STTP on Ethical Hacking & Cyber Forensics@IIITK
Day 4: Man-In-The-Middle Attacks & Suricata IDS/IPS

Dr. JAI GANESH & Dr. Renu Mary Daniel

June 12, 2025

Contents

1 Understanding Man-In-The-Middle (MITM)

Man-in-the-Middle (MITM) is a cyberattack where an attacker secretly intercepts and relays communications between two parties who believe they are communicating directly with each other. The attacker can then eavesdrop on, or even alter, the communication.

1.1 Types of MITM Attacks

MITM attacks encompass various techniques to intercept traffic. Here are 8 common types:

1. **Email Hijacking**: Gaining unauthorized control over an email account to send, receive, or modify emails.
2. **IP Spoofing**: Creating IP packets with a false source IP address to masquerade as another computer system.
3. **Session Hijacking**: Exploiting a valid computer session to gain unauthorized access to information or services in a computer system.
4. **Wi-Fi Eavesdropping**: Intercepting data packets over an unsecured Wi-Fi network.
5. **DNS Spoofing**: Corrupting DNS server data, causing the server to return an incorrect IP address for a domain name.
6. **Secure Sockets Layer (SSL) Hijacking**: Intercepting and decrypting HTTPS traffic, often by tricking a client into accepting a fake SSL certificate.
7. **ARP Cache Poisoning**: Manipulating ARP tables to redirect network traffic to the attacker.
8. **Stealing Browser Cookies**: Capturing session cookies to impersonate a legitimate user.

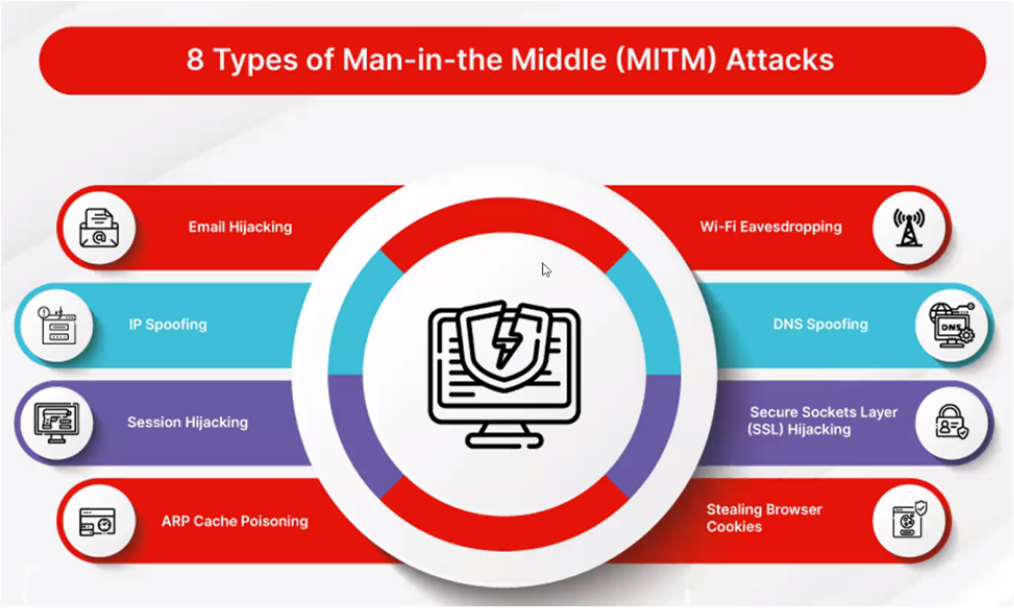

Figure 1: 8 Types of Man-in-the-Middle (MITM) Attacks

2 Understanding ARP (Address Resolution Protocol)

The Address Resolution Protocol (ARP) is a crucial protocol used to resolve IP addresses to MAC (Media Access Control) addresses on local networks. This mapping is essential for devices to communicate within the same local area network (LAN).

2.1 What is ARP?

ARP's primary function is to enable devices to deliver messages to the correct machine on a local network by resolving logical IP addresses to physical MAC addresses. It acts as a "matchmaker" between Layer 3 (Network Layer) IP addresses and Layer 2 (Data Link Layer) MAC addresses in the OSI/TCP-IP models.

2.2 ARP Analogy (Apartment Building)

Imagine a large apartment building as your network.

- You know your friend's **name** (which is like an **IP address**).
- But you don't know which **flat number** (which is like a **MAC address**) they stay in.
- So, you stand in the hallway and shout: "Hey! Who lives at IP address 192.168.1.5?"
- Your friend replies from their flat: "That's me! My MAC address is AA:BB:CC:DD:EE:FF."

This "hallway shouting system" is exactly what ARP does: it maps a logical IP address to a permanent MAC address.

2.3 ARP in Action

Let's illustrate ARP's operation with two devices, Host A and Host B, on the same network.

1. **Host A knows:** Host B's IP address (e.g., 192.168.1.10) but not its MAC address.
2. **Host A broadcasts an ARP Request:** "Who has 192.168.1.10? Tell me your MAC!" This request is sent to all devices on the local network.
3. **Host B Responds with an ARP Reply:** "That's me! My MAC is AA:BB:CC:DD:EE:FF." This reply is a unicast message sent directly to Host A.
4. **Host A saves it in its ARP table:** Host A stores this IP-to-MAC mapping in its ARP cache, so it doesn't need to ask again for a while.

ARP works exclusively within the same Local Area Network (LAN), operating at Layer 2.

2.4 ARP Packet Creation (Cisco Packet Tracer)

In a simulated network environment like Cisco Packet Tracer, we can observe how ARP packets are created and how address resolution occurs. When a device (Host A) needs to communicate with another device (Host B) on the same subnet but doesn't have Host B's MAC address, it initiates the process by sending an ARP Request.

Initially, the Ethernet frame encapsulating this ARP Request is sent with a destination MAC address set to all 'F's (FFFFFFFFFFFF), which signifies a broadcast address. This ensures that all devices on the local network receive the ARP Request.

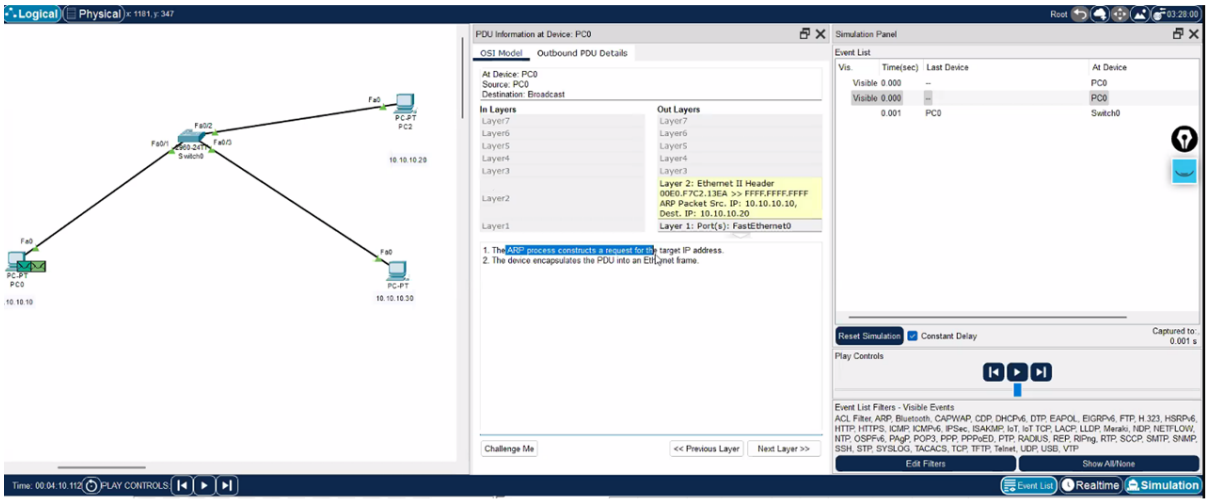

Figure 2: PDU Information showing destination MAC initially set to broadcast (all F's).

The PDU details confirm that the Ethernet II Header's destination address is broadcast, and within the ARP packet itself, it contains the source's IP and MAC addresses, the destination's IP address, while the target MAC address field is initially unknown (often zeroed out).

Upon receiving this broadcast ARP Request, the device whose IP address matches the target IP in the request (Host B) will perform two key actions:

- It updates its *own* ARP table with Host A's IP and MAC address (learned from the incoming ARP Request packet).
- It then sends an ARP Reply directly back to Host A (a unicast message) containing its own MAC address.

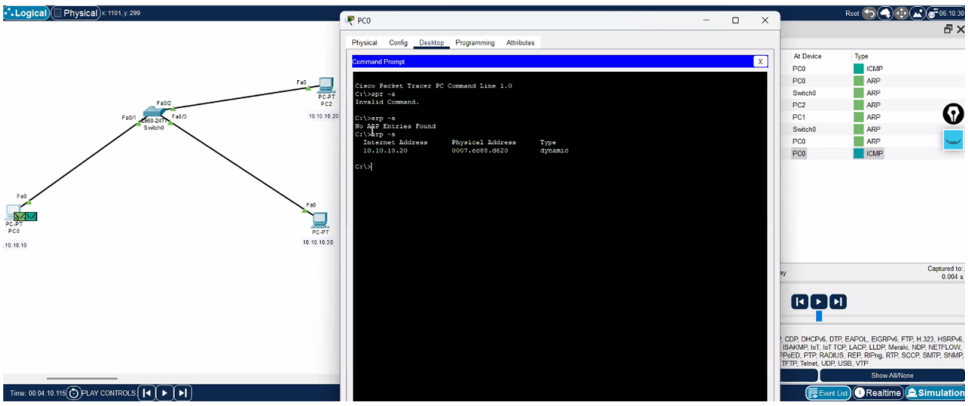

Figure 3: ICMP response.

ARP then broadcasts message to map the ip address to mac address

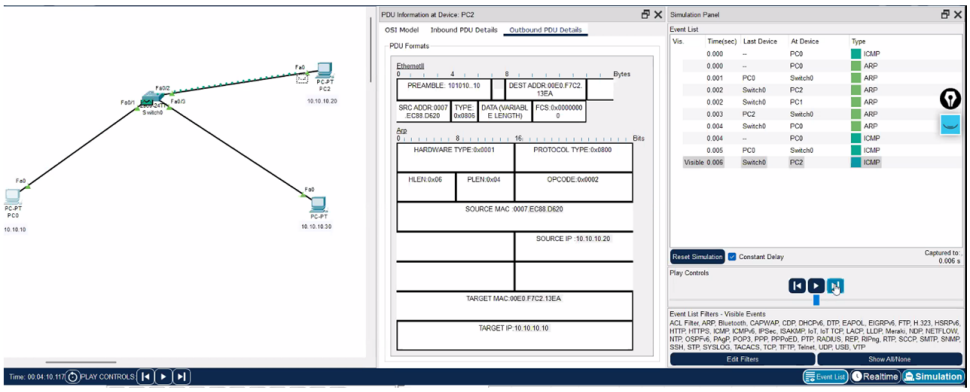

Figure 4: ARP Broadcast

When Host A receives Host B's ARP Reply, it updates *its own* ARP table with Host B's IP and corresponding MAC address. This successful mapping then allows Host A to send subsequent IP packets (like an ICMP ping request) directly to Host B's MAC address for communication within the local network.

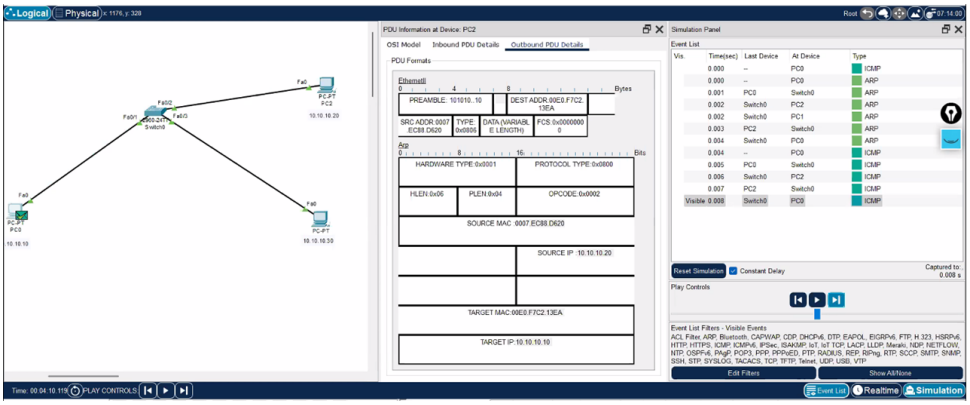

Figure 5: Packet simulation completed.

3 ARP Spoofing & MITM Attacks

ARP's design, while efficient, introduces a significant security vulnerability: it does not verify the authenticity of ARP replies.

3.1 Why ARP is Vulnerable?

- **No Authentication:** ARP doesn't verify if a reply is legitimate. Anyone on the network can claim to be a specific IP address without needing to prove it. This is like someone shouting "I'm your pizza guy" without providing any credentials.
- **Blind Trust:** When a device receives an ARP reply, even if it never asked for it, it updates its ARP table accordingly. This inherent trust makes it easy for an attacker to poison the ARP table with fake IP-to-MAC mappings.

3.2 The Man-in-the-Middle

The result of ARP poisoning is that packets meant for a legitimate destination (e.g., the router or another device) are silently diverted to the attacker first. The attacker then becomes the "Man-in-the-Middle", capable of:

- **Seeing:** Eavesdropping on all communications.
- **Modifying:** Altering the data being transmitted.
- **Redirecting:** Sending the traffic to unintended destinations.

Essentially, if ARP is a friendly guy handing out house numbers, ARP Spoofing is that shady neighbor who lies and says, 'Yeah, all deliveries come to my flat now.'

4 Lab Walkthrough: ARP Spoofing with Ettercap

This lab demonstrates ARP spoofing using Ettercap in a VirtualBox environment.

4.1 Lab Setup

We will use three virtual machines configured with a host-only adapter and promiscuous mode enabled for the attacker machine:

- **Server (Windows 10):** IP: `192.168.15.8`
- **Client (Windows 10):** IP: `192.168.15.6`
- **Attacker (Kali Linux):** IP: `192.168.15.7`

The objective is to listen to the communication between the client and server after the attacker initiates ARP poisoning.

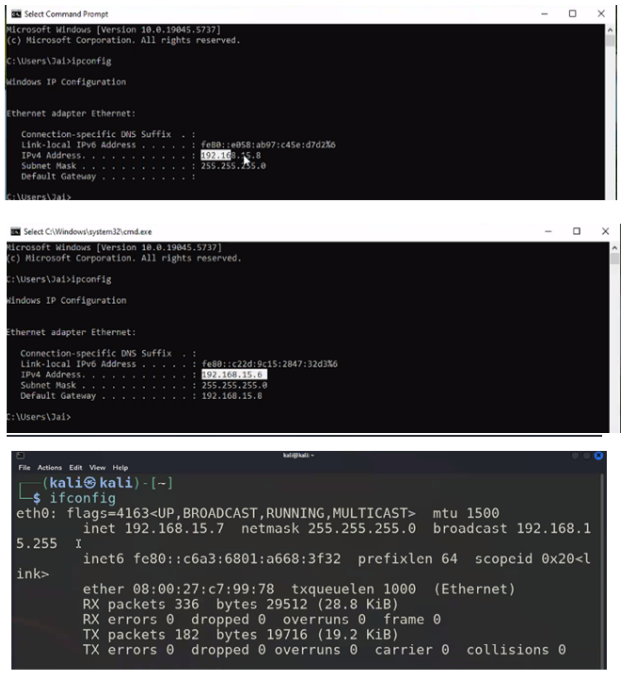

Figure 6: IP addresses of Server (192.168.15.8), Client (192.168.15.6), and Kali (192.168.15.7).

4.2 Pre-Spoofing Preparation

Before initiating the ARP spoofing attack, it's good practice to clear the ARP cache on the target machines and verify initial ARP entries.

4.2.1 Clearing ARP Cache

On Windows machines (Server and Client), open Command Prompt as Administrator and run:

```
arp -d *
```

Listing 1: Clearing ARP cache

4.2.2 Initial ARP Table Verification

After clearing the cache, ping other machines from both the server and client to populate their ARP tables. Then, check the ARP tables using `arp -a`.

```
ping 192.168.15.6
ping 192.168.15.7
arp -a
```

Listing 2: Pinging and verifying ARP table on Server

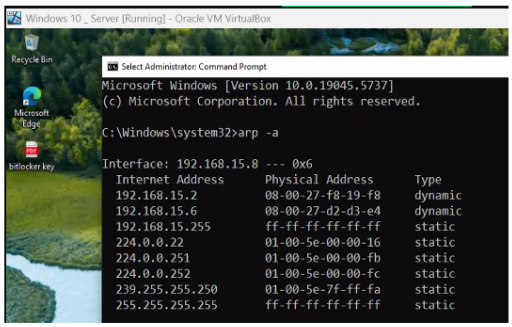

Figure 7: Server's initial ARP table entries after pings.

4.3 Ettercap Configuration

The attacker (Kali Linux) will use Ettercap to perform the ARP poisoning.

4.3.1 Enable IP Forwarding on Kali

This step is crucial for the attacker to correctly forward traffic between the victim machines, otherwise, communication will break.

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

Listing 3: Enabling IP forwarding on Kali

4.3.2 Start Ettercap and Scan for Hosts

Launch Ettercap, ensure the correct primary interface (e.g., 'eth0') is selected for sniffing, and then scan for hosts on the network. Promiscuous mode should also be enabled.

```
sudo ettercap -G
```

Listing 4: Starting Ettercap

Alternatively gui application could be directly opened and setup

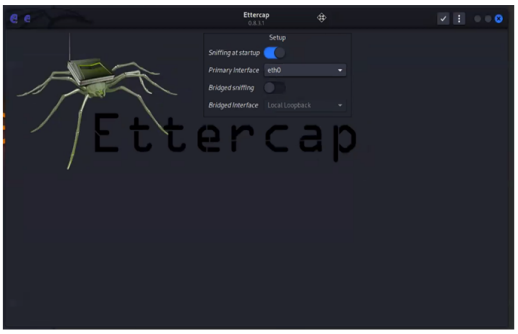

Figure 8: Ettercap sniffing setup and interface selection.

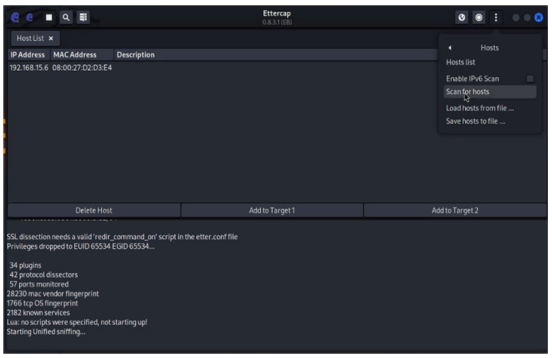

Figure 9: Ettercap GUI: Scanning for hosts.

4.3.3 Select Targets for ARP Poisoning

From the scanned host list, select the Server (`192.168.15.8`) and add it as Target 2. Select the Client (`192.168.15.6`) and add it as Target 1. This configures Ettercap to intercept traffic between these two specific hosts.

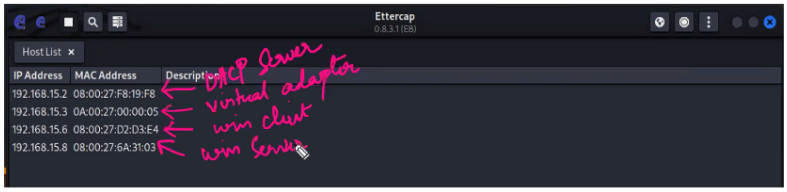

Figure 10: Ettercap hosts list with identified IPs and MACs.

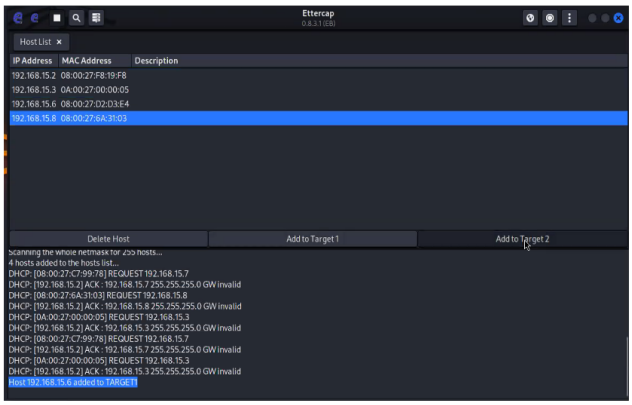

Figure 11: Adding Server (192.168.15.8) and Client (192.168.15.6) as targets.

4.3.4 Start ARP Poisoning Attack

Go to the "MITM" menu in Ettercap and select "ARP poisoning". Choose to "Sniff remote connections".

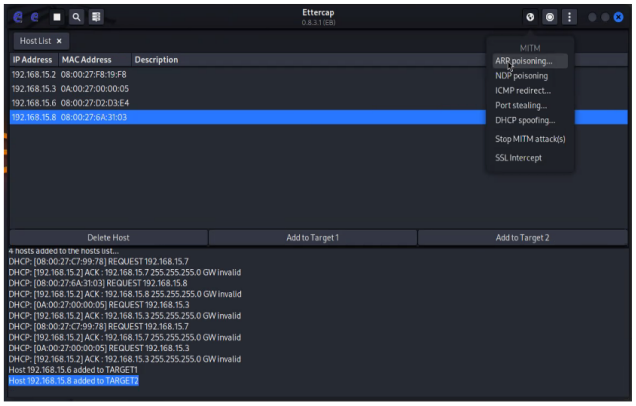

Figure 12: Starting ARP poisoning attack in Ettercap.

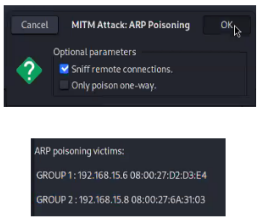

Figure 13: ARP poisoning victims confirmation.

4.4 Post-Spoofing Verification

After the ARP poisoning attack starts, check the ARP tables on the Server and Client again. You should observe that the MAC addresses corresponding to the other host (and potentially the gateway) now point to Kali's MAC address (08:00:27:C7:99:78 for Kali's eth0).

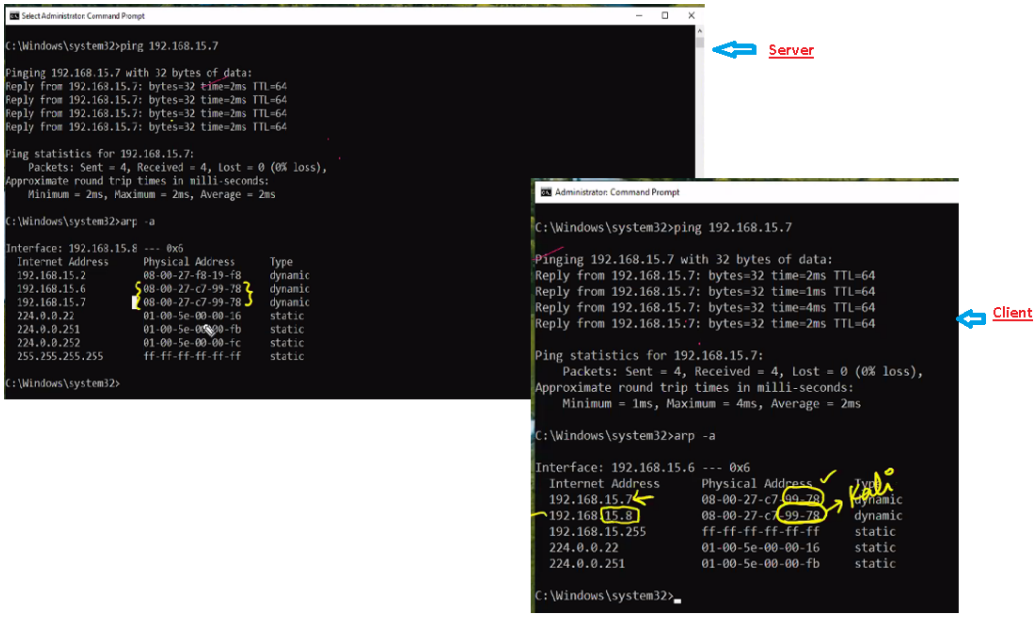

Figure 14: ARP tables on Server and Client after ARP poisoning. Note the altered MAC addresses.

Even with poisoned ARP tables, communication between the server and client should still work because Kali is forwarding the packets. However, all traffic is now passing through Kali, allowing the attacker to intercept it. Ping the client from the server and observe successful replies, indicating traffic is being routed through the attacker.

```
ping 192.168.15.6
arp -a
```

Listing 5: Pinging Client from Server after ARP Poisoning

5 DNS Spoofing with Ettercap

DNS Spoofing is another type of MITM attack where an attacker redirects domain name queries to a malicious server, often to phishing sites.

5.1 Enabling DNS Spoofing

In Ettercap, you can enable DNS spoofing through the "Plugins" menu. Select the "dns_spoof" plugin.

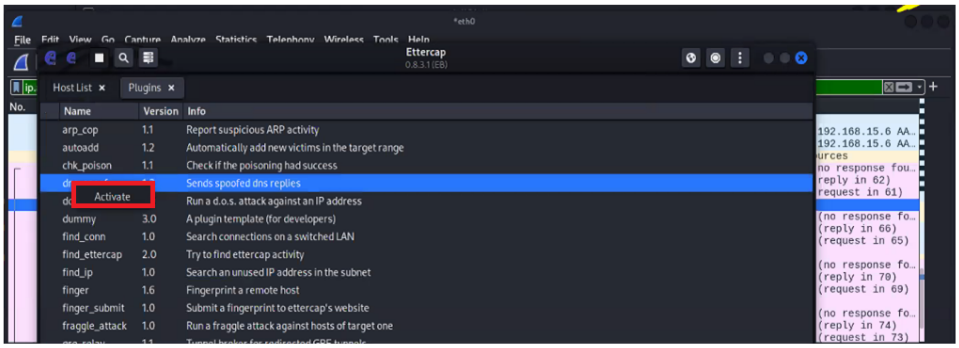

Figure 15: Enabling DNS Spoofing plugin in Ettercap.

5.2 Modifying `etter.dns` file

To configure DNS spoofing, you need to edit the `etter.dns` file, usually located at `/usr/share/ettercap/etter.dns`. In this file, you can specify domain names and the IP addresses they should resolve to. For example, to redirect ‘1bank.com‘ to the Kali machine’s IP (`192.168.15.7`), you would add the following line:

```
bank.com A 192.168.15.7
```

Listing 6: Editing etter.dns file

After modifying the file and enabling the plugin, any DNS queries for ‘bank.com‘ from the poisoned victims will be redirected to Kali’s IP.

6 Suricata: Intrusion Detection/Prevention System (IDS/IPS)

Suricata is a powerful, open-source network threat detection engine used for Intrusion Detection System (IDS), Intrusion Prevention System (IPS), and Network Security Monitoring (NSM).

6.1 Introduction to Suricata

- **Open-source network threat detection engine.**
- Developed and maintained by the **Open Information Security Foundation (OISF)**.
- **Functions as:**
  - **Intrusion Detection System (IDS):** Monitors traffic for suspicious activity.
  - **Intrusion Prevention System (IPS):** Actively blocks malicious traffic.
  - **Network Security Monitoring (NSM):** Gathers network metadata and analyzes it for security insights.
- **Analyzes network traffic**, detects potential threats, and can take actions to prevent malicious activity.

6.2 Suricata Requirements (VirtualBox Setup)

For the Suricata lab, we will use two Virtual Machines:

- **Kali VM (Attacker):** To generate malicious traffic.
- **Ubuntu VM (Suricata):** Where Suricata will be installed and run.

Important Network Configuration:

- Both VMs must be on the **same network** (e.g., using a Bridged Adapter).
- They must be able to **ping each other**.
- The Ubuntu VM (Suricata) needs **Promiscuous Mode: ON** in its network adapter settings to capture all network traffic.
- Ensure both VMs are connected to the internet for package installations.

6.3 Suricata Rule/Signature Structure

A Suricata rule consists of three main components:

- **Action:** What happens when the rule matches (e.g., ‘alert‘, ‘drop‘).
- **Header:** Defines the protocol, IP addresses, ports, and direction of the traffic.
- **Rule Options:** Specifies the specifics of the rule, including payload content, message, SID (Signature ID), and revision.

Syntax:

```
action protocol src_ip src_port -> dst_ip dst_port (rule-options)
```

Example:

```
alert tcp $HOME_NET any -> any 80 (msg: "Visit to malicious http site"; sid
    :10001; rev:1;)
```

6.4 Rule Direction

The arrow in the rule header indicates the direction of traffic the rule applies to:

- `->`: Matches traffic flowing from ‘src_ip‘ to ‘dst_ip‘. Only the traffic from the client to the server will be matched by this rule.
- `<>`: Matches traffic in both directions (bidirectional). Suricata will internally duplicate the rule to cover both ‘src_ip‘ to ‘dst_ip‘ and ‘dst_ip‘ to ‘src_ip‘ traffic.

Note: There is no ’reverse’ style direction (i.e., no ‘<-‘).

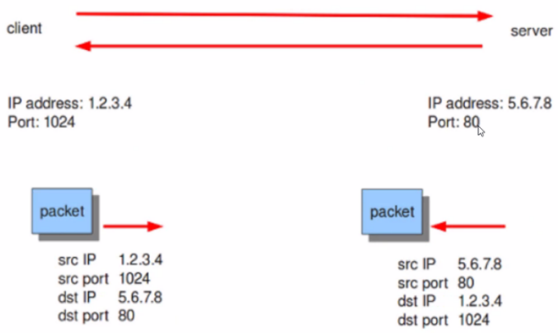

Figure 16: Explanation of Suricata rule directions.

6.5 Valid Suricata Actions

Suricata rules can specify different actions upon a match:

- `alert`: Generates an alert/log message. This is used in IDS mode.
- `pass`: Stops further inspection of the packet and considers it safe. Packets matching this rule are forwarded without any inspection by Suricata.
- `drop`: Silently drops the matching packet and generates an alert. The sender never gets notified that their packet was dropped. Used in IPS mode.
- `reject`: Sends an RST (Reset) for TCP traffic or an ICMP unreachable error message for UDP, actively terminating the connection. This makes it clear to the sender that their traffic is blocked. Used in IPS mode.

6.6 Suricata Modes: IDS vs. IPS

- **IDS (Intrusion Detection System) Mode:**
  - Monitors network traffic in real-time and generates alerts based on predefined rules (e.g., emerging threats, suspicious behaviors).
  - **Actions:** `pass`, `alert`. It observes and informs.
- **IPS (Intrusion Prevention System) Mode:**
  - When deployed inline (between the internal network and the internet, or between network segments), it can actively block malicious traffic.
  - **Actions:** `drop`, `reject`. It observes and actively intervenes.

6.7 Network Placement

The deployment of Suricata differs based on whether it's operating as an IDS or an IPS.

- **IPS Placement:** Deployed inline with network traffic (e.g., between a switch/router and the internet, or internal networks). All traffic flows through the IPS device, allowing it to actively block threats.
- **IDS Placement:** Often deployed with port mirroring (SPAN port) on a switch. A copy of all network traffic is sent to the IDS, allowing it to monitor without being in the direct path of the traffic. This prevents the IDS from becoming a single point of failure.

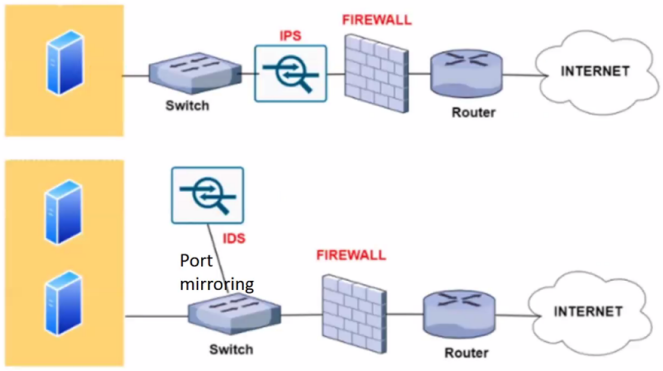

Figure 17: Suricata network placement for IPS (inline) and IDS (port mirroring).

7 Lab Walkthrough: Suricata Installation and Configuration

7.1 Initial Setup and Installation (on Ubuntu VM)

1. **Install `software-properties-common`:** This package helps in managing software repositories.

```
sudo apt-get install software-properties-common

```

Listing 7: Install software-properties-common

2. **Add Suricata Stable PPA:** Add the official Suricata PPA to get the latest stable version.

```
sudo add-apt-repository ppa:oisf/suricata-stable

```

Listing 8: Add Suricata PPA

3. **Update Package Lists Again:** After adding the PPA, update the package list to include Suricata packages.

```
sudo apt-get update

```

Listing 9: Update package lists again

4. **Install Suricata:**

```
sudo apt-get install suricata

```

Listing 10: Install Suricata

5. **Check Suricata Service Status:** Verify that Suricata is installed and running.

```
sudo systemctl status suricata

```

Listing 11: Check Suricata status

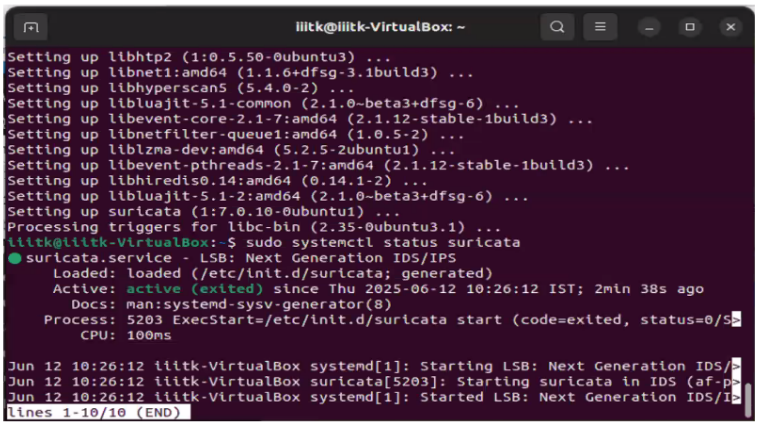

Figure 18: Suricata service status, showing 'active (exited)' initially as it's not permanently running.

7.2 Configuring `suricata.yaml`

The main configuration file for Suricata is `/etc/suricata/suricata.yaml`. We need to edit it to specify network interfaces and other settings.

1. **Open `suricata.yaml` for editing:**

```
sudo gedit /etc/suricata/suricata.yaml

```

Listing 12: Open suricata.yaml

2. **Configure `HOME_NET`:** Set the network range that Suricata should consider as your "home" or internal network. This is crucial for distinguishing internal and external traffic. In our lab, the home network is `192.168.17.0/24`.

```
# line 18
HOME_NET: "[192.168.17.0/24]"

```

Listing 13: Setting HOME_{NET}

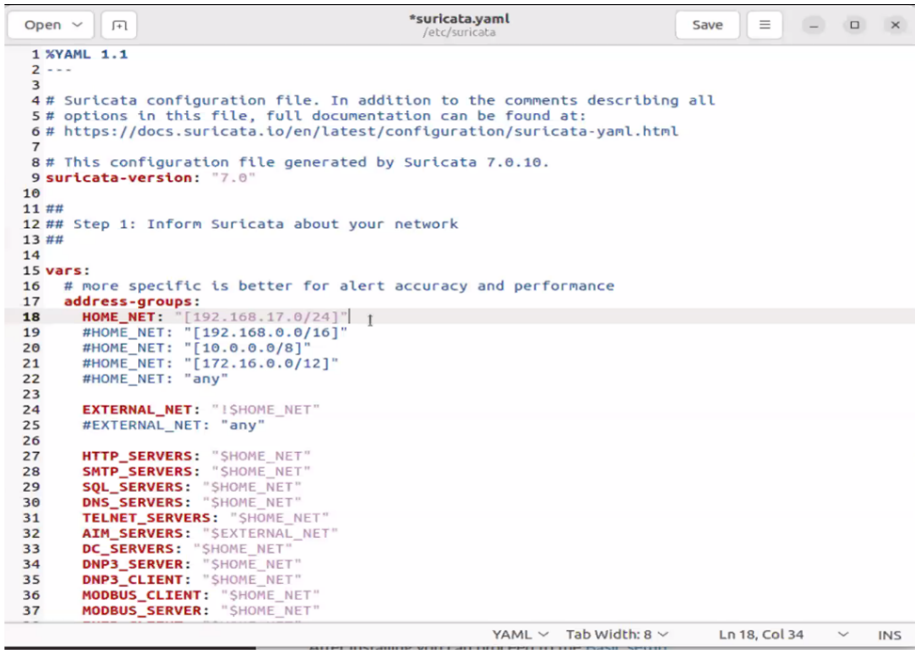

Figure 19: Modifying `HOME_NET` in `suricata.yaml`.

3. **Configure `af-packet` section:** This section specifies how Suricata captures packets from the network interface.
   - **Interface:** Set to your Ubuntu VM's network interface (e.g., `enp0s8`). You can find this using 'ifconfig' or 'ip a'.
   - `cluster-id`**:** A unique ID for this Suricata instance in a cluster. '99' is a common choice.
   - `cluster-type`**:** Set to 'cluster_flow' for flow-based packet load balancing.
   - `defrag`**:** Set to 'yes' to enable kernel-level IP defragmentation.

```
# line 621 onwards
af-packet:
  - interface: enp0s8
    cluster-id: 99
    cluster-type: cluster_flow
    defrag: yes
    # ... other options ...
    #promisc: yes # Ensure promiscuous mode is enabled for the interface
    #snaplen: 1518 # Recommended snaplen for full capture

```

Listing 14: Configuring af-packet

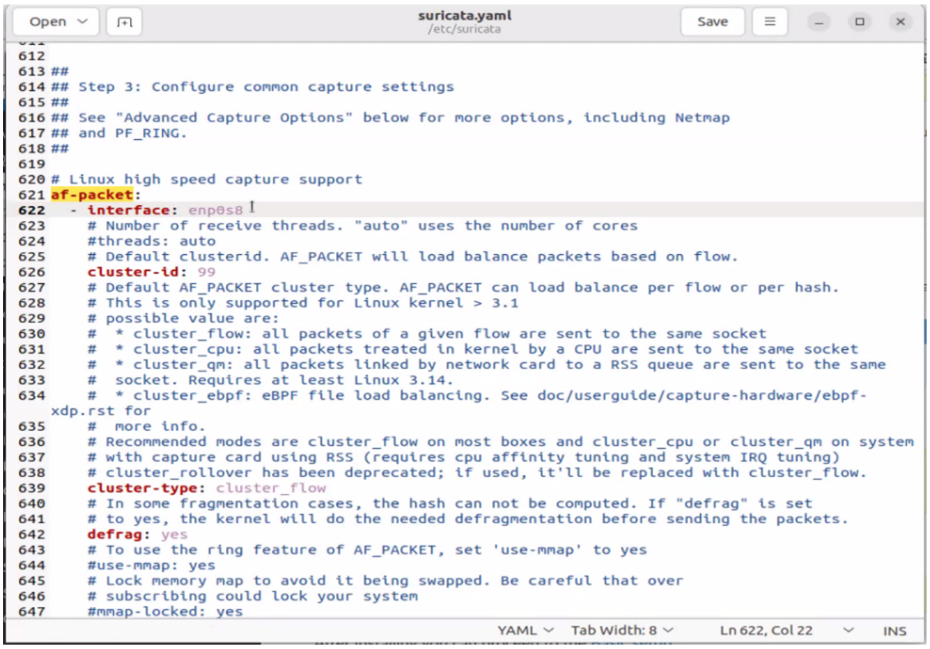

Figure 20: Configuring `af-packet` interface and cluster-id.

4. **Configure `pcap` section:** If using ‘pcap‘ capture method (instead of ‘af-packet‘ or ‘nfqueue‘), ensure ‘interface‘ is set and ‘promisc: yes‘ (promiscuous mode) is enabled.

```
# line 818 onwards
pcap:
  - interface: enp0s8
    # ... other options ...
    promisc: yes
    snaplen: 1518

```

Listing 15: Configuring pcap (optional

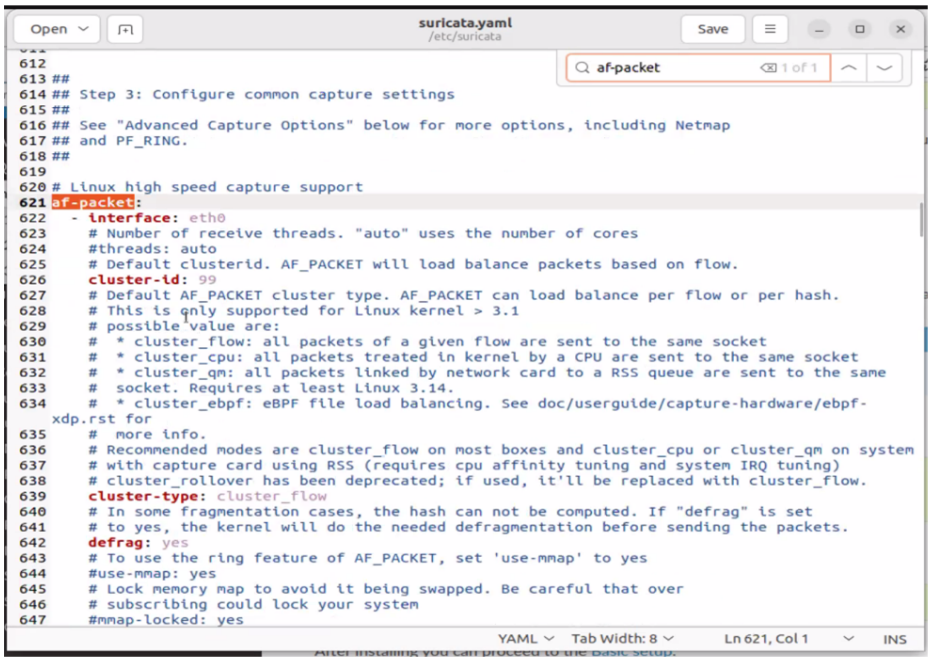

Figure 21: Initial `pcap` interface.

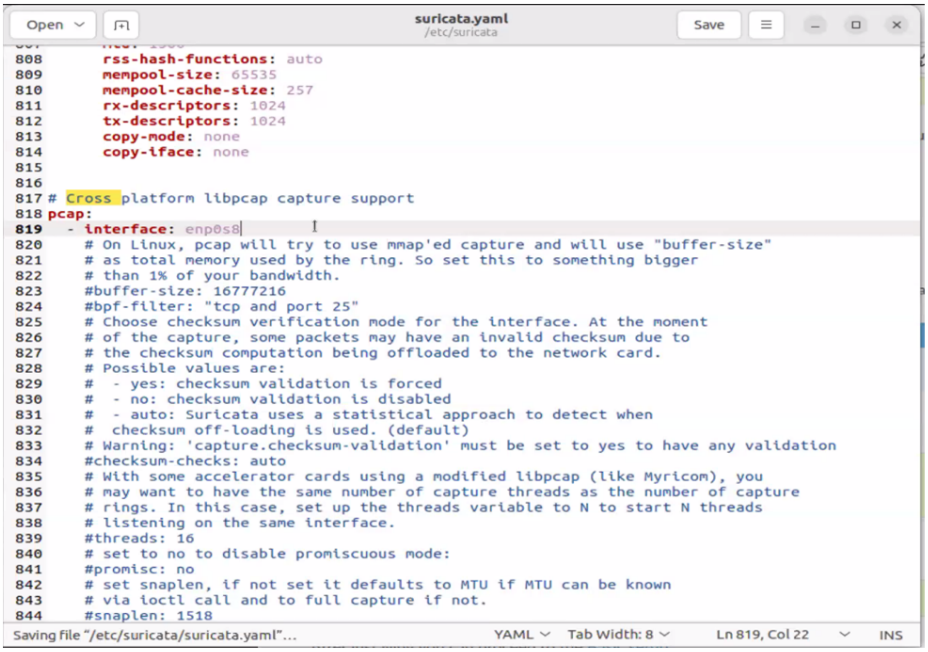

Figure 22: Configuring `pcap` interface.

5. **Enable `pcap-log`:** This feature captures all detected traffic to PCAP files, useful for later analysis in Wireshark. Set ‘enabled: yes‘.

```
# line 398 onwards
pcap-log:
  enabled: yes
  filename: log.pcap

```

Listing 16: Enabling pcap-log

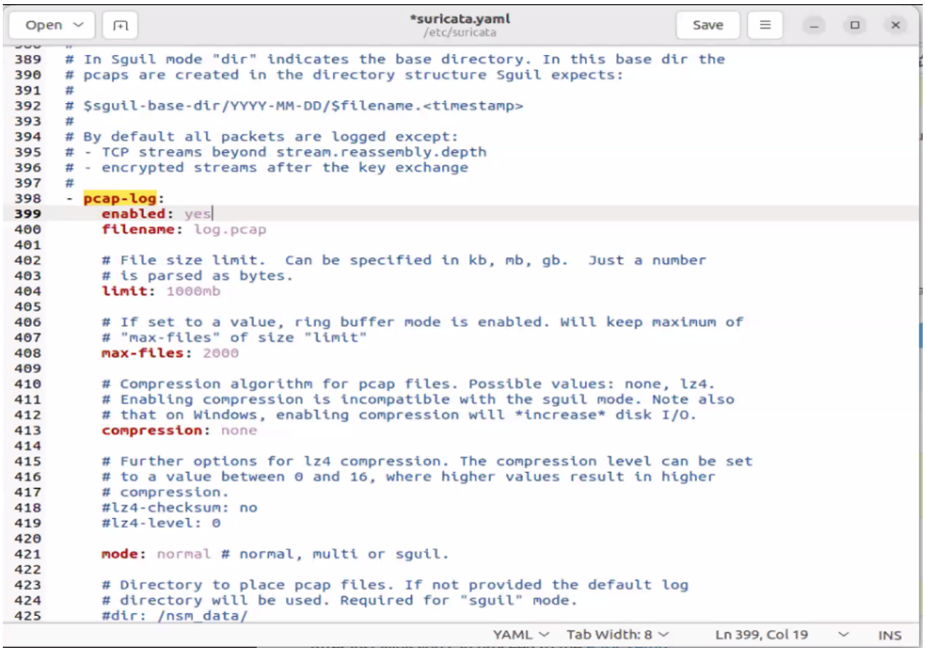

Figure 23: Enabling `pcap-log`.

6. **Include `local.rules`:** Ensure that Suricata loads your custom rules. Look for the 'rule-files' section and add 'local.rules'.

```
# line 2189 onwards
rule-files:
  - suricata.rules
  - local.rules

```

Listing 17: Including local.rules

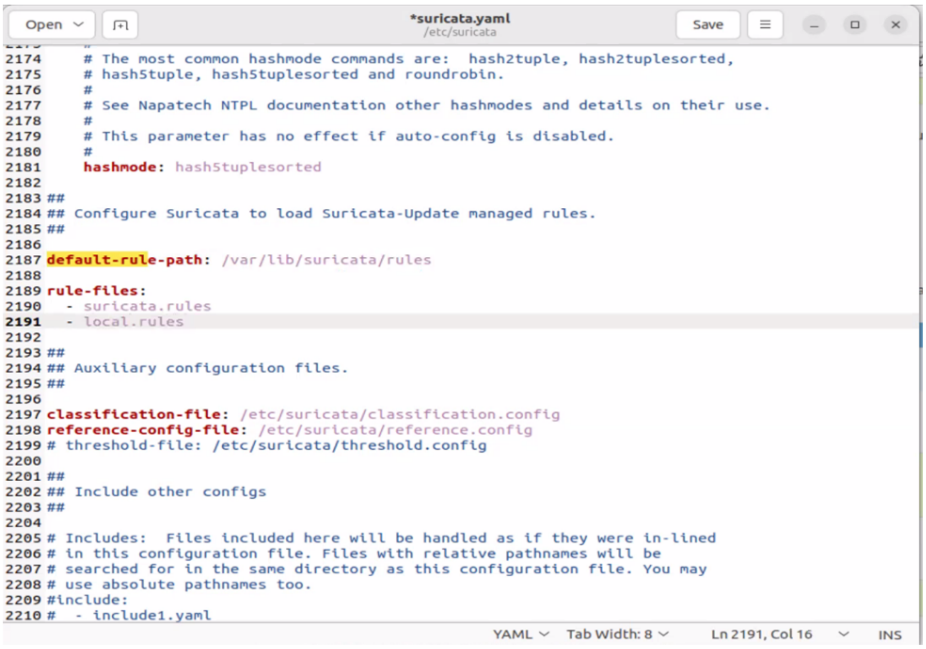

Figure 24: Ensuring 'local.rules' is included in `suricata.yaml`.

7. **Save and Close** the `suricata.yaml` file.

7.3 Starting Suricata Service

After modifying `suricata.yaml` and ensuring 'local.rules' is included, update Suricata rules. This command downloads and compiles the latest community rulesets and integrates them with your custom rules.

```
sudo suricata-update
```

Listing 18: Updating Suricata rules

7.4 Testing Suricata Configuration

It's vital to test the configuration before starting Suricata in production mode to catch any errors.

```
sudo suricata -T -c /etc/suricata/suricata.yaml -v
```

Listing 19: Test Suricata configuration

A successful test will end with "Configuration provided was successfully loaded. Exiting." Also implement the rules by staring Suricata service using the command.

```
sudo systemctl start service suricata
```

Listing 20: Start Suricata

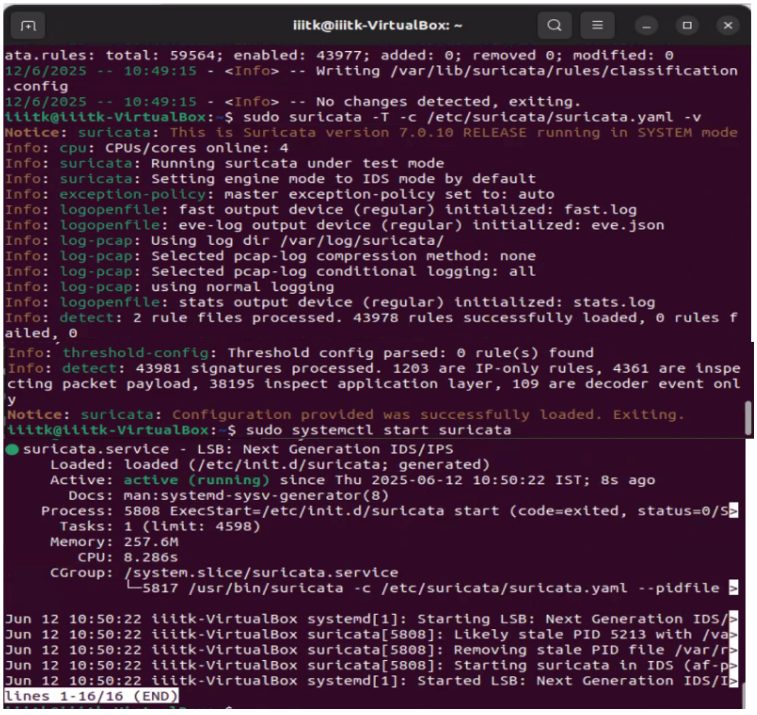

Figure 25: Testing Suricata configuration.

8 Lab Walkthrough: Suricata in IDS Mode

In IDS mode, Suricata will monitor traffic and generate alerts based on rules, but it won't actively block any packets.

8.1 Creating Custom Rules (`local.rules`)

We'll create a new rule file 'local.rules' to define custom alerts.

1. **Create/Open `local.rules`:**

```
sudo gedit /var/lib/suricata/rules/local.rules

```

Listing 21: Open local.rules for editing

2. **Add the following rules:**

   - **ICMP Ping Alert:** This rule will generate an alert whenever there's an ICMP (ping) request from the Kali machine to any device on the home network.
   - **SSH Login Attempt Alert:** This rule will alert on any TCP traffic targeting port 22 (SSH) from Kali to the home network.
   - **Facebook Access Attempt Alert:** This rule will alert on any TCP traffic from the home network attempting to connect to Facebook's IP (`57.144.156.1`) on port 443 (HTTPS).

```
alert icmp 192.168.17.24 any -> $HOME_NET any (msg:"Ping from Kali";sid
    :10001;rev:1;)
alert tcp 192.168.17.24 any -> $HOME_NET 22 (msg:"SSH Login attempt from
    kali";sid:10002;rev:1;)
alert tcp $HOME_NET any -> 57.144.156.1 443 (msg:"Facebook access attempt";
    sid:10003;rev:1;)

```

Listing 22: Content of local.rules for IDS mode

3. **Save and Close `local.rules`.**

8.2 Starting Suricata in IDS Mode

1. **Start Suricata:**

```
sudo systemctl start suricata

```

Listing 23: Start Suricata service

Figure 26: Starting Suricata in IDS mode.

8.3 Generating Traffic and Observing Alerts

From the Kali machine, generate different types of traffic to trigger the rules.

1. **Ping the Ubuntu machine (Suricata):**

```
ping 192.168.17.34

```

Listing 24: Pinging from Kali to Ubuntu

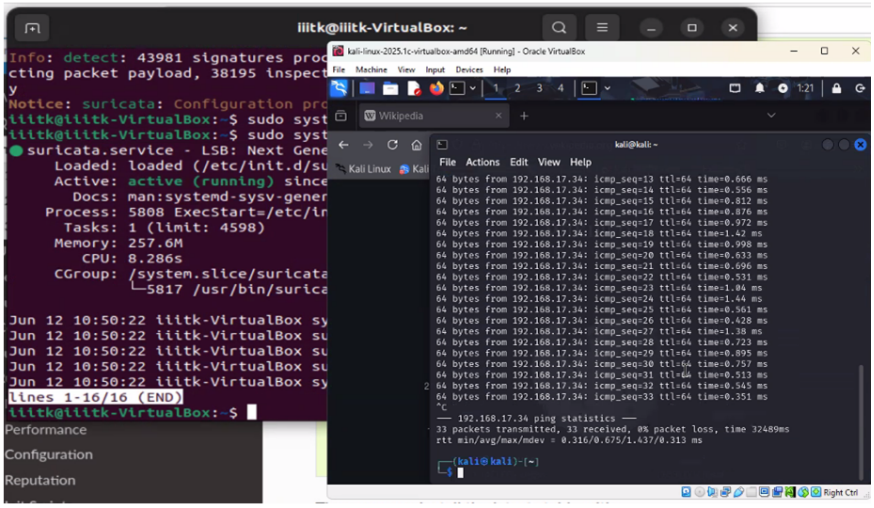

Figure 27: Ping command from Kali to Ubuntu.

2. **Attempt SSH login from Kali to Ubuntu:**

```
sudo apt install openssh-server
sudo systemtcl start service openssh

```

Listing 25: SSH initialisation in Ubuntu

```
ssh iiitk@192.168.17.34

```

Listing 26: SSH attempt from Kali to Ubuntu

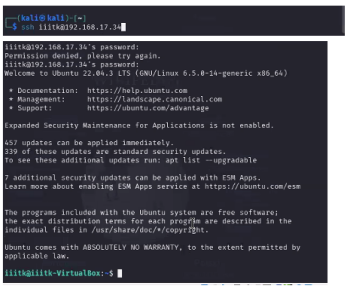

Figure 28: SSH login attempt from Kali to Ubuntu.

3. **Perform DNS lookup for facebook.com (from Ubuntu, to test Facebook rule):**

```
nslookup facebook.com

```

Listing 27: DNS lookup for facebook.com on Ubuntu

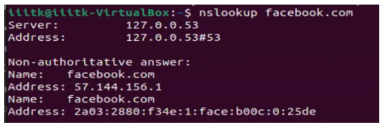

Figure 29: DNS lookup for facebook.com on Ubuntu.

4. **View Suricata's `fast.log`:** This log file records the alerts generated by Suricata.

```
sudo cat /var/log/suricata/fast.log

```

Listing 28: View Suricata fast.log

You should see entries like "Ping from Kali" and "SSH Login attempt from kali" and "Facebook access attempt".

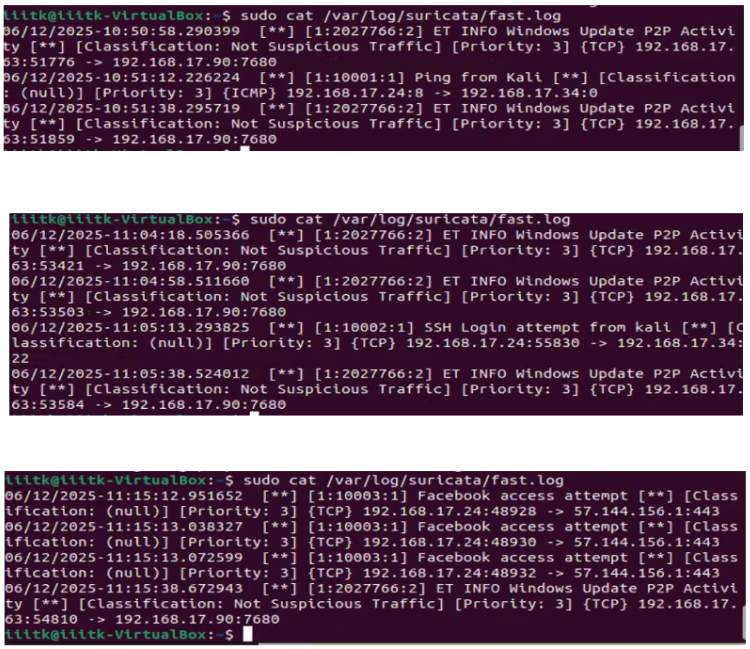

Figure 30: `fast.log` showing alerts for Ping, SSH, and Facebook access.

5. **Analyze PCAP log with Wireshark:** Suricata can capture packets to a PCAP file (`log.pcap.*`). You can open this file in Wireshark to inspect the traffic that triggered the alerts.

```
sudo wireshark /var/log/suricata/log.pcap.1749705639 # The filename changes
    with timestamp

```

Listing 29: Open PCAP log in Wireshark

6. **Truncate logs :** This command can be used to clear the log and pcap files.

```
sudo truncate -s 0 /var/log/suricata/fast.log
sudo truncate -s 0 /var/log/suricata/eve.json
sudo rm -r /var/log/suricata/log.pcap.*

```

Listing 30: Open PCAP log in Wireshark

9 Lab Walkthrough: Suricata in IPS Mode

To transition Suricata into IPS mode, we need to adjust its network configuration and modify rule actions from ‘alert‘ to ‘drop‘ or ‘reject‘.

9.1 Configuring for IPS Mode

1. **Modify `/etc/default/suricata`:** This file configures how Suricata starts.

```
sudo gedit /etc/default/suricata

```

Listing 31: Open /etc/default/suricata

Change the ‘LISTENMODE‘ to ‘nfqueue‘ and specify the network interface.

```
# line 4
RUN=yes

# line 17
LISTENMODE=nfqueue

# line 20
IFACE=enp0s8 # Use your specific interface name

# line 23
NFQUEUE="-q 0"

```

Listing 32: Configuring /etc/default/suricata for IPS

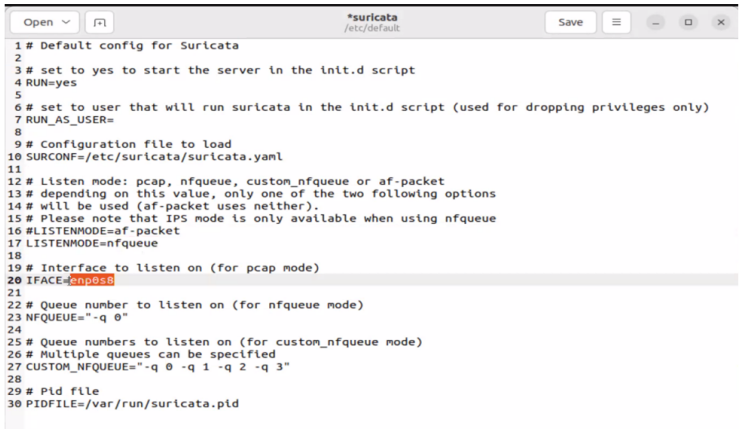

Figure 31: Modifying `/etc/default/suricata` for IPS mode (LISTENMODE, IFACE, NFQUEUE).

2. **Modify `/etc/ufw/before.rules`:** For Suricata to operate in ‘nfqueue‘ mode, Uncomplicated Firewall (UFW) needs to redirect packets to the ‘nfqueue‘.

```
sudo gedit /etc/ufw/before.rules

```

Listing 33: Open /etc/ufw/before.rules

Add the following lines under the ‘*filter‘ section (e.g., around line 19), which tells UFW to send all input and output traffic to ‘nfqueue‘.

```
# suricata
-I INPUT -j NFQUEUE
-I OUTPUT -j NFQUEUE

```

Listing 34: Adding NFQUEUE rules to /etc/ufw/before.rules

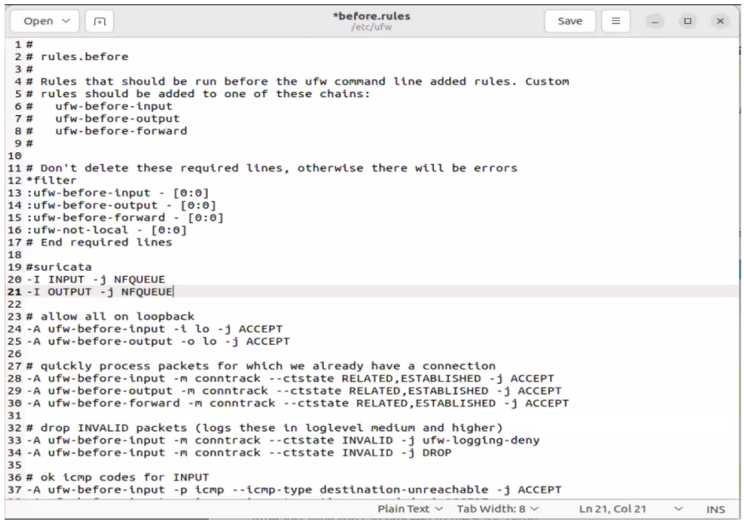

Figure 32: Adding NFQUEUE rules to `/etc/ufw/before.rules`.

3. **Enable UFW:**

```
sudo ufw enable

```

Listing 35: Enable UFW

Confirm if prompted.

9.2 Modifying Rules for Blocking

Now, change the actions in ‘local.rules‘ from ‘alert‘ to ‘drop‘ to actively block traffic.

1. **Open `local.rules` for editing:**

```
sudo gedit /var/lib/suricata/rules/local.rules

```

Listing 36: Open local.rules for editing

2. **Modify rules to use ‘drop‘ action:**
   - Change ‘alert icmp‘ to ‘drop icmp‘.
   - Change ‘alert tcp‘ to ‘drop tcp‘ for SSH and Facebook access rules.
   - Add a rule to detect and drop potential SYN floods.

```
drop icmp 192.168.17.24 any -> $HOME_NET any (msg:"Ping from Kali";sid
    :10001;rev:1;)
drop tcp 192.168.17.24 any -> $HOME_NET 22 (msg:"SSH Login attempt from
    kali";sid:10002;rev:1;)
drop tcp $HOME_NET any -> 57.144.156.1 443 (msg:"Facebook access attempt";
    sid:10003;rev:1;)
drop tcp any any -> $HOME_NET any (msg:"Potential SYN Flood"; flags:S;
    threshold:type threshold, track by_src, count 10, seconds 12; sid
    :10004;rev:1;)

```

Listing 37: Content of local.rules for IPS mode

3. **Save and Close `local.rules`.**
4. **Run `suricata-update` again** after rule modification to ensure new rules are loaded.

```
sudo suricata-update

```

Listing 38: Update Suricata rules after modification

5. **Test Suricata configuration again:**

```
sudo suricata -T -c /etc/suricata/suricata.yaml -v

```

Listing 39: Test Suricata configuration

9.3 Testing IPS Functionality

1. **Clear Suricata logs and PCAPs:**

```
sudo truncate -s 0 /var/log/suricata/fast.log
sudo truncate -s 0 /var/log/suricata/eve.json
sudo rm -r /var/log/suricata/log.pcap.*

```

Listing 40: Clear Suricata logs and PCAPs

2. **Stop Suricata, then Start it again:**

```
sudo systemctl stop suricata
sudo systemctl start suricata

```

Listing 41: Restart Suricata for IPS mode

3. **Attempt to Ping from Kali to Ubuntu:**

```
ping 192.168.17.34

```

Listing 42: Pinging from Kali to Ubuntu (should be blocked)

You should see "Destination Host Unreachable" or 100% packet loss, indicating Suricata dropped the ICMP packets.

Figure 33: Ping from Kali to Ubuntu, showing 100% packet loss due to IPS blocking.

```
sudo cat /var/log/suricata/fast.log

```

Listing 43: View fast.log for ICMP drop alert

You will see a '[Drop]' entry in 'fast.log' for the 'Ping from Kali' rule.

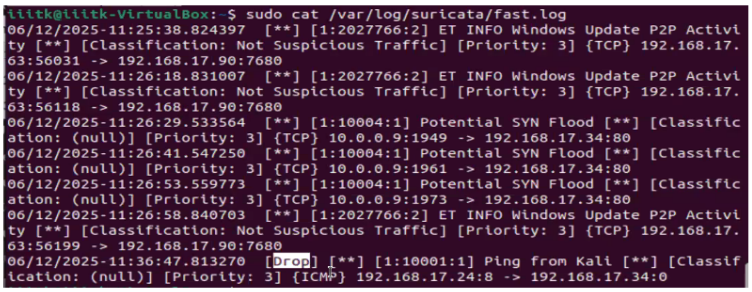

Figure 34: `fast.log` showing '[Drop]' alert for ICMP ping.

4. **Attempt SSH login from Kali to Ubuntu:**

```
ssh iiitk@192.168.17.34

```

Listing 44: SSH attempt from Kali to Ubuntu (should be blocked)

The SSH connection should fail or time out, indicating the SSH traffic is being dropped.

```
sudo cat /var/log/suricata/fast.log

```

Listing 45: View fast.log for SSH drop alert

You will see a '[Drop]' entry for the 'SSH Login attempt from kali' rule.

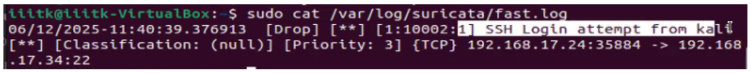

Figure 35: `fast.log` showing '[Drop]' alert for SSH login attempt.

5. **Perform a DOS attack using `hping3` from Kali to Ubuntu:** This will test the SYN flood rule.

```
sudo hping3 -S -a 10.0.0.9 -p 80 192.168.17.34

```

Listing 46: SYN Flood with hping3

This command sends SYN packets to '192.168.17.34' from a spoofed IP '10.0.0.9'. The output should show 100% packet loss, indicating Suricata is dropping the SYN packets.

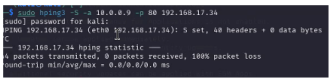

Figure 36: hping3 SYN Flood from Kali, showing 100% packet loss.

```
sudo cat /var/log/suricata/fast.log

```

Listing 47: View fast.log for SYN Flood drop alert

You should see ‘[Drop]‘ alerts for "Potential SYN Flood".

These labs demonstrate the fundamental concepts of MITM attacks (specifically ARP spoofing) and how Suricata can be deployed and configured to detect and prevent such network-based threats.