

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)	3
1.1 Types of MITM Attacks	3
2 Understanding ARP (Address Resolution Protocol)	4
2.1 What is ARP?	4
2.2 ARP Analogy (Apartment Building)	4
2.3 ARP in Action	4
2.4 ARP Packet Creation (Cisco Packet Tracer)	4
3 ARP Spoofing & MITM Attacks	6
3.1 Why ARP is Vulnerable?	6
3.2 The Man-in-the-Middle	7
4 Lab Walkthrough: ARP Spoofing with Ettercap	7
4.1 Lab Setup	7
4.2 Pre-Spoofing Preparation	7
4.2.1 Clearing ARP Cache	8
4.2.2 Initial ARP Table Verification	8
4.3 Ettercap Configuration	8
4.3.1 Enable IP Forwarding on Kali	8
4.3.2 Start Ettercap and Scan for Hosts	8
4.3.3 Select Targets for ARP Poisoning	9
4.3.4 Start ARP Poisoning Attack	10
4.4 Post-Spoofing Verification	11
5 DNS Spoofing with Ettercap	12
5.1 Enabling DNS Spoofing	12
5.2 Modifying <code>etter.dns</code> file	13
6 Suricata: Intrusion Detection/Prevention System (IDS/IPS)	14
6.1 Introduction to Suricata	14
6.2 Suricata Requirements (VirtualBox Setup)	14
6.3 Suricata Rule/Signature Structure	14
6.4 Rule Direction	15
6.5 Valid Suricata Actions	15
6.6 Suricata Modes: IDS vs. IPS	15
6.7 Network Placement	16
7 Lab Walkthrough: Suricata Installation and Configuration	16
7.1 Initial Setup and Installation (on Ubuntu VM)	16
7.2 Configuring <code>suricata.yaml</code>	17
7.3 Starting Suricata Service	22
7.4 Testing Suricata Configuration	22
8 Lab Walkthrough: Suricata in IDS Mode	22
8.1 Creating Custom Rules (<code>local.rules</code>)	23
8.2 Starting Suricata in IDS Mode	23
8.3 Generating Traffic and Observing Alerts	24
9 Lab Walkthrough: Suricata in IPS Mode	26
9.1 Configuring for IPS Mode	27
9.2 Modifying Rules for Blocking	28
9.3 Testing IPS Functionality	29

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.

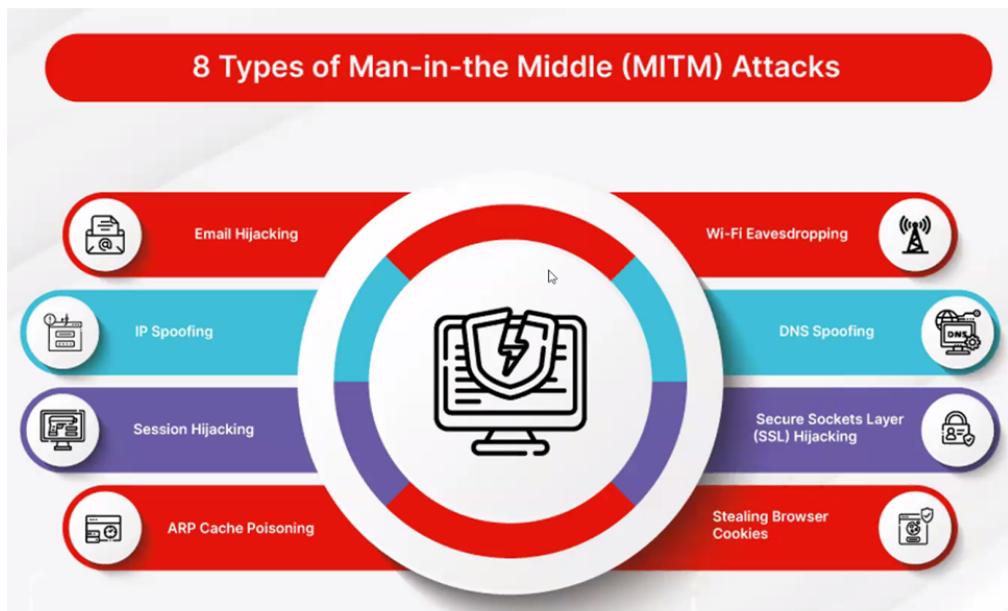


Figure 1: 8 Types of Man-in-the-Middle (MITM) Attacks

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 (FFFFFFFFFF), which signifies a broadcast address. This ensures that all devices on the local network receive the ARP Request.

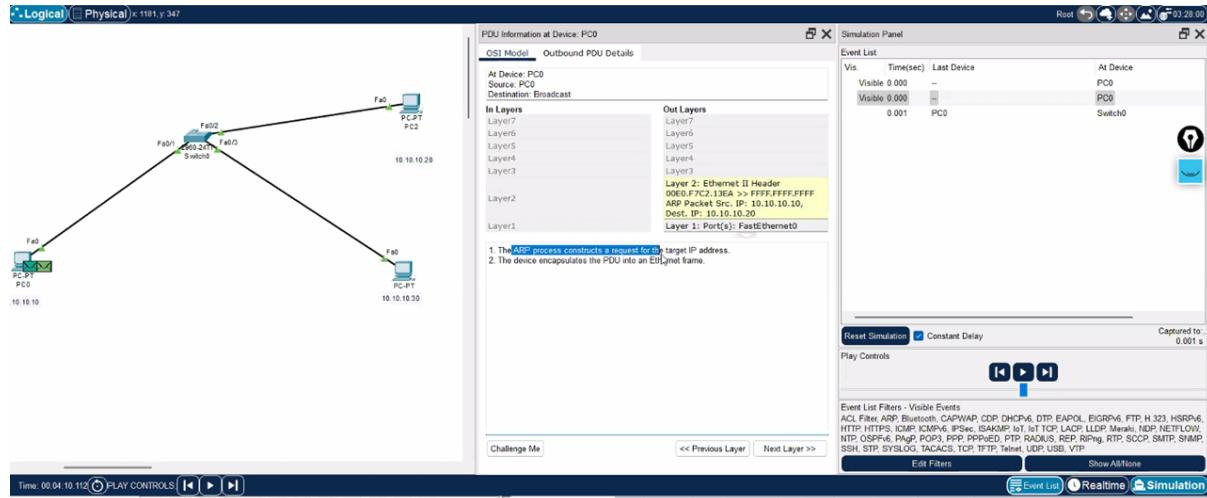


Figure 2: PDU Information showing destination MAC initially set to broadcast (all F's).

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.

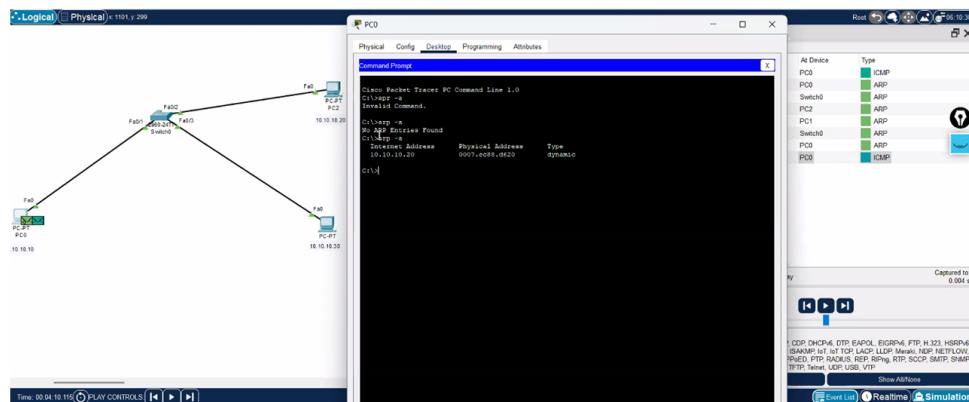


Figure 3: ICMP response.

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

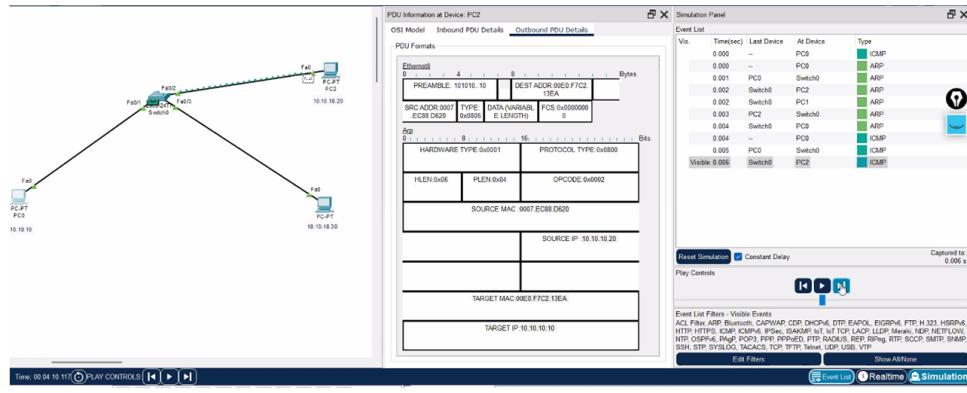


Figure 4: ARP Broadcast

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.

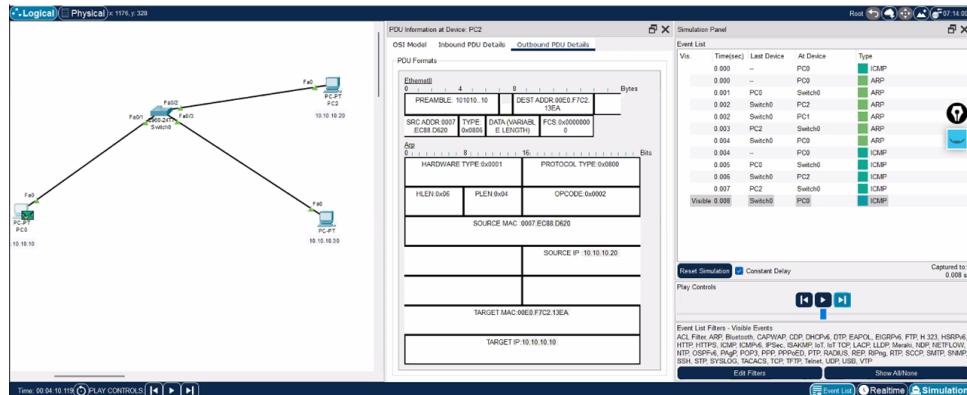


Figure 5: Packet simulation completed.

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.

The figure consists of three vertically stacked terminal windows. The top window is titled 'Select Command Prompt' and shows the output of 'ipconfig' for a Windows 10 Server. It lists an Ethernet adapter with an IPv4 address of 192.168.15.8, subnet mask 255.255.255.0, and default gateway 192.168.15.8. The middle window is titled 'Select C:\Windows\system32\cmd.exe' and shows the output of 'ipconfig' for a Windows 10 Client. It lists an Ethernet adapter with an IPv4 address of 192.168.15.6, subnet mask 255.255.255.0, and default gateway 192.168.15.8. The bottom window is titled '(kali㉿kali)-[~]' and shows the output of 'ifconfig' for a Kali Linux Attacker. It lists two interfaces: eth0 (IPv4 192.168.15.7, subnet 255.255.255.0, broadcast 192.168.15.255) and eth0:1 (IPv6 fe80::c6a3:6801:a668:3f32). The Kali interface also shows statistics for RX and TX packets.

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:

```
1 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`.

```
1 ping 192.168.15.6
2 ping 192.168.15.7
3 arp -a
```

Listing 2: Pinging and verifying ARP table on Server

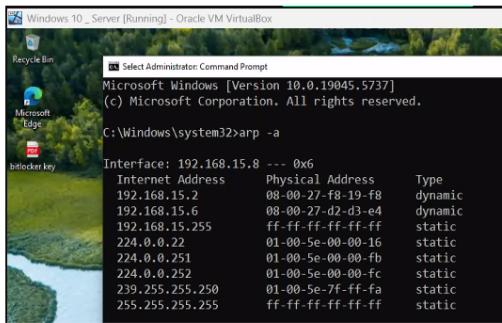


Figure 7: Server's initial ARP table entries after pings.

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.

```
1 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.

```
1 sudo ettercap -G
```

Listing 4: Starting Ettercap

Alternatively gui application could be directly opened and setup

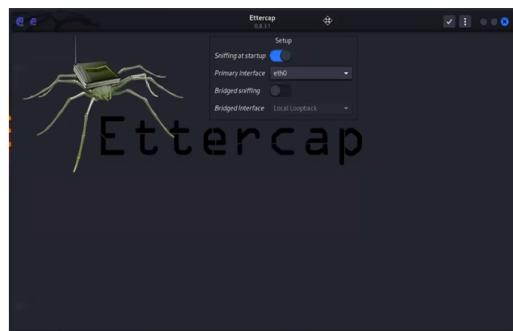


Figure 8: Ettercap sniffing setup and interface selection.

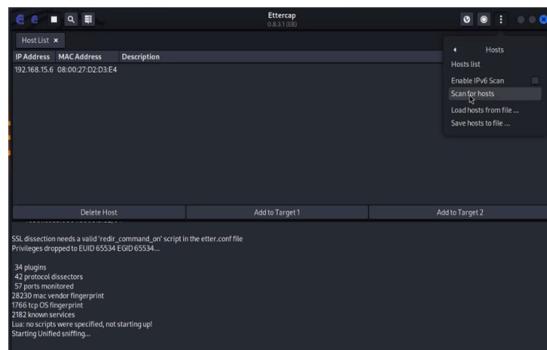


Figure 9: Ettercap GUI: Scanning for hosts.

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.

IP Address	MAC Address	Description
192.168.15.2	08:00:27:F8:19:F8	DHCP Server virtual adaptor
192.168.15.3	0A:00:27:00:00:05	win client
192.168.15.6	08:00:27:D2:D3:E4	win Server
192.168.15.8	08:00:27:6A:31:02	

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

IP Address	MAC Address	Description
192.168.15.2	08:00:27:F8:19:F8	
192.168.15.3	0A:00:27:00:00:05	
192.168.15.6	08:00:27:D2:D3:E4	
192.168.15.8	08:00:27:6A:31:02	

Searching the whole network for <2> hosts...
4 hosts added to the hosts list...

```
DHCP: [08:00:27:C7:99:78] REQUEST 192.168.15.7
DHCP: [08:00:27:6A:31:03] REQUEST 192.168.15.8
DHCP: [08:00:27:6A:31:03] ACK 192.168.15.7 255.255.255.0 GW invalid
DHCP: [08:00:27:20:00:05] REQUEST 192.168.15.3
DHCP: [08:00:27:6A:31:03] REQUEST 192.168.15.8
DHCP: [08:00:27:C7:99:78] REQUEST 192.168.15.7
DHCP: [08:00:27:6A:31:03] ACK 192.168.15.7 255.255.255.0 GW invalid
DHCP: [08:00:27:20:00:05] REQUEST 192.168.15.3
DHCP: [08:00:27:6A:31:03] REQUEST 192.168.15.8
DHCP: [08:00:27:6A:31:03] ACK 192.168.15.3 255.255.255.0 GW invalid
DHCP: [08:00:27:6A:31:03] ACK 192.168.15.8 255.255.255.0 GW invalid
Host 192.168.15.6 added to TARGET
```

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".

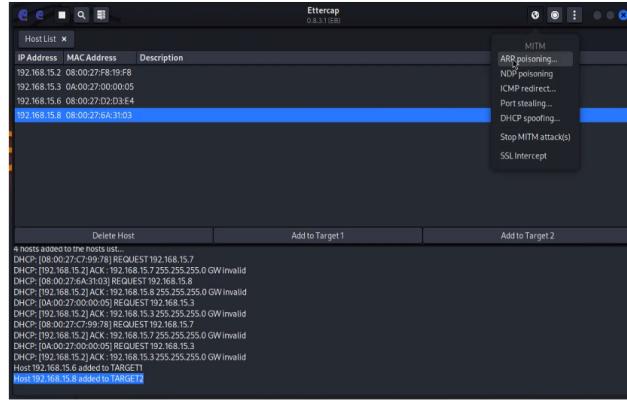


Figure 12: Starting ARP poisoning attack in Ettercap.

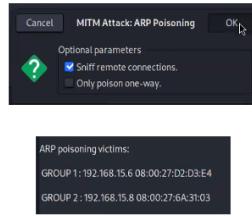


Figure 13: ARP poisoning victims confirmation.

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).

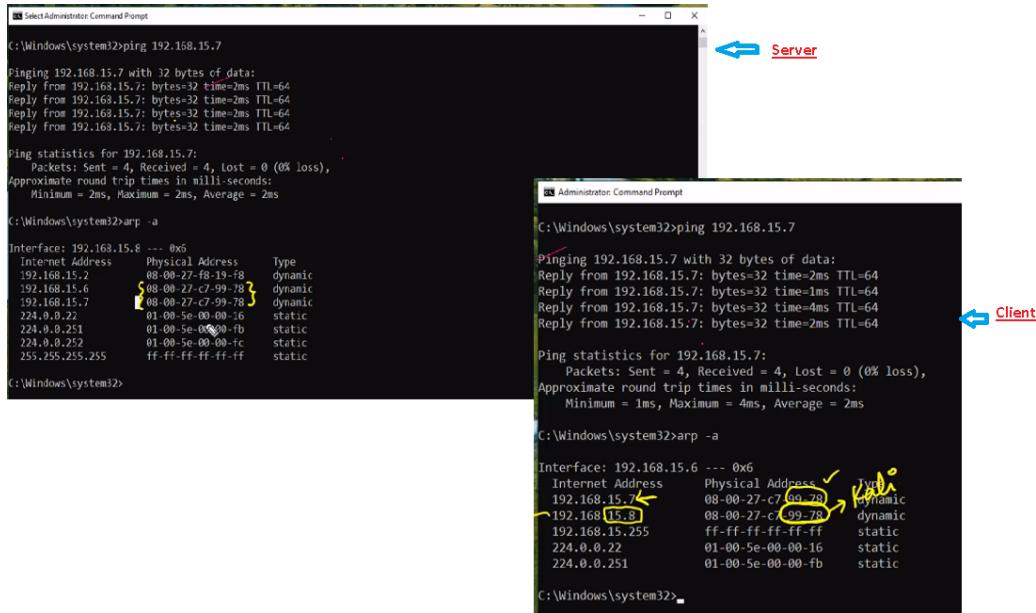


Figure 14: ARP tables on Server and Client after ARP poisoning. Note the altered MAC addresses.

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.

```
1 ping 192.168.15.6
2 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.

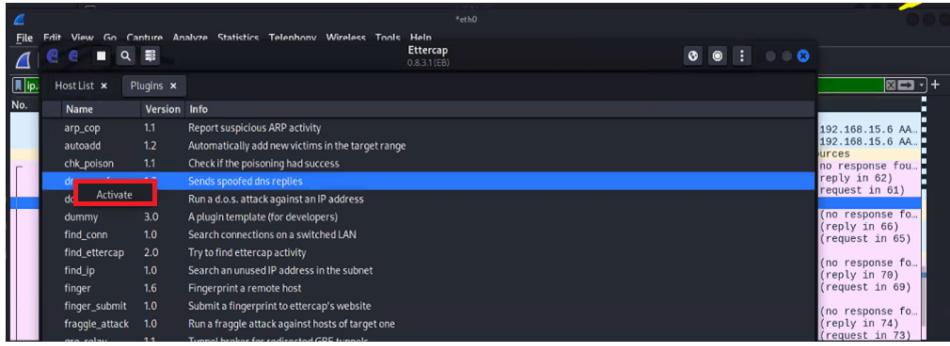


Figure 15: Enabling DNS Spoofing plugin in Ettercap.

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 ‘bank.com’ to the Kali machine’s IP (192.168.15.7), you would add the following line:

```
1 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:

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

Example:

```
1 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 '<-').

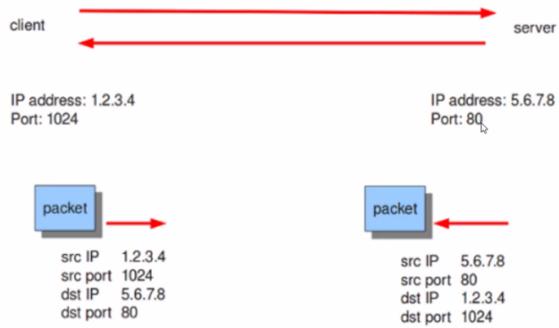


Figure 16: Explanation of Suricata rule directions.

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.

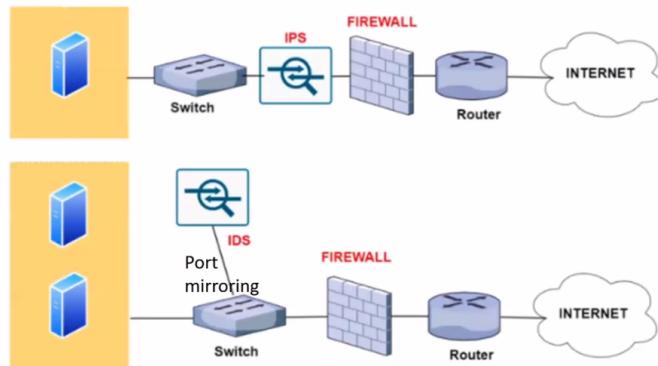


Figure 17: Suricata network placement for IPS (inline) and IDS (port mirroring).

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.

```
1 sudo apt-get install software-properties-common  
2
```

Listing 7: Install software-properties-common

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

```
1 sudo add-apt-repository ppa:oisf/suricata-stable  
2
```

Listing 8: Add Suricata PPA

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

```
1 sudo apt-get update  
2
```

Listing 9: Update package lists again

4. **Install Suricata:**

```
1 sudo apt-get install suricata  
2
```

Listing 10: Install Suricata

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

```
1 sudo systemctl status suricata  
2
```

Listing 11: Check Suricata status

```
iiitk@iiitk-VirtualBox: ~  
Setting up libhttp2 (1:0.5.50-0ubuntu3) ...  
Setting up libneti:amd64 (1.1.6+dfsg-3.1build3) ...  
Setting up libhyperscans (5.4.0-2) ...  
Setting up libluajit-5.1-common (2.1.0-beta3+dfsg-6) ...  
Setting up libevent-core-2.1-7:amd64 (2.1.12-stable-1build3) ...  
Setting up libnetfilter-queue1:amd64 (1.0.5-2) ...  
Setting up liblzma-dev:amd64 (5.2.5-2ubuntu1) ...  
Setting up libevent-pthreads-2.1-7:amd64 (2.1.12-stable-1build3) ...  
Setting up libiredis0.14:amd64 (0.14.1-2) ...  
Setting up libluajit-5.1-2:amd64 (2.1.0-beta3+dfsg-6) ...  
Setting up suricata (1:7.0.10-0ubuntu1) ...  
Processing triggers for libc-bin (2.35-0ubuntu3.1) ...  
iiitk@iiitk-VirtualBox: $ sudo systemctl status suricata  
● suricata.service - LSB: Next Generation IDS/IPS  
    Loaded: loaded (/etc/init.d/suricata; generated)  
      Active: active (exited) since Thu 2025-06-12 10:26:12 IST; 2min 38s ago  
        Docs: man:systemd-sysv-generator(8)  
    Process: 5203 ExecStart=/etc/init.d/suricata start (code=exited, status=0/success)  
       CPU: 100ms  
  
Jun 12 10:26:12 iiitk-VirtualBox systemd[1]: Starting LSB: Next Generation IDS/IPS  
Jun 12 10:26:12 iiitk-VirtualBox suricata[5203]: Starting suricata in IDS (af-packet)  
Jun 12 10:26:12 iiitk-VirtualBox systemd[1]: Started LSB: Next Generation IDS/IPS  
lines 1-10/10 (END)
```

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:

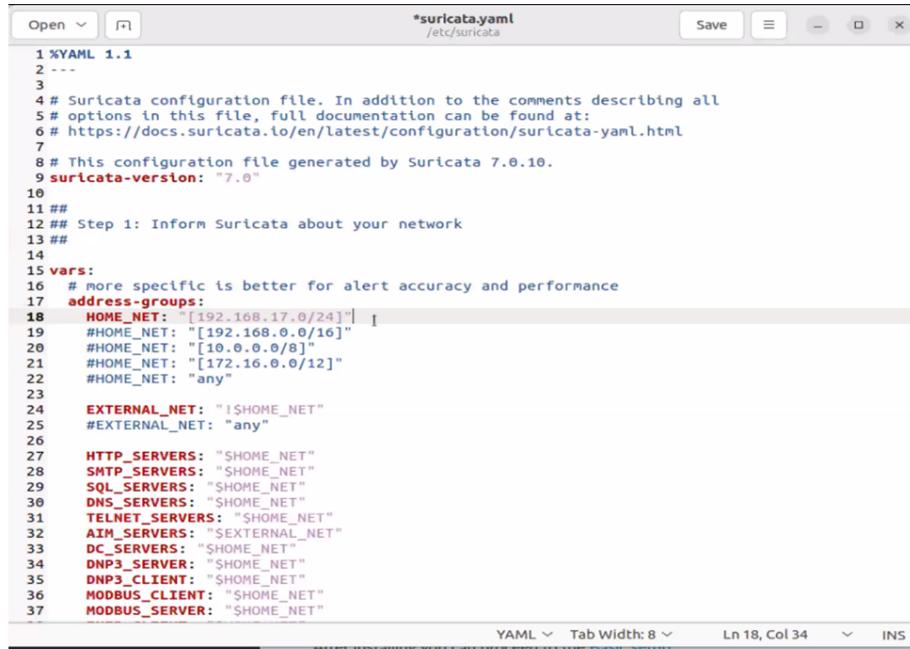
```
1 sudo gedit /etc/suricata/suricata.yaml  
2
```

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`.

```
1 # line 18  
2 HOME_NET: "[192.168.17.0/24]"  
3
```

Listing 13: Setting `HOME_NET`



```
1 %YAML 1.1
2 ---
3
4 # Suricata configuration file. In addition to the comments describing all
5 # options in this file, full documentation can be found at:
6 # https://docs.suricata.io/en/latest/configuration/suricata-yaml.html
7
8 # This configuration file generated by Suricata 7.0.10.
9 suricata-version: "7.0"
10
11 ##
12 ## Step 1: Inform Suricata about your network
13 ##
14
15 vars:
16   # more specific is better for alert accuracy and performance
17   address-groups:
18     HOME_NET: "[192.168.17.0/24]" | [192.168.0.0/16]
19     #HOME_NET: "[192.168.0.0/16]"
20     #HOME_NET: "[10.0.0.0/8]"
21     #HOME_NET: "[172.16.0.0/12]"
22     #HOME_NET: "any"
23
24     EXTERNAL_NET: "!$HOME_NET"
25     #EXTERNAL_NET: "any"
26
27     HTTP_SERVERS: "$HOME_NET"
28     SMTP_SERVERS: "$HOME_NET"
29     SQL_SERVERS: "$HOME_NET"
30     DNS_SERVERS: "$HOME_NET"
31     TELNET_SERVERS: "$HOME_NET"
32     AIM_SERVERS: "$EXTERNAL_NET"
33     DC_SERVERS: "$HOME_NET"
34     DNP3_SERVER: "$HOME_NET"
35     DNP3_CLIENT: "$HOME_NET"
36     MODBUS_CLIENT: "$HOME_NET"
37     MODBUS_SERVER: "$HOME_NET"
```

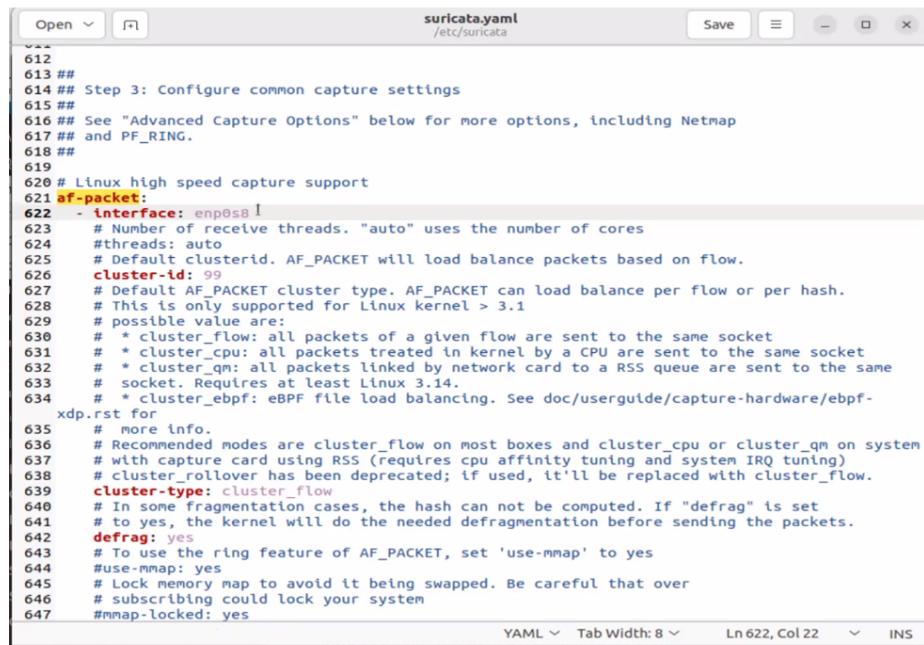
Figure 19: Modifying HOME_NET in `suricata.yaml`.

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.

```
1 # line 621 onwards
2 af-packet:
3   - interface: enp0s8
4     cluster-id: 99
5     cluster-type: cluster_flow
6     defrag: yes
7     # ... other options ...
8     #promisc: yes # Ensure promiscuous mode is enabled for the interface
9     #snaplen: 1518 # Recommended snaplen for full capture
10
```

Listing 14: Configuring af-packet



```

112
113 ##
114 ## Step 3: Configure common capture settings
115 ##
116 ## See "Advanced Capture Options" below for more options, including Netmap
117 ## and PF_RING.
118 ##
119
120 # Linux high speed capture support
121 af-packet:
122   - interface: enp0s8
123     # Number of receive threads. "auto" uses the number of cores
124     #threads: auto
125     # Default clusterid. AF_PACKET will load balance packets based on flow.
126     cluster-id: 99
127     # Default AF_PACKET cluster type. AF_PACKET can load balance per flow or per hash.
128     # This is only supported for Linux kernel > 3.1
129     # possible value are:
130     # * cluster_flow: all packets of a given flow are sent to the same socket
131     # * cluster_cpu: all packets treated in kernel by a CPU are sent to the same socket
132     # * cluster_qm: all packets linked by network card to a RSS queue are sent to the same
133     #   socket. Requires at least Linux 3.14.
134     # * cluster_ebpf: eBPF file load balancing. See doc/userguide/capture-hardware/ebpf-
135       xdp.rst for
136       # more info.
137       # Recommended modes are cluster_flow on most boxes and cluster_cpu or cluster_qm on system
138       # with capture card using RSS (requires cpu affinity tuning and system IRQ tuning)
139       # cluster_rollover has been deprecated; if used, it'll be replaced with cluster_flow.
140     cluster-type: cluster_flow
141     # In some fragmentation cases, the hash can not be computed. If "defrag" is set
142     # to yes, the kernel will do the needed defragmentation before sending the packets.
143     defrag: yes
144     # To use the ring feature of AF_PACKET, set 'use-mmap' to yes
145     use-mmap: yes
146     # Lock memory map to avoid it being swapped. Be careful that over
147     # subscribing could lock your system
148     #mmap-locked: yes

```

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

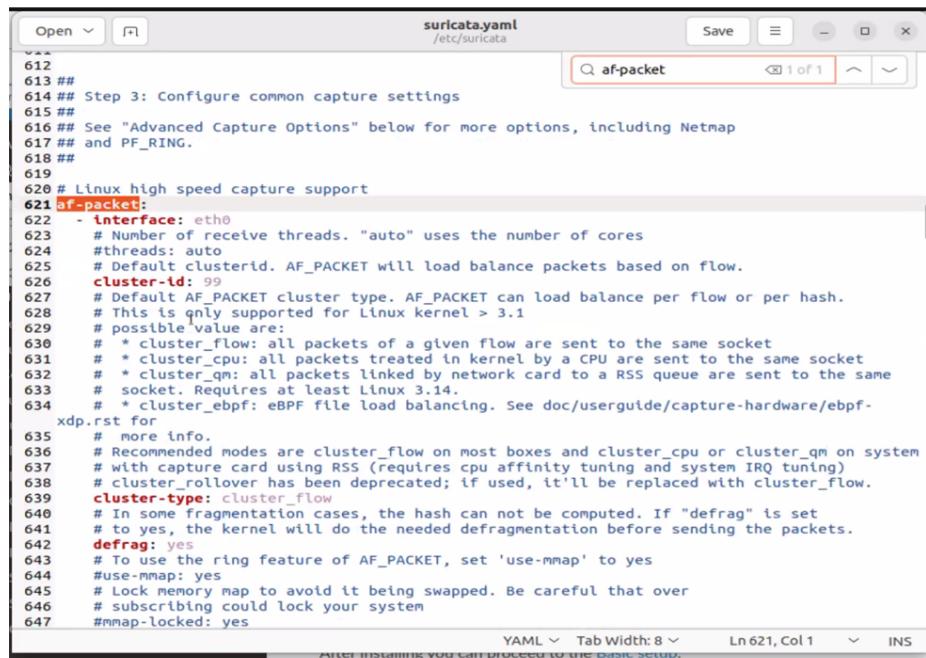
- Configure pcap section: If using ‘pcap’ capture method (instead of ‘af-packet’ or ‘nfqueue’), ensure ‘interface’ is set and ‘promisc: yes’ (promiscuous mode) is enabled.

```

1 # line 818 onwards
2 pcap:
3   - interface: enp0s8
4     # ... other options ...
5     promisc: yes
6     snaplen: 1518
7

```

Listing 15: Configuring pcap (optional)

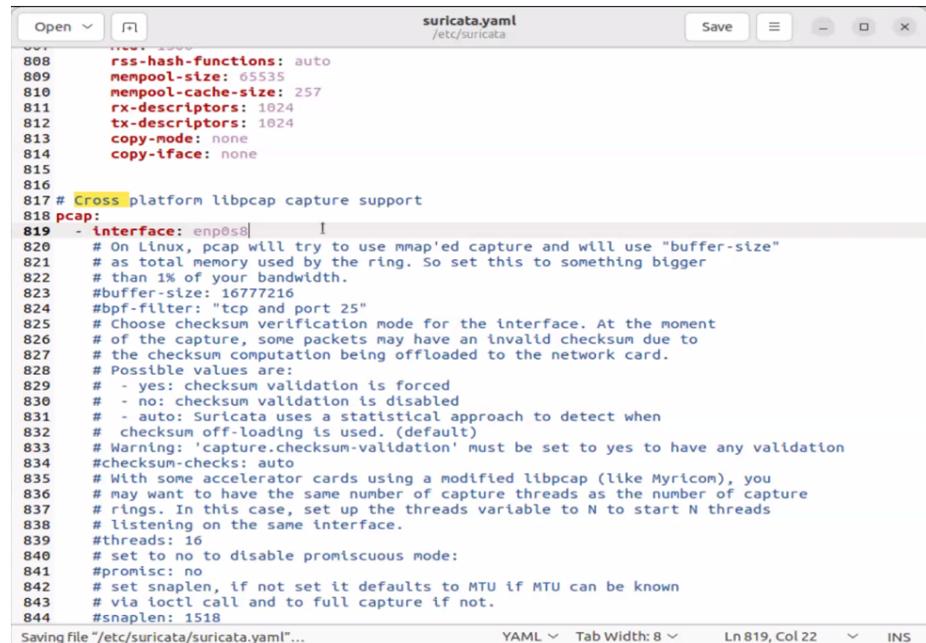


```

612
613 ## Step 3: Configure common capture settings
615 ##
616 ## See "Advanced Capture Options" below for more options, including Netmap
617 ## and PF_RING.
618 ##
619
620 # Linux high speed capture support
621 af-packet:
622   - interface: eth0
623     # Number of receive threads. "auto" uses the number of cores
624     #threads: auto
625     # Default clusterid. AF_PACKET will load balance packets based on flow.
626     cluster-id: 99
627     # Default AF_PACKET cluster type. AF_PACKET can load balance per flow or per hash.
628     # This is only supported for Linux kernel > 3.1
629     # possible values are:
630     # * cluster_flow: all packets of a given flow are sent to the same socket
631     # * cluster_cpu: all packets treated in kernel by a CPU are sent to the same socket
632     # * cluster_qm: all packets linked by network card to a RSS queue are sent to the same
633     #   socket. Requires at least Linux 3.14.
634     # * cluster_ebpf: eBPF file load balancing. See doc/ugrude/capture-hardware/ebpf-
635     xdp.rst for
636     # more info.
637     # Recommended modes are cluster_flow on most boxes and cluster_cpu or cluster_qm on system
638     # with capture card using RSS (requires cpu affinity tuning and system IRQ tuning)
639     # cluster_rollover has been deprecated; if used, it'll be replaced with cluster_flow.
639     cluster-type: cluster_flow
640     # In some fragmentation cases, the hash can not be computed. If "defrag" is set
641     # to yes, the kernel will do the needed defragmentation before sending the packets.
642     defrag: yes
643     # To use the ring feature of AF_PACKET, set 'use-mmap' to yes
644     #use-mmap: yes
645     # Lock memory map to avoid it being swapped. Be careful that over
646     # subscribing could lock your system
647     #mmap-locked: yes

```

Figure 21: Initial pcap interface.



```

807
808   rss-hash-functions: auto
809   mempool-size: 65535
810   mempool-cache-size: 257
811   rx-descriptors: 1024
812   tx-descriptors: 1024
813   copy-mode: none
814   copy-iface: none
815
816
817 # Cross platform libpcap capture support
818 pcap:
819   - interface: enp0s8
820     # On Linux, pcap will try to use mmap'ed capture and will use "buffer-size"
821     # as total memory used by the ring. So set this to something bigger
822     # than 1% of your bandwidth.
823     #buffer-size: 16777216
824     #bpf-filter: "tcp and port 25"
825     # Choose checksum verification mode for the interface. At the moment
826     # of the capture, some packets may have an invalid checksum due to
827     # the checksum computation being offloaded to the network card.
828     # Possible values are:
829     # - yes: checksum validation is forced
830     # - no: checksum validation is disabled
831     # - auto: Suricata uses a statistical approach to detect when
832     # checksum off-loading is used. (default)
833     # Warning: 'capture.checksum-validation' must be set to yes to have any validation
834     #checksum-checks: auto
835     # With some accelerator cards using a modified libpcap (like Myricom), you
836     # may want to have the same number of capture threads as the number of capture
837     # rings. In this case, set up the threads variable to N to start N threads
838     # listening on the same interface.
839     #threads: 16
840     # set to no to disable promiscuous mode:
841     #promisc: no
842     # set snaplen, if not set it defaults to MTU if MTU can be known
843     # via ioctl call and to full capture if not.
844     #snaplen: 1518

```

Figure 22: Configuring pcap interface.

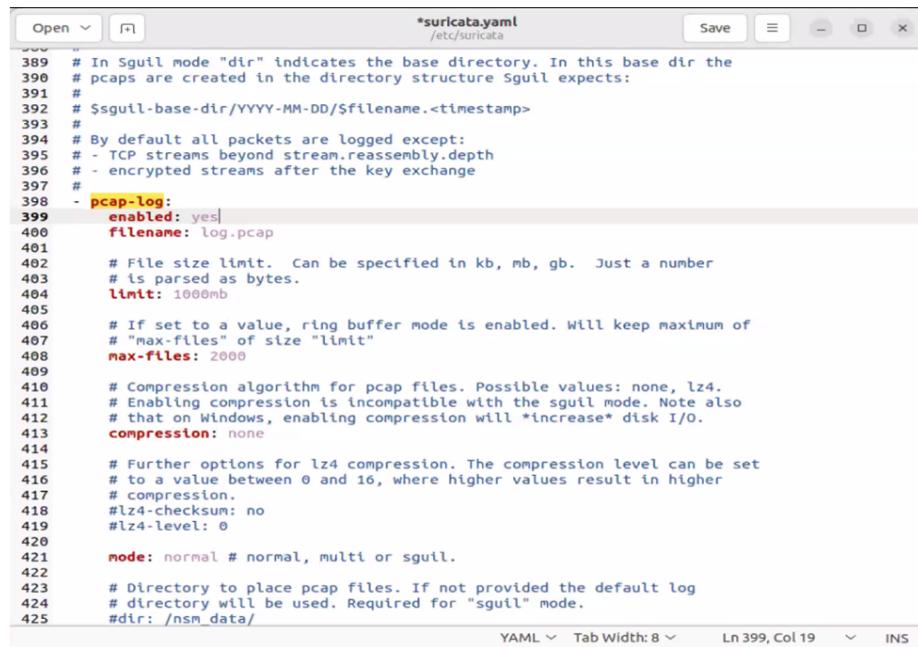
- 5. Enable pcap-log:** This feature captures all detected traffic to PCAP files, useful for later analysis in Wireshark. Set 'enabled: yes'.

```

1 # line 398 onwards
2 pcap-log:
3   enabled: yes
4   filename: log.pcap
5

```

Listing 16: Enabling pcap-log



```

389 # In Sguil mode "dir" indicates the base directory. In this base dir the
390 # pcaps are created in the directory structure Sguil expects:
391 #
392 # $sguil-base-dir/YYYY-MM-DD/$filename.<timestamp>
393 #
394 # By default all packets are logged except:
395 # - TCP streams beyond stream.reassembly.depth
396 # - encrypted streams after the key exchange
397 #
398 - pcap-log:
399   enabled: yes
400   filename: log.pcap
401
402   # File size limit. Can be specified in kb, mb, gb. Just a number
403   # is parsed as bytes.
404   limit: 1000mb
405
406   # If set to a value, ring buffer mode is enabled. Will keep maximum of
407   # "max-files" of size "limit"
408   max-files: 2000
409
410   # Compression algorithm for pcap files. Possible values: none, lz4.
411   # Enabling compression is incompatible with the sguil mode. Note also
412   # that on Windows, enabling compression will *increase* disk I/O.
413   compression: none
414
415   # Further options for lz4 compression. The compression level can be set
416   # to a value between 0 and 16, where higher values result in higher
417   # compression.
418   #lz4-checksum: no
419   #lz4-level: 0
420
421   mode: normal # normal, multi or sguil.
422
423   # Directory to place pcap files. If not provided the default log
424   # directory will be used. Required for "sguil" mode.
425   #dir: /nsm_data/

```

Figure 23: Enabling pcap-log.

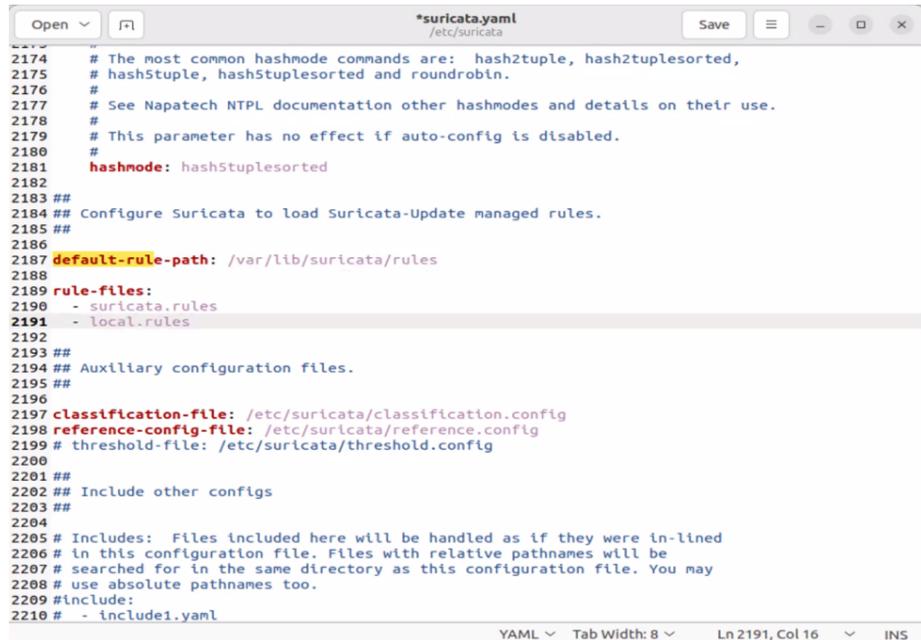
6. **Include local.rules:** Ensure that Suricata loads your custom rules. Look for the ‘rule-files’ section and add ‘local.rules’.

```

1 # line 2189 onwards
2 rule-files:
3   - suricata.rules
4   - local.rules
5

```

Listing 17: Including local.rules



```

2174   # The most common hashmode commands are: hash2tuple, hash2tuplesorted,
2175   # hashStuple, hashStuplesorted and roundrobin.
2176   #
2177   # See Napatech NTPL documentation other hashmodes and details on their use.
2178   #
2179   # This parameter has no effect if auto-config is disabled.
2180   #
2181   hashmode: hashStuplesorted
2182
2183 ##
2184 ## Configure Suricata to load Suricata-Update managed rules.
2185 ##
2186
2187 default-rule-path: /var/lib/suricata/rules
2188
2189 rule-files:
2190   - suricata.rules
2191   - local.rules
2192
2193 ##
2194 ## Auxiliary configuration files.
2195 ##
2196
2197 classification-file: /etc/suricata/classification.config
2198 reference-config-file: /etc/suricata/reference.config
2199 # threshold-file: /etc/suricata/threshold.config
2200
2201 ##
2202 ## Include other configs
2203 ##
2204
2205 # Includes: Files included here will be handled as if they were in-lined
2206 # in this configuration file. Files with relative pathnames will be
2207 # searched for in the same directory as this configuration file. You may
2208 # use absolute pathnames too.
2209 #include:
2210 #   - include1.yaml

```

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.

```
1 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.

```
1 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.

```
1 sudo systemctl start service suricata
```

Listing 20: Start Suricata

The screenshot shows a terminal window titled 'liltk@liltk-VirtualBox: ~'. The output of the command 'sudo suricata -T -c /etc/suricata/suricata.yaml -v' is displayed, showing various informational and configuration logs. It includes details about rules, engines, log files, and thresholds. Following this, the command 'sudo systemctl start suricata' is run, and the system status is checked, showing the service is active and running. The final part of the output shows the system log entries for the service starting.

```
liltk@liltk-VirtualBox: ~
at.rules: total: 59564; enabled: 43977; added: 0; removed 0; modified: 0
12/6/2025 -- 10:49:15 - <Info> -- Writing /var/lib/suricata/rules/classification.config
12/6/2025 -- 10:49:15 - <Info> -- No changes detected, exiting.
liltk@liltk-VirtualBox: ~$ sudo suricata -T -c /etc/suricata/suricata.yaml -v
Notice: suricata: This is Suricata version 7.0.10 RELEASE running in SYSTEM mode
Info: cpu: CPUs/cores online: 4
Info: suricata: Running suricata under test mode
Info: suricata: Setting engine mode to IDS mode by default
Info: exception-policy: master exception-policy set to: auto
Info: logopenfile: fast output device (regular) initialized: fast.log
Info: logopenfile: eve-log output device (regular) initialized: eve.json
Info: log-pcap: Using log dir /var/log/suricata/
Info: log-pcap: Selected pcap-log compression method: none
Info: log-pcap: Selected pcap-log conditional logging: all
Info: log-pcap: using normal logging
Info: logopenfile: stats output device (regular) initialized: stats.log
Info: detect: 2 rule files processed. 43978 rules successfully loaded, 0 rules failed, 0
Info: threshold-config: Threshold config parsed: 0 rule(s) found
Info: detect: 43981 signatures processed. 1203 are IP-only rules, 4361 are inspecting packet payload, 38195 inspect application layer, 109 are decoder event only
Notice: suricata: Configuration provided was successfully loaded. Exiting.
liltk@liltk-VirtualBox: ~$ sudo systemctl start suricata
● suricata.service - LSB: Next Generation IDS/IPS
    Loaded: loaded (/etc/init.d/suricata; generated)
    Active: active (running) since Thu 2025-06-12 10:50:22 IST; 8s ago
      Docs: man:systemd-sysv-generator(8)
   Process: 5808 ExecStart=/etc/init.d/suricata start (code=exited, status=0/S*
   Tasks: 1 (limit: 4598)
     Memory: 257.6M
        CPU: 8.286%
       CGroup: /system.slice/suricata.service
               └─5817 /usr/bin/suricata -c /etc/suricata/suricata.yaml --pidfile /var/run/suricata.pid
Jun 12 10:50:22 liltk-VirtualBox systemd[1]: Starting LSB: Next Generation IDS/IPS...
Jun 12 10:50:22 liltk-VirtualBox suricata[5808]: Likely stale PID 5213 with /var/run/suricata.pid
Jun 12 10:50:22 liltk-VirtualBox suricata[5808]: Removing stale PID file /var/run/suricata.pid
Jun 12 10:50:22 liltk-VirtualBox suricata[5808]: Starting suricata in IDS (af-packet)
Jun 12 10:50:22 liltk-VirtualBox systemd[1]: Started LSB: Next Generation IDS/IPS...
lines 1-16/16 (END)
```

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:

```
1 sudo gedit /var/lib/suricata/rules/local.rules  
2
```

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).

```
1 alert icmp 192.168.17.24 any -> $HOME_NET any (msg:"Ping from Kali";sid:  
    :10001;rev:1;)  
2 alert tcp 192.168.17.24 any -> $HOME_NET 22 (msg:"SSH Login attempt from  
    kali";sid:10002;rev:1;)  
3 alert tcp $HOME_NET any -> 57.144.156.1 443 (msg:"Facebook access attempt";  
    sid:10003;rev:1;)  
4
```

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:

```
1 sudo systemctl start suricata  
2
```

Listing 23: Start Suricata service

Run 'sudo suricata-update'

```
liltk@liltk-VirtualBox:~$ sudo suricata -T -c /etc/suricata/suricata.yaml -v
Notice: suricata: This is Suricata version 7.0.10 RELEASE running in SYSTEM mode
Info: cpu: CPUs/cores online: 4
Info: suricata: Running suricata under test mode
Info: suricata: Setting engine mode to IDS mode by default
Info: exception-policy: master exception-policy set to: auto
Info: logopenfile: fast output device (regular) initialized: fast.log
Info: logopenfile: eve-log output device (regular) initialized: eve.json
Info: log-pcap: Using log dir /var/log/suricata/
Info: log-pcap: Selected pcap-log compression method: none
Info: log-pcap: Selected pcap-log conditional logging: all
Info: log-pcap: using normal logging
Info: logopenfile: stats output device (regular) initialized: stats.log
```

```
Info: logopenfile: stats output device (regular) initialized: stats.log
Info: detect: 2 rule files processed. 43979 rules successfully loaded, 0 rules failed, 0
Info: threshold-config: Threshold config parsed: 0 rule(s) found
Info: detect: 43982 signatures processed. 1204 are IP-only rules, 4361 are inspecting packet payload, 38195 inspect application layer, 109 are decoder event only
Notice: suricata: Configuration provided was successfully loaded. Exiting.
liltk@liltk-VirtualBox:~$ sudo systemctl start suricata
```

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):

```
1 ping 192.168.17.34
2
```

Listing 24: Pinging from Kali to Ubuntu

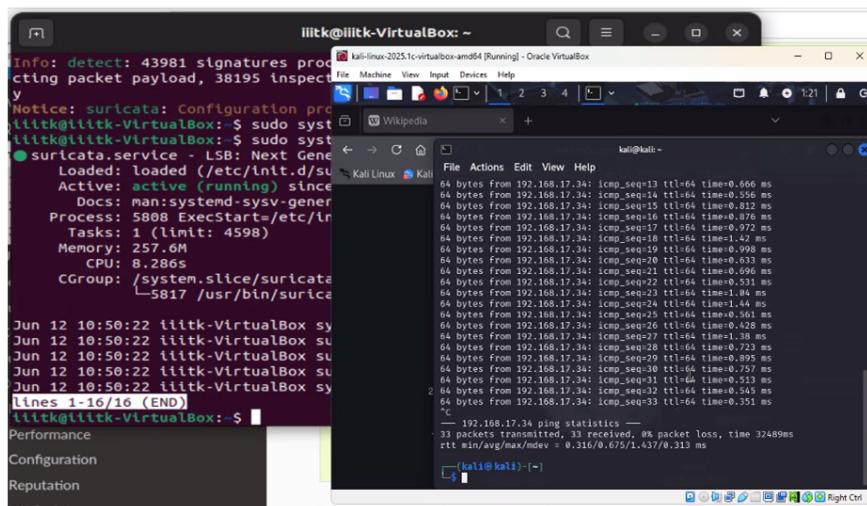


Figure 27: Ping command from Kali to Ubuntu.

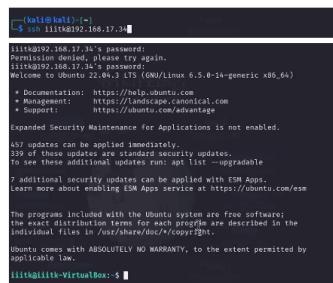
2. Attempt SSH login from Kali to Ubuntu:

```
1 sudo apt install openssh-server
2 sudo systemctl start service openssh
3
```

Listing 25: SSH initialisation in Ubuntu

```
1 ssh iiitk@192.168.17.34
2
```

Listing 26: SSH attempt from Kali to Ubuntu



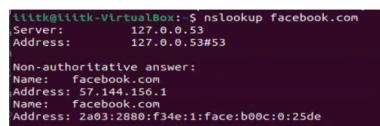
```
iiitk@kalix-kali: ~
iiitk@kalix-kali: ~$ ssh iiitk@192.168.17.34
iiitk@192.168.17.34's password:
iiitk@192.168.17.34:~$
```

Figure 28: SSH login attempt from Kali to Ubuntu.

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

```
1 nslookup facebook.com
2
```

Listing 27: DNS lookup for facebook.com on Ubuntu



```
iiitk@iiitk-VirtualBox: ~$ nslookup facebook.com
Server: 127.0.0.53
Address: 127.0.0.53#53

Non-authoritative answer:
Name: facebook.com
Address: 57.144.156.1
Name: facebook.com
Address: 2a03:2880:f34e:1:face:b00c:0:25de
```

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

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

```
1 sudo cat /var/log/suricata/fast.log  
2
```

Listing 28: View Suricata fast.log

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

```
liltk@liltk-VirtualBox:~$ sudo cat /var/log/suricata/fast.log
06/12/2025-10:50:58.290399 [**] [1:2027766:2] ET INFO Windows Update P2P Activity [**] [Classification: Not Suspicious Traffic] [Priority: 3] {TCP} 192.168.17.63:51776 -> 192.168.17.90:7680
06/12/2025-10:51:12.226224 [**] [1:10001:1] Ping from Kali [**] [Classification: (null)] [Priority: 3] {ICMP} 192.168.17.24.8 -> 192.168.17.34:0
06/12/2025-10:51:38.295719 [**] [1:2027766:2] ET INFO Windows Update P2P Activity [**] [Classification: Not Suspicious Traffic] [Priority: 3] {TCP} 192.168.17.63:51859 -> 192.168.17.90:7680

liltk@liltk-VirtualBox:~$ sudo cat /var/log/suricata/fast.log
06/12/2025-11:04:18.505366 [**] [1:2027766:2] ET INFO Windows Update P2P Activity [**] [Classification: Not Suspicious Traffic] [Priority: 3] {TCP} 192.168.17.63:53421 -> 192.168.17.90:7680
06/12/2025-11:04:58.511660 [**] [1:2027766:2] ET INFO Windows Update P2P Activity [**] [Classification: Not Suspicious Traffic] [Priority: 3] {TCP} 192.168.17.63:53503 -> 192.168.17.90:7680
06/12/2025-11:05:13.293825 [**] [1:10002:1] SSH Login attempt from kali [**] [Classification: (null)] [Priority: 3] {TCP} 192.168.17.24:55830 -> 192.168.17.34:22
06/12/2025-11:05:38.524012 [**] [1:2027766:2] ET INFO Windows Update P2P Activity [**] [Classification: Not Suspicious Traffic] [Priority: 3] {TCP} 192.168.17.90:7680 -> 192.168.17.63:53584

liltk@liltk-VirtualBox:~$ sudo cat /var/log/suricata/fast.log
06/12/2025-11:15:12.951652 [**] [1:10003:1] Facebook access attempt [**] [Classification: (null)] [Priority: 3] {TCP} 192.168.17.24:48928 -> 57.144.156.1:443
06/12/2025-11:15:13.038327 [**] [1:10003:1] Facebook access attempt [**] [Classification: (null)] [Priority: 3] {TCP} 192.168.17.24:48930 -> 57.144.156.1:443
06/12/2025-11:15:13.072599 [**] [1:10003:1] Facebook access attempt [**] [Classification: (null)] [Priority: 3] {TCP} 192.168.17.24:48932 -> 57.144.156.1:443
06/12/2025-11:15:38.672943 [**] [1:2027766:2] ET INFO Windows Update P2P Activity [**] [Classification: Not Suspicious Traffic] [Priority: 3] {TCP} 192.168.17.63:54810 -> 192.168.17.90:7680
liltk@liltk-VirtualBox:~$
```

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.

```
1 sudo wireshark /var/log/suricata/log.pcap.1749705639 # The filename changes  
with timestamp  
2
```

Listing 29: Open PCAP log in Wireshark

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

```
1 sudo truncate -s 0 /var/log/suricata/fast.log  
2 sudo truncate -s 0 /var/log/suricata/eve.json  
3 sudo rm -r /var/log/suricata/log.pcap.*  
4
```

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.

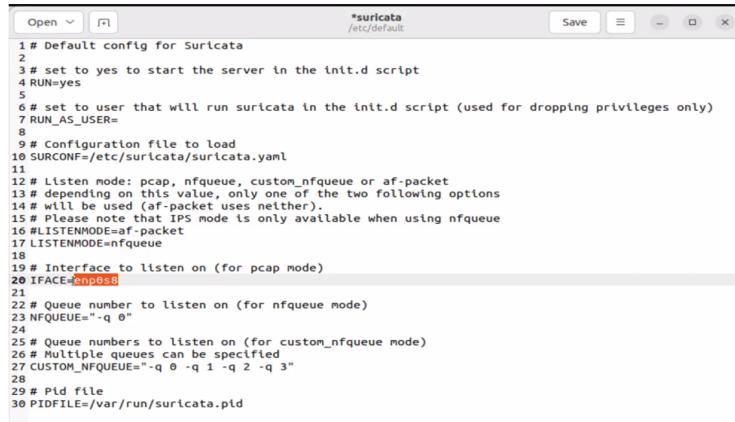
```
1 sudo gedit /etc/default/suricata
2
```

Listing 31: Open /etc/default/suricata

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

```
1 # line 4
2 RUN=yes
3
4 # line 17
5 LISTENMODE=nfqueue
6
7 # line 20
8 IFACE=enp0s8 # Use your specific interface name
9
10 # line 23
11 NFQUEUE="-q 0"
12
```

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



```
*suricata
/etc/default
Save - x
1 # Default config for Suricata
2
3 # set to yes to start the server in the init.d script
4 RUN=yes
5
6 # set to user that will run suricata in the init.d script (used for dropping privileges only)
7 RUN_AS_USER=
8
9 # Configuration file to load
10 SURCONF=/etc/suricata/suricata.yaml
11
12 # Listen mode: pcap, nfqueue, custom_nfqueue or af-packet
13 # depending on this value, only one of the two following options
14 # will be used (af-packet uses neither).
15 # Please note that IPS mode is only available when using nfqueue
16 #LISTENMODE=af-packet
17 LISTENMODE=nfqueue
18
19 # Interface to listen on (for pcap mode)
20 IFACE=enp0s8
21
22 # Queue number to listen on (for nfqueue mode)
23 NFQUEUE="-q 0"
24
25 # Queue numbers to listen on (for custom_nfqueue mode)
26 # Multiple queues can be specified
27 CUSTOM_NFQUEUE="-q 0 -q 1 -q 2 -q 3"
28
29 # Pid file
30 PIDFILE=/var/run/suricata.pid
```

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‘.

```
1 sudo gedit /etc/ufw/before.rules
2
```

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‘.

```
1 # suricata
2 -I INPUT -j NFQUEUE
3 -I OUTPUT -j NFQUEUE
4
```

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

```

1 #
2 # rules.before
3 #
4 # Rules that should be run before the ufw command line added rules. Custom
5 # rules should be added to one of these chains:
6 #   ufw-before-input
7 #   ufw-before-output
8 #   ufw-before-forward
9 #
10 # Don't delete these required lines, otherwise there will be errors
11 *filter
12 :ufw-before-input - [0:0]
13 :ufw-before-output - [0:0]
14 :ufw-before-forward - [0:0]
15 :ufw-not-local [0:0]
16 :ufw-not-local [0:0]
17 # End required lines
18
19 #suricata
20 -I INPUT -j NFQUEUE
21 -I OUTPUT -j NFQUEUE
22
23 # allow all on loopback
24 -A ufw-before-input -i lo -j ACCEPT
25 -A ufw-before-output -o lo -j ACCEPT
26
27 # quickly process packets for which we already have a connection
28 -A ufw-before-input -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
29 -A ufw-before-output -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
30 -A ufw-before-forward -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
31
32 # drop INVALID packets (logs these in loglevel medium and higher)
33 -A ufw-before-input -m conntrack --ctstate INVALID -j ufw-logging-deny
34 -A ufw-before-input -m conntrack --ctstate INVALID -j DROP
35
36 # ok icmp codes for INPUT
37 -A ufw-before-input -p icmp --icmp-type destination-unreachable -j ACCEPT

```

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

3. Enable UFW:

```

1 sudo ufw enable
2

```

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:

```

1 sudo gedit /var/lib/suricata/rules/local.rules
2

```

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.

```

1 drop icmp 192.168.17.24 any -> $HOME_NET any (msg:"Ping from Kali";sid
    :10001;rev:1;)
2 drop tcp 192.168.17.24 any -> $HOME_NET 22 (msg:"SSH Login attempt from
    kali";sid:10002;rev:1;)
3 drop tcp $HOME_NET any -> 57.144.156.1 443 (msg:"Facebook access attempt";
    sid:10003;rev:1;)
4 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;)
5

```

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.

```
1 sudo suricata -update  
2
```

Listing 38: Update Suricata rules after modification

5. Test Suricata configuration again:

```
1 sudo suricata -T -c /etc/suricata/suricata.yaml -v  
2
```

Listing 39: Test Suricata configuration

9.3 Testing IPS Functionality

1. Clear Suricata logs and PCAPs:

```
1 sudo truncate -s 0 /var/log/suricata/fast.log  
2 sudo truncate -s 0 /var/log/suricata/eve.json  
3 sudo rm -r /var/log/suricata/log.pcap.*  
4
```

Listing 40: Clear Suricata logs and PCAPs

2. Stop Suricata, then Start it again:

```
1 sudo systemctl stop suricata  
2 sudo systemctl start suricata  
3
```

Listing 41: Restart Suricata for IPS mode

3. Attempt to Ping from Kali to Ubuntu:

```
1 ping 192.168.17.34  
2
```

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.

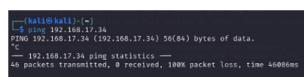


Figure 33: Ping from Kali to Ubuntu, showing 100% packet loss due to IPS blocking.

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

```
2
```

Listing 43: View fast.log for ICMP drop alert

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

```
iitk@iitk-VirtualBox: $ sudo cat /var/log/suricata/fast.log
06/12/2025-11:26:38.824397 [**] [1:2027766:2] ET INFO Windows Update P2P Activity [**] [Classification: Not Suspicious Traffic] [Priority: 3] {TCP} 192.168.17.63:56031 -> 192.168.17.90:7680
06/12/2025-11:26:18.831007 [**] [1:2027766:2] ET INFO Windows Update P2P Activity [**] [Classification: Not Suspicious Traffic] [Priority: 3] {TCP} 192.168.17.63:56118 -> 192.168.17.90:7680
06/12/2025-11:26:29.533564 [**] [1:10004:1] Potential SYN Flood [**] [Classification: (null)] [Priority: 3] {TCP} 10.0.0.9:1949 -> 192.168.17.34:80
06/12/2025-11:26:41.547250 [**] [1:10004:1] Potential SYN Flood [**] [Classification: (null)] [Priority: 3] {TCP} 10.0.0.9:1961 -> 192.168.17.34:80
06/12/2025-11:26:53.559773 [**] [1:10004:1] Potential SYN Flood [**] [Classification: (null)] [Priority: 3] {TCP} 10.0.0.9:1973 -> 192.168.17.34:80
06/12/2025-11:26:58.840703 [**] [1:2027766:2] ET INFO Windows Update P2P Activity [**] [Classification: Not Suspicious Traffic] [Priority: 3] {TCP} 192.168.17.63:56199 -> 192.168.17.90:7680
06/12/2025-11:36:47.813270 [Drop] [**] [1:10001:1] Ping from Kali [**] [Classification: (null)] [Priority: 3] {ICMP} 192.168.17.24:8 -> 192.168.17.34:0
```

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

4. Attempt SSH login from Kali to Ubuntu:

```
1 ssh iitk@192.168.17.34
```

```
2
```

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.

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

```
2
```

Listing 45: View fast.log for SSH drop alert

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

```
littk@littk-VirtualBox: ~ $ sudo cat /var/log/suricata/fast.log
06/12/2025-11:40:39.376913 [Drop] [**] [1:10002:1] SSH Login attempt from kali
[**] [Classification: (null)] [Priority: 3] {TCP} 192.168.17.24:35884 -> 192.168
.,17.34:22
```

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.

```
1 sudo hping3 -S -a 10.0.0.9 -p 80 192.168.17.34
2
```

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.

```
littk@littk-VirtualBox: ~ $ sudo hping3 -S -a 10.0.0.9 -p 80 192.168.17.34
sudo: password for kali:
[eth0] 192.168.17.34 (eth0 192.168.17.34): 5 set, a0 headers + 0 data bytes
-- 192.168.17.34 hping statistic
0 packets transmitted, 0 packets received, 100% packet loss
round-trip std/avg/max = 0.0/0.0/0.0 ms
```

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

```
1 sudo cat /var/log/suricata/fast.log
2
```

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.