and outgoing network traffic** based on predefined security rules. It acts as a **barrier** between a trusted internal network and untrusted external networks (such as the internet). Firewalls help prevent **unauthorized access, malware attacks, and data breaches**.

Importance of Firewalls in UNIX/Linux

- **Protects systems from cyber threats** such as hacking attempts and malware.
- **Filters network traffic** based on rules defined by administrators.
- **Prevents unauthorized access** to sensitive data and services.
- **Monitors network activity** to detect unusual or suspicious connections.
- **Implements network segmentation** for enhanced security.

A **security policy** defines how a system **handles access control, authentication, and network filtering** to ensure maximum security. Firewalls enforce these policies to protect critical resources.

Example: Checking Firewall Status in Linux

To check if a firewall is active:

```
sudo ufw status   # For UFW (Debian/Ubuntu)

sudo firewall-cmd --state  # For Firewalld (RHEL/CentOS)
```

Exercise

1. Identify which firewall service is running on your system.
2. List all active firewall rules using sudo iptables -L or sudo ufw status.

Case Study: Protecting an Online Banking System

A bank experiences frequent hacking attempts on its web server. By implementing **a strict firewall policy that allows only HTTPS traffic (port 443) and blocks unauthorized IPs**, they reduce cyber threats by 80%.

Chapter 2: Understanding Different Types of Firewalls

1. Network-Based vs. Host-Based Firewalls

- **Network-based firewall** – Installed on network gateways to filter traffic for multiple systems.
- **Host-based firewall** – Installed on individual servers to protect specific applications.

2. Types of Firewalls

Firewall Type	Description	Example
Packet Filtering	Examines IP packets and allows/blocks based on rules.	iptables
Stateful Inspection	Monitors active connections and filters packets accordingly.	ufw, firewalld

Proxy Firewall	Intermediates communication between users and servers.	Squid Proxy
Next-Gen Firewall	Uses AI and deep packet inspection for enhanced security.	Palo Alto, Cisco ASA

Example: Checking Active Connections Using netstat

```
netstat -tunlp
```

This lists all **active TCP and UDP connections**, helping administrators detect unauthorized access.

Exercise

1. Use netstat -tunlp to identify open ports on your system.
2. Compare the difference between **stateful** and **stateless** firewalls.

Case Study: Securing a University Network with Firewalls

A university implements a **network-based firewall** to protect student databases. By setting rules that **block all non-educational traffic**, they prevent students from accessing unsafe websites, improving security.

CHAPTER 3: CONFIGURING IPTABLES FIREWALL IN LINUX

What is iptables?

iptables is a command-line firewall utility in Linux that allows administrators to **define packet filtering rules**. It operates based on a set of rules organized into **chains and tables**.

1. Checking Current Firewall Rules

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

This displays all active firewall rules.

2. Creating Basic Firewall Rules

To **block all incoming traffic** and allow only SSH:

```
sudo iptables -P INPUT DROP

sudo iptables -P OUTPUT ACCEPT

sudo iptables -P FORWARD DROP

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

This ensures that only **SSH (port 22)** is open while all other traffic is blocked.

3. Saving and Applying Rules

To make rules persistent:

```
sudo iptables-save > /etc/iptables.rules
```

Reload rules after reboot:

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

Example: Allowing HTTP and HTTPS Traffic

To allow web traffic (ports 80 and 443):

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

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

Exercise

1. Block all incoming traffic except SSH and HTTP using iptables.

2. Save firewall rules and make them persistent across reboots.

Case Study: Hardening Security for an E-Commerce Website

An online store implements **strict iptables rules**, allowing only **HTTP, HTTPS, and SSH traffic** while blocking all unnecessary ports. This reduces attack risks and enhances data security.

Chapter 4: Configuring UFW (Uncomplicated Firewall) for Simplicity

What is UFW?

UFW is a **simplified firewall management tool** used in Debian/Ubuntu. It provides an easier way to configure firewalls without complex iptables commands.

1. Enabling UFW

sudo ufw enable

2. Allowing Essential Services

- Allow SSH:
- sudo ufw allow ssh
- Allow Web Traffic (HTTP/HTTPS):
- sudo ufw allow 80/tcp
- sudo ufw allow 443/tcp

3. Blocking All Traffic Except Allowed Services

sudo ufw default deny incoming

sudo ufw default allow outgoing

4. Checking Firewall Rules

sudo ufw status numbered

Example: Allowing a Custom Port for an Application

If an application requires port 8080:

sudo ufw allow 8080/tcp

Exercise

1. Enable UFW and allow only **SSH and web traffic**.
2. Block all other incoming traffic and verify using ufw status.

Case Study: Implementing UFW for Small Business Security

A startup uses **UFW** to configure **basic firewall rules** quickly, allowing only SSH and HTTPS traffic. This prevents unauthorized access and secures customer transactions.

CHAPTER 5: BEST PRACTICES FOR SECURITY POLICIES

1. Principle of Least Privilege (PoLP)

- Grant users **only necessary permissions**.
- Restrict access to **critical system files and services**.

2. Regular Firewall Audits

- Check open ports using:
- sudo netstat -tulnp
- Review logs for suspicious activity:
- sudo cat /var/log/auth.log | grep "failed"

3. Implement Intrusion Detection (IDS/IPS)

- Install fail2ban to block failed SSH login attempts:
- sudo apt install fail2ban
- sudo systemctl enable fail2ban
- Configure alerts for **suspicious network behavior**.

Example: Blocking Repeated SSH Login Failures

To block an IP after **5 failed SSH attempts**:

sudo ufw limit ssh

Exercise

1. Set up fail2ban to monitor SSH login attempts.
2. Check logs for unauthorized access attempts.

Case Study: Preventing Brute Force Attacks on Cloud Servers

A cloud provider implements **firewall restrictions, fail2ban, and regular log audits**, preventing over **1000 unauthorized login attempts per month**.

Conclusion

This guide covered:

✅ Configuring **firewalls (iptables, ufw)** for security.
✅ Implementing **network filtering and traffic rules.**
✅ Using **security best practices** to protect UNIX/Linux servers.

Hardening UNIX Servers: Enhancing Security and Protection

Chapter 1: Introduction to UNIX Server Hardening

What is Server Hardening?

Server hardening is the process of **securing a UNIX/Linux server by minimizing vulnerabilities**, reducing attack surfaces, and enforcing strict security measures. A hardened server is **less vulnerable to cyberattacks, unauthorized access, and data breaches**.

Importance of Hardening UNIX Servers

- **Protects sensitive data from unauthorized access**
- **Prevents security breaches, malware, and cyberattacks**
- **Enhances system performance and reliability**
- **Reduces the attack surface by disabling unnecessary services**
- **Ensures compliance with security policies and industry standards**

UNIX/Linux server hardening involves **implementing security policies, restricting access, configuring firewalls, enforcing authentication mechanisms, and monitoring system activity**.

Example: Checking the Current Security Status of a UNIX Server

Use the following command to check open ports and services running on the system:

```
netstat -tulpn
```

Exercise

1. Run netstat -tulpn to identify **open ports** and active services on your system.
2. List **all active user accounts** on the server using cat /etc/passwd.

Case Study: Hardening a Web Hosting Server

A company hosting web applications hardens its servers by **disabling unused services, enforcing strong authentication, and configuring a firewall**. This reduces **attack attempts by 90%** and prevents unauthorized access.

CHAPTER 2: USER AND ACCESS CONTROL HARDENING

1. Creating and Managing Secure User Accounts

- Remove **unnecessary user accounts**:
- sudo userdel -r olduser
- Disable login for the **root** user:
- sudo passwd -l root

2. Enforcing Strong Password Policies

Modify the password policy in /etc/login.defs:

```
PASS_MAX_DAYS 90

PASS_MIN_DAYS 7

PASS_WARN_AGE 14
```

This ensures:

- Passwords expire every **90 days**.
- Users must wait **7 days before changing** a password.
- A **warning is issued 14 days** before expiration.

3. Implementing SSH Security Best Practices

Edit /etc/ssh/sshd_config and configure:

```
PermitRootLogin no

PasswordAuthentication no

AllowUsers admin user1
```

Restart SSH:

```
sudo systemctl restart ssh
```

Example: Listing All Users and Their Last Login

```
lastlog
```

Exercise

1. Disable **password authentication** and enable **key-based authentication** for SSH.
2. Restrict SSH access to specific users only.

Case Study: Securing a Financial Institution's Server

A bank **disables root login**, enforces **strong password policies**, and **restricts SSH access** to administrators only. This **eliminates unauthorized access attempts** and improves security compliance.

Chapter 3: Firewall and Network Security Hardening

1. Configuring a Firewall (iptables or UFW)

- Enable a firewall:
- sudo ufw enable
- Allow only **SSH, HTTP, and HTTPS**:
- sudo ufw allow 22
- sudo ufw allow 80
- sudo ufw allow 443

2. Disabling Unnecessary Network Services

List all active services:

sudo systemctl list-units --type=service

Disable unnecessary services:

sudo systemctl disable cups

3. Preventing Unauthorized Network Connections

- Check all open network ports:
- netstat -tulpn
- Close unnecessary ports using iptables:
- sudo iptables -A INPUT -p tcp --dport 23 -j DROP

Example: Blocking a Specific IP Address

sudo iptables -A INPUT -s 192.168.1.100 -j DROP

Exercise

1. Configure **a firewall rule** that blocks incoming connections on all ports except SSH.

2. Disable all **unused services** and verify using systemctl.

Case Study: Preventing Unauthorized Access to an E-commerce Server

An online store **blocks all ports except HTTP/HTTPS**, restricts SSH access, and **uses firewall rules to prevent brute-force attacks**, improving security and uptime.

CHAPTER 4: SECURING FILE SYSTEM AND PERMISSIONS

1. Enforcing Least Privilege File Permissions

- Restrict **home directory access**:
- sudo chmod 700 /home/*
- Remove **world-writable files**:
- sudo find / -type f -perm 777 -exec chmod 644 {} \;

2. Using chattr to Prevent Unauthorized File Changes

- Protect critical files from modification:
- sudo chattr +i /etc/passwd
- Verify immutable files:
- lsattr /etc/passwd

3. Enforcing Secure File Deletion

- Use shred to delete sensitive files securely:
- shred -u confidential.txt

Example: Finding Files with Weak Permissions

find / -type f -perm -o+w

This lists **all files that are writable by others**, helping in security auditing.

Exercise

1. Set file permissions to **protect user home directories** from unauthorized access.
2. Use chattr to **prevent modification** of critical system files.

Case Study: Protecting Critical Data in a Healthcare System

A hospital IT team **hardens file permissions**, ensuring that only authorized personnel **can access patient records**, complying with data privacy laws.

CHAPTER 5: MONITORING AND LOGGING FOR INTRUSION DETECTION

1. Enabling and Configuring System Logs

- View authentication logs:
- sudo cat /var/log/auth.log
- Monitor system errors:
- sudo cat /var/log/syslog | grep "error"

2. Installing Intrusion Detection Systems (fail2ban)

Install fail2ban to prevent brute-force attacks:

sudo apt install fail2ban

sudo systemctl enable fail2ban

Enable SSH protection:

```
sudo nano /etc/fail2ban/jail.local
```

```
[sshd]

enabled = true

bantime = 600

maxretry = 3
```

Restart fail2ban:

```
sudo systemctl restart fail2ban
```

3. Automating Security Audits

Run a basic security audit using lynis:

```
sudo apt install lynis

sudo lynis audit system
```

Example: Checking for Unauthorized Login Attempts

```
sudo grep "Failed password" /var/log/auth.log
```

Exercise

1. Install and configure **fail2ban** to prevent **SSH brute-force attacks**.
2. Set up a **scheduled log monitoring script** to detect failed logins.

Case Study: Detecting and Blocking Unauthorized SSH Attempts

A cloud hosting provider notices **thousands of failed SSH login attempts** daily. By **implementing fail2ban and monitoring logs,** they block attackers in real-time, **reducing security threats significantly**.

CONCLUSION

This guide covered:

- ✅ Hardening **user access and authentication**.
- ✅ Configuring **firewalls and network security**.
- ✅ Protecting the **file system and permissions**.
- ✅ Monitoring logs and **detecting intrusions**.

INTRUSION DETECTION AND PREVENTION TECHNIQUES IN UNIX/LINUX

CHAPTER 1: INTRODUCTION TO INTRUSION DETECTION AND PREVENTION

What is Intrusion Detection and Prevention?

Intrusion Detection and Prevention Systems (**IDPS**) are **security mechanisms** designed to **identify, monitor, and prevent unauthorized access, malicious activity, and cyber threats** in UNIX/Linux environments.

Why is Intrusion Detection Important?

- **Detects unauthorized access** before damage occurs.
- **Identifies suspicious behavior** such as brute-force attacks, malware injections, and privilege escalation attempts.
- **Prevents security breaches** by taking automated actions to block threats.
- **Ensures compliance with security standards** (e.g., PCI-DSS, HIPAA, ISO 27001).

IDPS tools can be **host-based (HIDS)** or **network-based (NIDS)**:

- **Host-Based Intrusion Detection Systems (HIDS)** → Monitors logs and system files for suspicious activity (**e.g., AIDE, OSSEC**).
- **Network-Based Intrusion Detection Systems (NIDS)** → Analyzes network traffic for malicious behavior (**e.g., Snort, Suricata**).

Example: Checking System Logs for Unauthorized Access Attempts

```
sudo grep "Failed password" /var/log/auth.log
```

This command **identifies failed SSH login attempts**, indicating potential brute-force attacks.

Exercise

1. List the last 10 failed login attempts using grep "Failed password" /var/log/auth.log | tail -10.
2. Check if your system has an IDS installed (sudo systemctl status ossec).

Case Study: Preventing Unauthorized Access to a Government Server

A government agency implements **host-based IDS (OSSEC) and network IDS (Snort)** to monitor security logs and detect **suspicious access attempts**. This prevents data breaches and enhances national cybersecurity.

CHAPTER 2: HOST-BASED INTRUSION DETECTION (HIDS)

What is HIDS?

A **Host-Based Intrusion Detection System (HIDS)** monitors **system logs, file integrity, and user activities** for anomalies. It detects attacks like:

- Unauthorized **root access** attempts.
- **Changes in critical system files** (/etc/passwd, /etc/shadow).
- **Brute-force login attempts** and failed authentication.

1. Installing and Configuring OSSEC (Open Source HIDS)

OSSEC is a popular **HIDS tool** that monitors system logs, file integrity, and user actions.

Installation (Debian/Ubuntu & RHEL/CentOS)

```
wget -qO - https://updates.atomicorp.com/installers/atomic | sudo bash
```

```
sudo yum install ossec-hids
```

Start the OSSEC service:

```
sudo systemctl start ossec
```

```
sudo systemctl enable ossec
```

2. Configuring OSSEC for Monitoring

Modify OSSEC's configuration file:

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

Add a rule to **monitor SSH login failures**:

```
<localfile>

  <log_format>syslog</log_format>

  <location>/var/log/auth.log</location>

</localfile>
```

Restart OSSEC:

```
sudo systemctl restart ossec
```

3. Checking OSSEC Alerts

To review detected threats:

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

Example: Detecting Unauthorized File Changes

Enable file integrity monitoring in /var/ossec/etc/ossec.conf:

```
<syscheck>

   <frequency>3600</frequency>

   <directories check_all="yes">/etc,/usr/bin,/var/log</directories>

</syscheck>
```

Restart OSSEC:

```
sudo systemctl restart ossec
```

Exercise

1. Install and configure OSSEC to monitor **SSH login failures**.
2. Check alerts.log for **recent security events**.

Case Study: Protecting a Financial Database with OSSEC

A bank deploys **OSSEC to monitor database access logs**. OSSEC **alerts administrators of unauthorized access attempts**, preventing data breaches.

Chapter 3: Network-Based Intrusion Detection (NIDS)

What is NIDS?

A **Network-Based Intrusion Detection System (NIDS)** monitors **network traffic** for suspicious patterns, such as:

- **DDoS attacks** (Distributed Denial-of-Service).

- **Port scanning** and unauthorized network access.
- **Malicious packet injections** or SQL injections.

1. Installing and Configuring Snort (Popular NIDS Tool)

Snort is a real-time **network intrusion detection tool**.

Installation (Debian/Ubuntu & RHEL/CentOS)

```
sudo apt install snort  # Debian/Ubuntu
```

```
sudo yum install snort  # RHEL/CentOS
```

2. Running Snort in IDS Mode

To detect suspicious traffic:

```
sudo snort -A console -q -c /etc/snort/snort.conf -i eth0
```

- -A console → Displays alerts in real time.
- -c /etc/snort/snort.conf → Uses Snort's rules.
- -i eth0 → Monitors the primary network interface.

3. Checking Detected Intrusions

To view Snort logs:

```
cat /var/log/snort/alert
```

Example: Detecting Port Scanning Attempts

Snort rule to block unauthorized scanning:

```
alert tcp any any -> any 22 (msg:"SSH Brute Force Attack"; flags:S;
sid:1001;)
```

Save and restart Snort:

```
sudo systemctl restart snort
```

Exercise

1. Install Snort and run it in **IDS mode** to monitor network traffic.
2. Identify **unauthorized network access attempts** in /var/log/snort/alert.

Case Study: Preventing a DDoS Attack on an E-commerce Website

An online retailer deploys **Snort to analyze incoming traffic**. When Snort detects an **unusually high number of requests from a single IP**, it alerts admins, who block the attacker using a firewall.

Chapter 4: Preventing Intrusions with Fail2Ban

What is Fail2Ban?

Fail2Ban is an **intrusion prevention tool** that blocks **IP addresses** that show signs of automated attacks.

1. Installing Fail2Ban

```
sudo apt install fail2ban  # Debian-based

sudo yum install fail2ban  # RHEL-based
```

Start Fail2Ban:

```
sudo systemctl start fail2ban

sudo systemctl enable fail2ban
```

2. Configuring Fail2Ban for SSH Protection

Edit the Fail2Ban configuration file:

```
sudo nano /etc/fail2ban/jail.local
```

Enable SSH protection:

```
[sshd]

enabled = true

bantime = 600

maxretry = 3
```

Restart Fail2Ban:

```
sudo systemctl restart fail2ban
```

3. Checking Banned IP Addresses

```
sudo fail2ban-client status sshd
```

Example: Unblocking a Banned IP

```
sudo fail2ban-client set sshd unbanip 192.168.1.200
```

Exercise

1. Configure Fail2Ban to block repeated **failed SSH login attempts**.
2. Verify **banned IP addresses** using fail2ban-client status sshd.

Case Study: Stopping Brute Force Attacks on Cloud Servers

A cloud provider implements **Fail2Ban** to block **automated SSH attacks**, reducing security threats by **95%**.

Conclusion

This guide covered:

✅ **Installing and configuring HIDS (OSSEC) for system**

monitoring.

✅ Deploying NIDS (Snort) for real-time network intrusion detection.

✅ Implementing Fail2Ban to block malicious login attempts.

✅ Preventing cyberattacks with automated security policies.

Assignment Solution: Configuring SSH for Secure Remote Access

Objective

This assignment provides a **step-by-step guide** to configure **SSH (Secure Shell) for secure remote access** in UNIX/Linux. By the end of this guide, you will:

✅ Install and enable SSH on a UNIX/Linux system.

✅ Configure SSH for **key-based authentication** and disable **password login**.

✅ Restrict SSH access to specific users and enforce security policies.

✅ Implement additional security measures such as **port change and time-based restrictions**.

Step 1: Install and Enable SSH

1. Check if SSH is Installed

Most UNIX/Linux distributions come with **OpenSSH** pre-installed. To verify, run:

```
which sshd
```

If SSH is not installed, install it based on your OS:

Debian/Ubuntu:

```
sudo apt update
```

```
sudo apt install openssh-server -y
```

RHEL/CentOS:

```
sudo yum install openssh-server -y
```

Arch Linux:

```
sudo pacman -S openssh
```

2. Enable and Start SSH Service

```
sudo systemctl enable ssh
```

```
sudo systemctl start ssh
```

Verify SSH is running:

```
sudo systemctl status ssh
```

Expected output:

```
Active: active (running)
```

Step 2: Configure SSH for Secure Remote Access

1. Open SSH Configuration File

Edit the SSH configuration file located at /etc/ssh/sshd_config:

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

2. Change SSH Default Port

By default, SSH runs on **port 22**, which is often targeted by attackers. Change it to a custom port, e.g., **2222**:

```
Port 2222
```

After changing the port, update the firewall rule:

```
sudo ufw allow 2222/tcp  # For UFW
sudo firewall-cmd --permanent --add-port=2222/tcp  # For Firewalld

sudo firewall-cmd --reload
```

3. Disable Root Login

For security reasons, **disable direct root login**:

```
PermitRootLogin no
```

This forces users to log in with non-root accounts.

4. Disable Password Authentication (Enforce Key-Based Login)

To disable password-based login and enforce **SSH key authentication**, set:

```
PasswordAuthentication no

PubkeyAuthentication yes
```

5. Allow Specific Users to Access SSH

Restrict SSH access to specific users, e.g., admin and user1:

```
AllowUsers admin user1
```

Save and exit (Ctrl + X, then Y, and Enter).

Restart SSH service for changes to take effect:

```
sudo systemctl restart ssh
```

Step 3: Configure Key-Based Authentication for Secure Login

1. Generate SSH Key Pair on Client Machine

Run the following command on the **client system** (the system you want to connect from):

```
ssh-keygen -t rsa -b 4096
```

- -t rsa → Specifies RSA encryption.
- -b 4096 → Uses a 4096-bit key for stronger security.

Press **Enter** to save the key in the default location (~/.ssh/id_rsa).

2. Copy the Public Key to the Server

Use ssh-copy-id to transfer the public key to the **remote server**:

```
ssh-copy-id -i ~/.ssh/id_rsa.pub user@remote_IP
```

Alternatively, manually copy the key:

```
cat ~/.ssh/id_rsa.pub | ssh user@remote_IP "mkdir -p ~/.ssh && cat >> ~/.ssh/authorized_keys"
```

Verify permissions on the **remote server**:

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

```
chmod 600 ~/.ssh/authorized_keys
```

3. Test SSH Key Authentication

Try logging in **without a password**:

```
ssh -p 2222 user@remote_IP
```

If login is successful, password authentication is correctly disabled.

Step 4: Additional SSH Security Enhancements

1. Enable SSH Logging for Security Auditing

Enable logging of SSH login attempts by modifying /etc/ssh/sshd_config:

```
LogLevel VERBOSE
```

Restart SSH:

```
sudo systemctl restart ssh
```

View SSH logs:

```
sudo cat /var/log/auth.log | grep "sshd"
```

2. Limit Login Attempts to Prevent Brute Force Attacks

Configure **Fail2Ban** to automatically block repeated failed SSH login attempts:

Install Fail2Ban

```
sudo apt install fail2ban -y  # Debian/Ubuntu

sudo yum install fail2ban -y  # RHEL/CentOS
```

Configure SSH Protection

Edit /etc/fail2ban/jail.local:

```
[sshd]

enabled = true

bantime = 600

maxretry = 3
```

Restart Fail2Ban:

```
sudo systemctl restart fail2ban
```

3. Restrict SSH Access by IP (Whitelist Trusted IPs)

Edit /etc/hosts.allow to allow only specific IPs:

```
sshd: 192.168.1.10 192.168.1.20
```

Block all others by adding to /etc/hosts.deny:

```
sshd: ALL
```

Restart SSH:

```
sudo systemctl restart ssh
```

Step 5: Verify and Test Secure SSH Configuration

1. Check SSH Listening Port

Ensure SSH is listening on the new port:

```
sudo netstat -tulpn | grep ssh
```

Expected output:

```
tcp   0   0 0.0.0.0:2222   0.0.0.0:*   LISTEN   1234/sshd
```

2. Verify SSH Key-Based Authentication

Ensure login works without a password:

```
ssh -p 2222 user@remote_IP
```

3. Check SSH Logs for Unauthorized Attempts

```
sudo cat /var/log/auth.log | grep "Failed password"
```

Identify and **block** attackers if needed.

CONCLUSION

✅ Installed and enabled SSH for secure remote access.

✅ Configured SSH **key-based authentication** and disabled **password login**.

✅ Hardened SSH security by **changing default ports, limiting users, and enabling logging**.

✅ Implemented **Fail2Ban to block brute-force attacks** and restricted SSH to trusted IPs.

Assignment Solution: Writing a Script to Automate Network Configuration in UNIX/Linux

Objective

This assignment provides a **step-by-step guide** to writing a **Bash script** to automate **network configuration** in UNIX/Linux. By the end of this guide, you will:

- ✅ Configure **a static or dynamic IP address** automatically.
- ✅ Automate **DNS settings and routing rules**.
- ✅ Apply **firewall rules** for security.
- ✅ Optimize the script for **different Linux distributions**.

Step 1: Understand Network Configuration Components

To configure a network automatically, the script must:

1. **Identify available network interfaces**.
2. **Set a static or dynamic (DHCP) IP address**.
3. **Configure DNS settings** for name resolution.
4. **Apply routing rules** to define network traffic paths.
5. **Enable firewall rules** for security.

Check your network interface:

```
ip link show
```

Check current IP configuration:

```
ip addr show
```

Step 2: Writing the Network Configuration Automation Script

1. Create the Script File

Create a new Bash script:

```
sudo nano network_config.sh
```

2. Add the Script Header

Start the script with a **shebang** and description:

```
#!/bin/bash

# Network Configuration Automation Script

# This script configures a network interface with a static or dynamic IP


# Ensure the script runs with root privileges

if [ "$(id -u)" -ne 0 ]; then

   echo "Please run as root!"

   exit 1

fi
```

3. Define Network Interface and IP Configuration

Prompt the user to **choose between static or dynamic (DHCP) IP**:

```
echo "Enter network interface (e.g., eth0, ens33, wlan0):"

read INTERFACE
```

```
echo "Choose IP configuration:"

echo "1) Static IP"

echo "2) Dynamic IP (DHCP)"

read CHOICE
```

4. Configure Static or Dynamic IP

Option 1: Static IP Configuration

If the user selects **Static IP**, prompt for details:

```
if [ "$CHOICE" -eq 1 ]; then

    echo "Enter Static IP Address (e.g., 192.168.1.100):"

    read IPADDR

    echo "Enter Subnet Mask (e.g., 255.255.255.0):"

    read NETMASK

    echo "Enter Default Gateway (e.g., 192.168.1.1):"

    read GATEWAY

    echo "Enter DNS Server (e.g., 8.8.8.8):"

    read DNS


    # Configure network settings

    echo "Configuring Static IP..."

    cat <<EOF > /etc/network/interfaces
```

```
auto $INTERFACE
iface $INTERFACE inet static
  address $IPADDR
  netmask $NETMASK
  gateway $GATEWAY
  dns-nameservers $DNS
EOF

  echo "Static IP configured successfully!"
fi
```

Option 2: Dynamic IP (DHCP) Configuration

If the user selects **DHCP**, configure the interface accordingly:

```
if [ "$CHOICE" -eq 2 ]; then
  echo "Configuring DHCP for $INTERFACE..."
  cat <<EOF > /etc/network/interfaces
auto $INTERFACE
iface $INTERFACE inet dhcp
EOF

  echo "DHCP configuration applied!"
fi
```

5. Restart Networking Service

Once the configuration is applied, restart the network service to activate changes:

```
echo "Restarting network service..."

sudo systemctl restart networking

echo "Network configuration updated successfully!"
```

6. Verify Network Configuration

After restarting, verify the applied settings:

```
echo "Checking network configuration..."

ip addr show $INTERFACE
```

Step 3: Make the Script Executable

Save and exit (Ctrl + X, then Y and Enter).
Make the script executable:

```
sudo chmod +x network_config.sh
```

Step 4: Run the Script and Test Configuration

Execute the script:

```
sudo ./network_config.sh
```

Test Cases

✅ **Case 1:** Choose **Static IP**, enter values, and verify ip addr show.
✅ **Case 2:** Choose **DHCP**, restart networking, and check ip addr show.

Step 5: Enhancing the Script with Firewall Rules

To improve security, the script can also configure **firewall rules**:

```
echo "Applying basic firewall rules..."

sudo ufw allow ssh

sudo ufw allow 80

sudo ufw allow 443

sudo ufw enable

echo "Firewall rules applied successfully!"
```

Final Script: Automated Network Configuration in UNIX/Linux

```
#!/bin/bash

# Network Configuration Automation Script

# This script sets up a static or dynamic IP and applies firewall rules


# Ensure script runs as root

if [ "$(id -u)" -ne 0 ]; then

  echo "Please run as root!"

  exit 1
```

```
fi


# Get network interface
echo "Enter network interface (e.g., eth0, ens33, wlan0):"
read INTERFACE


# Choose configuration type
echo "Choose IP configuration:"
echo "1) Static IP"
echo "2) Dynamic IP (DHCP)"
read CHOICE


if [ "$CHOICE" -eq 1 ]; then
   echo "Enter Static IP Address (e.g., 192.168.1.100):"
   read IPADDR
   echo "Enter Subnet Mask (e.g., 255.255.255.0):"
   read NETMASK
   echo "Enter Default Gateway (e.g., 192.168.1.1):"
   read GATEWAY
   echo "Enter DNS Server (e.g., 8.8.8.8):"
   read DNS
```

```
    echo "Configuring Static IP..."
    cat <<EOF > /etc/network/interfaces
auto $INTERFACE
iface $INTERFACE inet static
    address $IPADDR
    netmask $NETMASK
    gateway $GATEWAY
    dns-nameservers $DNS
EOF

    echo "Static IP configured successfully!"

elif [ "$CHOICE" -eq 2 ]; then
    echo "Configuring DHCP for $INTERFACE..."
    cat <<EOF > /etc/network/interfaces
auto $INTERFACE
iface $INTERFACE inet dhcp
EOF
    echo "DHCP configuration applied!"
else
```

```
    echo "Invalid selection. Exiting..."
    exit 1
fi

# Restart network service
echo "Restarting network service..."
sudo systemctl restart networking
echo "Network configuration updated successfully!"

# Apply basic firewall rules
echo "Applying firewall rules..."
sudo ufw allow ssh
sudo ufw allow 80
sudo ufw allow 443
sudo ufw enable
echo "Firewall configured successfully!"

# Verify configuration
echo "Checking network settings..."
ip addr show $INTERFACE
```

CONCLUSION

- ☑ **Automated network configuration for static/DHCP IP.**
- ☑ **Restarted network service and verified settings.**
- ☑ **Implemented firewall rules for SSH and web access.**
- ☑ **Created a reusable and flexible Bash script.**