VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY

UNIVERSITY OF INFORMATION TECHNOLOGY

FINAL REPORT

Intrusion Detection using Snort with DVWA

Nguyễn Đức Minh Quân 24560015

Lecturer: MSc. Trần Thị Dung

HO CHI MINH CITY

Table of Contents

Table of Figures

Table of Tables

List of Abbreviations

Abbreviation	Full Term
DHCP	Dynamic Host Configuration Protocol
DVWA	Damn Vulnerable Web Application
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
NAT	Network Address Translation
SIEM	Security Information and Event Management
SQL	Structured Query Language
SSH	Secure Shell
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VM	Virtual Machine

1. Introduction

1.1 Definition

Snort is a network-based intrusion detection system written in the C programming language, originally developed in 1998 by Martin Roesch and now maintained by Cisco. It is free, open-source software that can also function as a real-time packet sniffer, enabling network administrators to monitor traffic and identify packets that may pose a threat to the system. Snort’s rule-based structure allows for straightforward creation, customization, and deployment of detection rules, making it highly adaptable to diverse operating systems and network environments.

Although primarily used as an IDS—a system that monitors and analyzes network traffic to detect suspicious activity or policy violations—Snort distinguishes itself through its flexibility, modular design, and strong community support. Unlike many proprietary IDS solutions, Snort’s open-source nature allows users to inspect, modify, and extend its functionality according to specific organizational needs. In IDS mode, it performs real-time packet inspection, protocol decoding, content searching, and alert generation. Additionally, it supports comprehensive logging and integrates seamlessly with security tools such as SIEM platforms for correlation, reporting, and automated response.

Snort’s combination of accessibility, extensibility, and effective detection capabilities has made it one of the most widely used network intrusion detection solutions worldwide. Its popularity stems not only from being free and open-source but also from the ability to rapidly develop and share detection rules within a global community, providing organizations with timely updates against emerging threats and ensuring robust, continuous network monitoring.

1.2 Components

To implement a Snort-based IDS, several components are necessary:

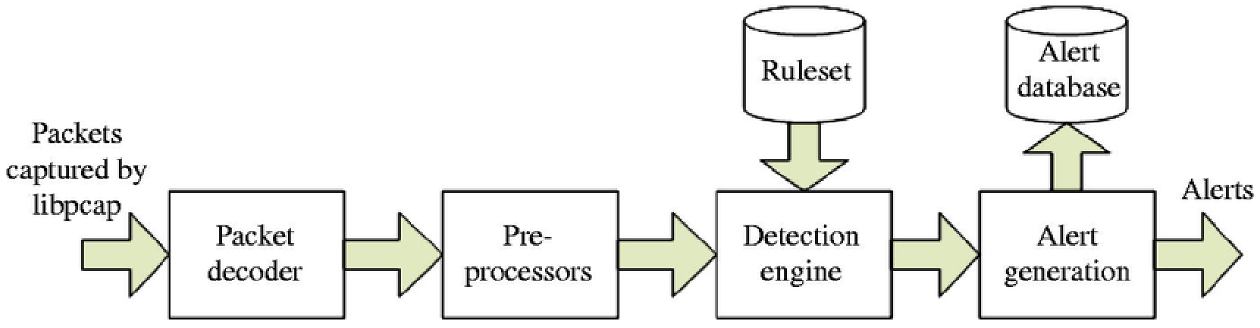


Figure 1: Snort Architecture (Source: Snort Manual)

Sniffer and Packet Decoder

The initial stage of Snort combines the sniffer and packet decoder to efficiently handle incoming network traffic. The sniffer continuously monitors network interfaces and captures all packets in real time. The decoder then interprets these packets by parsing protocol headers across Ethernet, IP, TCP/UDP, and application layers, extracting key information such as addresses, ports, flags, and payloads. By transforming raw packets into structured, analyzable data, this stage establishes the foundation for the subsequent stages of analysis, ensuring accuracy and completeness for further processing.

Preprocessing

Following decoding, the preprocessing stage prepares the traffic for detection. Preprocessors normalize packet content, reassemble fragmented streams, and detect preliminary anomalies such as TCP irregularities, port scans, or protocol violations. Content normalization also mitigates evasion techniques, including unusual encoding or fragmentation. This stage acts as a filter and standardizer, delivering clean, organized, and reliable traffic to the detection engine and reducing the likelihood of false positives.

Detection Engine

The detection engine is the central component of Snort, responsible for the actual identification of network threats. It performs **signature-based analysis**, which searches packet payloads for known attack patterns, and **rule-based analysis**, which evaluates packets against conditions defined in user-configurable rules, including protocol behavior, packet fields, content patterns, and temporal thresholds. Advanced features, such as thresholding, event correlation, and suppression, allow it to detect complex or repeated attack scenarios while prioritizing relevant alerts. The effectiveness of the detection engine relies on accurate input from preprocessing, making it the core analytical stage that transforms raw and preprocessed traffic into actionable security insights.

Alerts and Logging

Once the detection engine identifies suspicious activity, the alerts and logging component generates notifications and records comprehensive details of each event. Logs include timestamps, packet attributes, source and destination addresses, and triggered rules. These records support forensic analysis, incident response, compliance auditing, and integration with external monitoring platforms such as SIEM systems, ensuring that administrators have actionable information for timely mitigation of threats.

1.3 Operation

1.3.1 Snort Architecture and Workflow

Snort’s architecture consists of multiple key modules working in sequence to capture, process, analyze, and respond to network traffic:

he operation of Snort can be viewed as a continuous, modular workflow, where each stage builds upon the previous one to provide real-time threat detection. Initially, the combined *sniffer and packet decoder stage* captures all network traffic in real time and interprets it into structured, analyzable data. The sniffer monitors network interfaces without interruption, while the decoder parses protocol headers across Ethernet, IP, TCP/UDP, and application layers, extracting essential details such as addresses, ports, flags, and payloads. This produces protocol-compliant traffic, which ensures accuracy and reliability for subsequent analysis.

Next, the *preprocessing stage* refines this decoded traffic. Packets are normalized, fragmented streams are reassembled, and protocol compliance is verified. Preprocessors also detect early anomalies, such as irregular TCP flags, port scans, or unusual sequence numbers, while mitigating evasion attempts like atypical encoding. By delivering clean, standardized, and compliant traffic, preprocessing ensures the **detection engine** receives high-quality input, minimizing false positives and enhancing threat detection.

As the core analytical component, the *detection engine* evaluates preprocessed traffic using both signature-based and rule-based methods. Signature-based analysis identifies known attack patterns, while rule-based inspection evaluates protocol fields, payload content, temporal thresholds, and behavioral anomalies. Thresholding and event correlation allow complex or repeated attacks to be detected accurately.

Finally, the *alerts and logging component* records validated events with detailed metadata, including timestamps, packet details, source and destination addresses, triggered rules, and protocol information. In real-world scenarios, these logs serve as an indispensable resource for forensic investigations, incident response, compliance audits, and integration with SIEM systems. By maintaining structured, retrievable information, administrators can review historical events, correlate multiple alerts, and make informed decisions to prevent or mitigate future attacks. This end-to-end workflow ensures Snort operates as a robust, real-time, and actionable intrusion detection system.

1.3.2. Rules Configuration

a. Define Network Context and Variables

Before writing or customizing rules, it is essential to define the network environment through variables in the Snort configuration. Common variables include:

- HOME_NET: Represents the protected internal network(s). Rules typically monitor traffic destined to or originating from this network.
- EXTERNAL_NET: Defines networks considered external or untrusted.
- OBJECT_PORTS: Frequently used port ranges or services.

Accurate definition of these variables enables rules to be written generically and applied flexibly across different network segments, reducing configuration complexity.

b. Understand the Attack Patterns and Protocols

When creating rules, a thorough understanding of the attack signatures, protocol behaviors, and potential evasion techniques is vital. This includes knowledge of:

- Protocol specifics (TCP flags, ICMP types, HTTP methods)
- Typical payload patterns of attacks (e.g., SQL injection keywords, SSH login attempts)
- Behavior thresholds that distinguish normal traffic from malicious activity

This comprehension ensures rules target meaningful indicators rather than overly broad or irrelevant traffic.

c. Craft Precise Rule Headers and Options

Rules should be as specific as possible to minimize false positives. Considerations include:

- Protocol and Port Specification: Narrow rules to specific protocols (TCP, UDP, ICMP) and relevant ports.
- IP Address Ranges: Use defined network variables instead of broad any statements when possible.
- Content Matching: Specify unique strings or byte patterns found only in malicious payloads.
- Flow and Flags: Utilize flow direction (to_server, to_client) and TCP flags (SYN, ACK) to refine detection.
- Thresholding: Apply detection filters or thresholds to reduce alert noise from benign high-frequency events.

d. Incorporate Metadata and Documentation

Each rule should include:

- Message (msg) Field: A clear, descriptive alert message explaining the detected condition.
- SID (Snort ID): A unique identifier for tracking and management.
- Revision Number: For version control of rule updates.
- Reference Links: Optionally, include references to CVEs, advisories, or documentation for context.

Well-documented rules facilitate maintenance, collaboration, and incident response. More accurate the rules will lead to more accurate alerts.

1.4 Attacks Simulated

The evaluation of the intrusion detection system involves simulating a series of common network and application-layer attacks to assess its detection capabilities. These attacks, widely recognized in cybersecurity, are designed to probe, exploit, or disrupt network resources, each with distinct characteristics and objectives. Below is a brief overview of the four attacks simulated, highlighting their purpose and typical behavior in a general network environment.

ICMP Flood

An ICMP Flood attack consists of overwhelming a target system by sending a large volume of ICMP Echo Request packets, commonly known as ping requests. The objective is to saturate the network bandwidth or consume processing resources of the target device, leading to degradation or denial of legitimate network services. This attack exploits the simplicity and low overhead of ICMP traffic to rapidly exhaust system resources, potentially causing network unavailability.

HPING3 SYN Flood

The HPING3 SYN Flood is a type of denial-of-service attack targeting the TCP three-way handshake mechanism. The attacker sends a continuous stream of TCP packets with the SYN flag set to the target system without completing the handshake process. This results in many half-open connections, exhausting the connection table of the target and preventing legitimate users from establishing new TCP sessions. This attack specifically disrupts services relying on TCP, such as SSH or web servers.

SQL Injection

SQL Injection is a web application attack technique wherein an attacker inserts malicious SQL code into input fields or URL parameters. This malicious code is then executed by the backend database, allowing unauthorized data access, modification, or deletion. Successful SQL injection can lead to data breaches, unauthorized administrative access, and complete compromise of the affected web application.

SSH

This is not an attack, a normal SSH connection attempt, while benign, can be monitored to detect unauthorized or suspicious access patterns. Monitoring connection attempts on forwarded ports can help identify reconnaissance or preliminary stages of brute force attacks. Differentiating normal connection activity from attack behavior is essential in maintaining network security.

2. Implementation

2.1 Topology

Basic Objective: To identify and detect internal attacks to local network:

- ICMP Flood
- SYN Flood

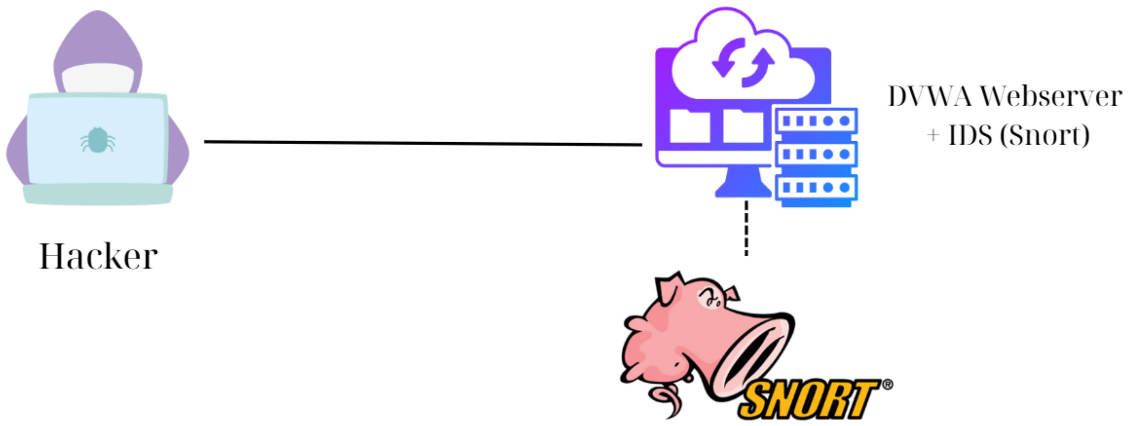


Figure 2: Basic Topology Diagram

VM Name	Role	IP
kali-web	Web Server + Snort	192.168.88.132
kali-attacker	Attacker	192.168.88.129

Table 1: Basic Part Components

- Both machines are on the same internal network.
- DVWA is hosted on kali-web.

Advanced Objective: Put Snort on a Router, separate from the client to visualize real life scenario which identify and detect more complex attacks on the application level from external sources before coming to the internal networks:

- ICMP Flood, SYN Flood
- SSH
- SQL injection

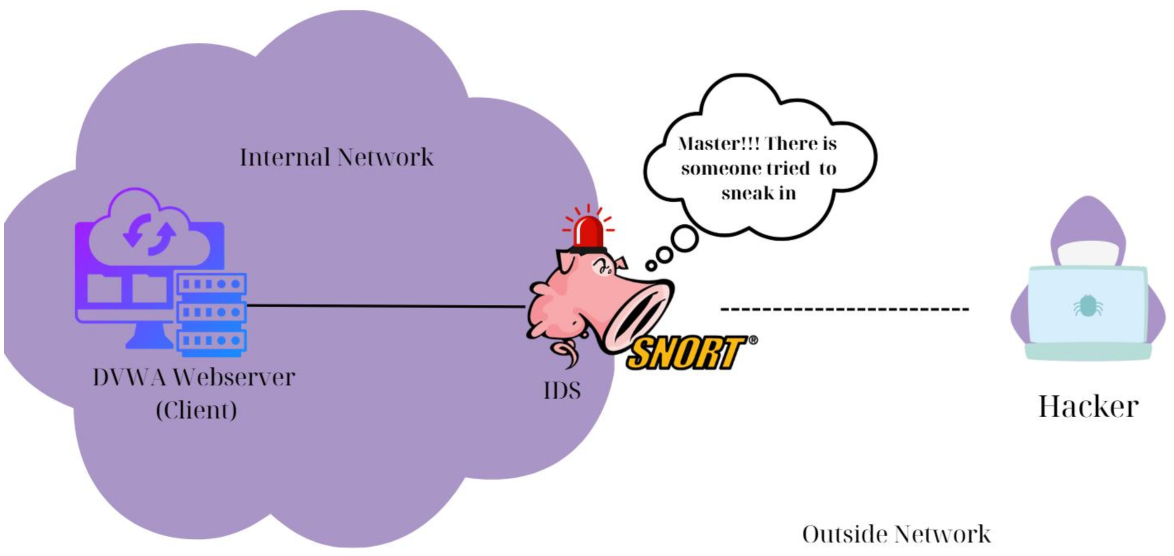


Figure 3: Advanced Topology Diagram

VM Name	Role	NICs	IP
kali-router	Router + Snort	1 NAT + 1 Host only	192.168.100.1
kali-victim	Target Web Server	Host only	192.168.100.10
kali-attacker	Attacker	Any configuration	192.168.56.20

Table 2: Advanced Part Components

- kali-router performs NAT, port forwarding, and runs Snort.
- kali-attacker accesses kali-victim through forwarded ports.

2.2 Installation

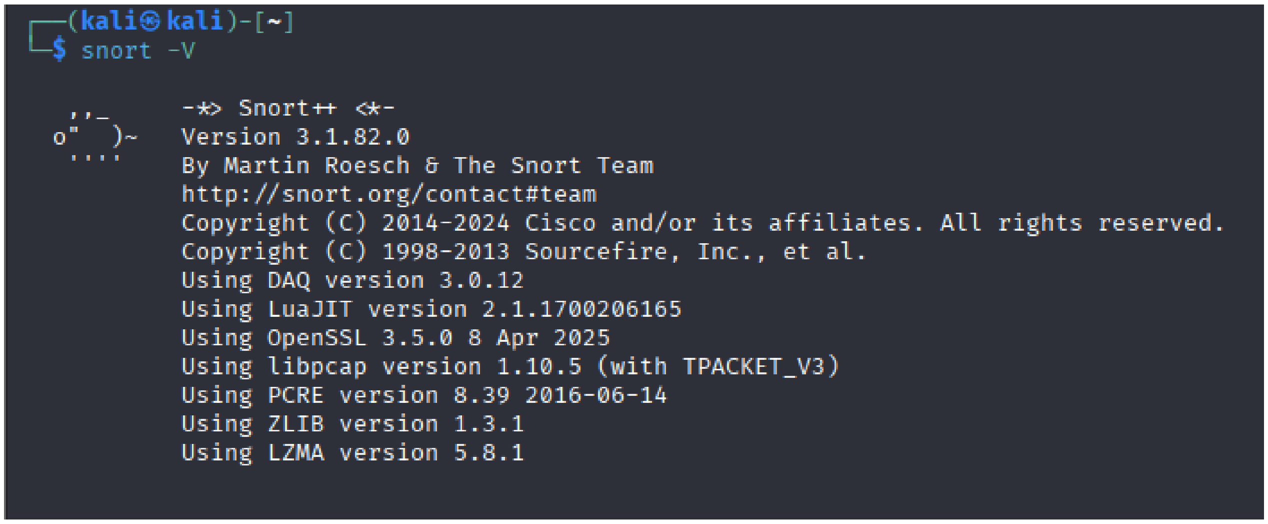
Both basic and advanced part use my DVWA webserver that I created on previous lab, so I just show the status of the web instead of showing how I created it.

2.2.1. Snort Installation

On kali-web and kali-router use this command to install Snort:

```
sudo apt update
sudo apt install snort -y
```

After that, use snort -V to check the status of Snort installation and also the version of Snort.



```
┌──(kali㉿kali)-[~]
└─$ snort -V

   ,,_     -*> Snort++ <*-
  o"  )~   Version 3.1.82.0
   ''''    By Martin Roesch & The Snort Team
           http://snort.org/contact#team
           Copyright (C) 2014-2024 Cisco and/or its affiliates. All rights reserved.
           Copyright (C) 1998-2013 Sourcefire, Inc., et al.
           Using DAQ version 3.0.12
           Using LuaJIT version 2.1.1700206165
           Using OpenSSL 3.5.0 8 Apr 2025
           Using libpcap version 1.10.5 (with TPACKET_V3)
           Using PCRE version 8.39 2016-06-14
           Using ZLIB version 1.3.1
           Using LZMA version 5.8.1
```

Figure 4: Snort Status

Then make directories for snort rules – where you put your own rules and prebuilt rules from the community and snort logs – for storing events and can parse it to monitoring tools like SIEM like a black box which recorded everything:

```
sudo mkdr -p /etc/snort/rules
sudo mkdr -p /var/log/snort
```

2.2.2. Attacker Tool Installation

As aforementioned in section 1.4 and section 2.1, to perform SYN flood, ICMP flood, SSH, and SQL injection attacks, although some are prebuilt in Kali Linux (SSH, ping), installing dedicated libraries is a must:

Command

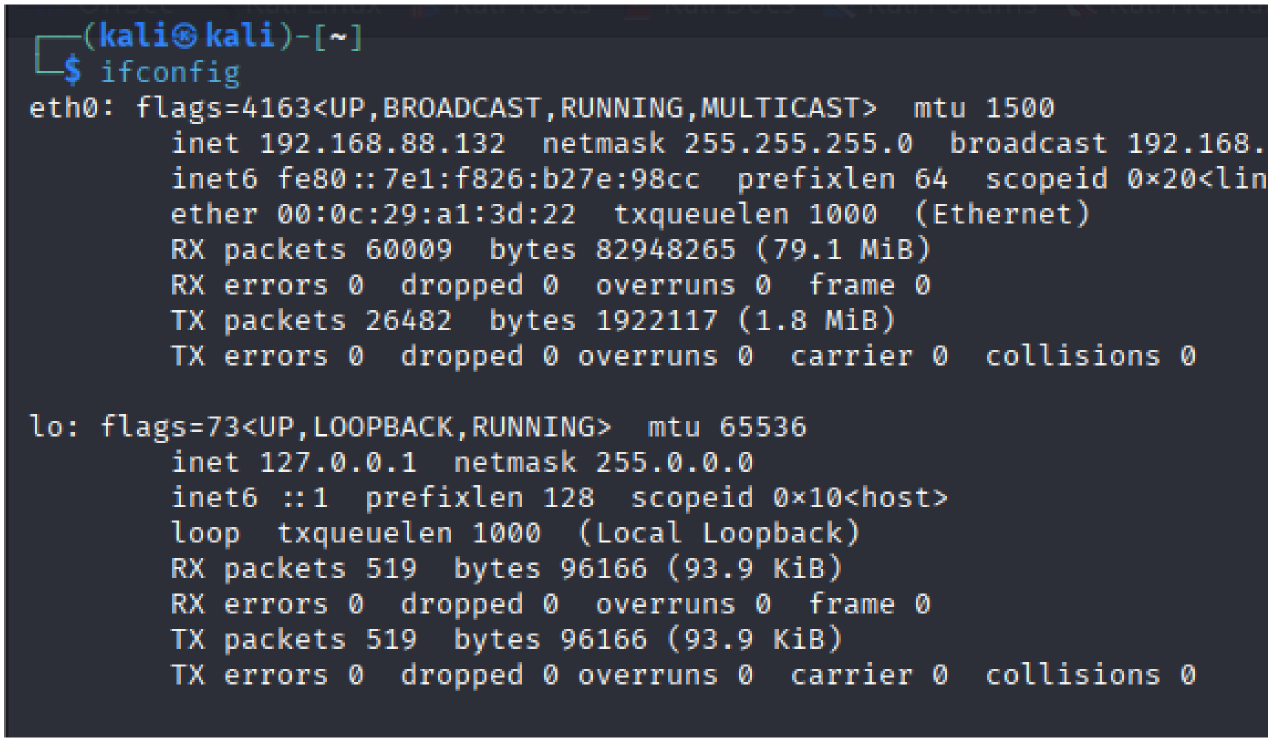
```
sudo apt install hping3 curl -y
```

2.3. Configuration

The configuration of Snort follows a structured workflow in which administrators first define detection rules based on specific security objectives, then integrate these rules into the Snort configuration files, and finally activate the system to monitor network traffic in real time; while both basic and advanced deployments share the same foundational configuration—including network definitions, rule inclusion, and preprocessing settings—the advanced setup requires additional configuration steps tailored to each objective, such as thresholding, event correlation, or protocol-specific tuning, enabling more sophisticated detection and alerting capabilities beyond the standard baseline. For clarity and better organization, detection rules for each type of attack are designed separately and presented in a dedicated section, allowing focused analysis and easier management of both basic and advanced detection scenarios.

2.3.1. Basic:

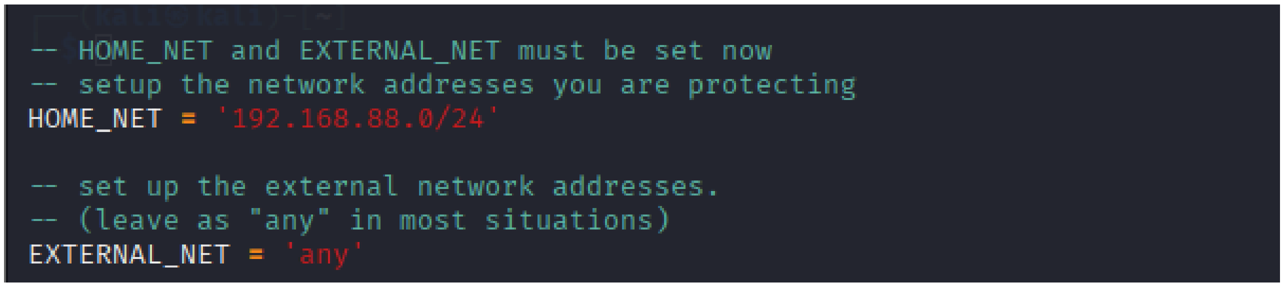
Before creating detection rules, Snort must be configured to define the network it will monitor. In the *snort.lua* - configuration file, the *HOME_NET* variable specifies the internal network that Snort will protect, while *EXTERNAL_NET* represents traffic originating from outside this network. Since the web server is hosted on the same VM, the *HOME_NET* is set to the subnet 192.168.88.0/24. This ensures that all internal hosts and services within this subnet are monitored as shown in the figures below.



```
┌──(kali㉿kali)-[~]
└─$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.88.132  netmask 255.255.255.0  broadcast 192.168.
        inet6 fe80::7e1:f826:b27e:98cc  prefixlen 64  scopeid 0x20<lin
        ether 00:0c:29:a1:3d:22  txqueuelen 1000  (Ethernet)
        RX packets 60009  bytes 82948265 (79.1 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 26482  bytes 1922117 (1.8 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 519  bytes 96166 (93.9 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 519  bytes 96166 (93.9 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 5: IP check on Client VM



```
-- HOME_NET and EXTERNAL_NET must be set now
-- setup the network addresses you are protecting
HOME_NET = '192.168.88.0/24'

-- set up the external network addresses.
-- (leave as "any" in most situations)
EXTERNAL_NET = 'any'
```

Figure 6: Basic Network configuration

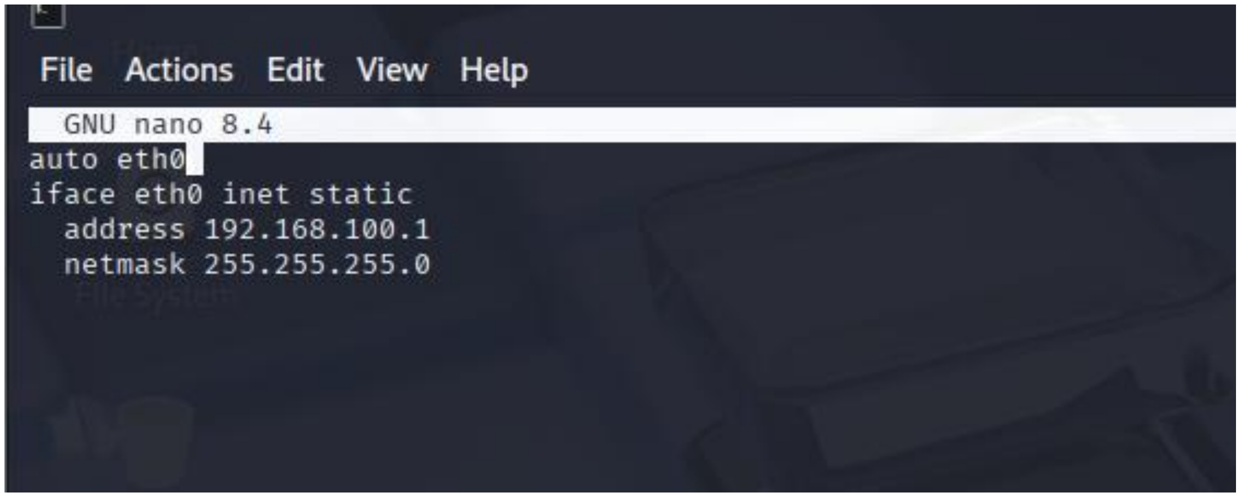
2.3.2. Advanced:

a. Configuring DHCP Server:

The first step involves configuring the Kali VM to act as a DHCP server. The network interfaces file is edited using:

```
sudo nano /etc/network/interfaces
```

The following configuration is added to assign a static IP to the Kali router with the IP of 192.168.100.1/24



```
auto eth0
iface eth0 inet static
  address 192.168.100.1
  netmask 255.255.255.0
```

Figure 7: Assigning IP for Kali router

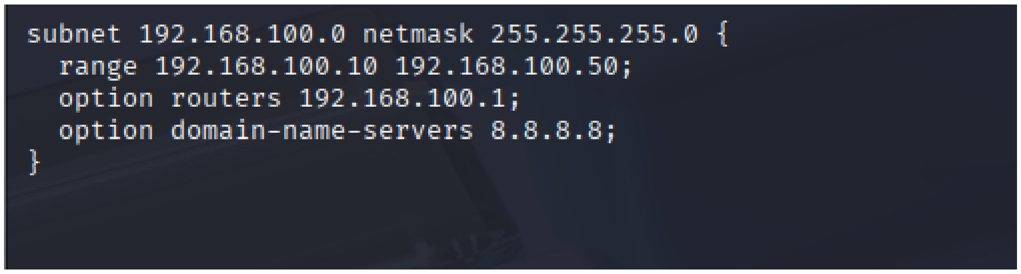
Then need to restart networking to save all the configurations.

```
sudo systemctl restart networking
```

Next, the DHCP server package is installed:

```
sudo apt update && sudo apt install isc-dhcp-server
```

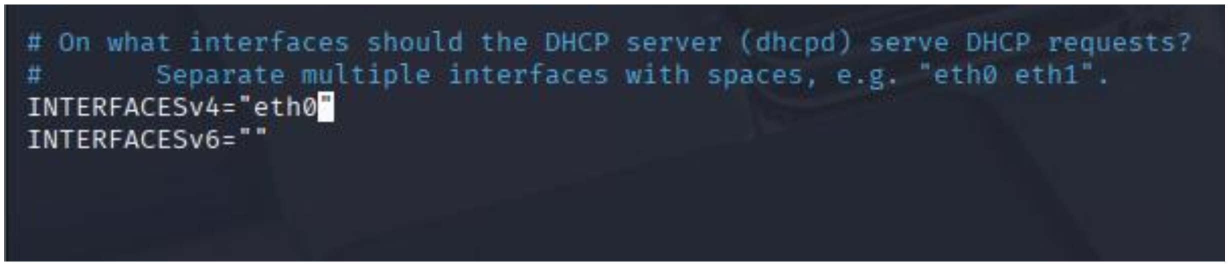
The DHCP configuration file - */etc/dhcp/dhcpd.conf* is edited to create an IP pool for clients connected to VMNet2, automatically assigning IP addresses while defining the Kali router as the default gateway and specifying a DNS server.



```
subnet 192.168.100.0 netmask 255.255.255.0 {
  range 192.168.100.10 192.168.100.50;
  option routers 192.168.100.1;
  option domain-name-servers 8.8.8.8;
}
```

Figure 8: Adding DHCP Pool

After that go to /etc/default/isc-dhcp-server, set the Router’s internal interface,t his will use the eth0 interface as a router interface directly connected to LAN.



```
# On what interfaces should the DHCP server (dhcpd) serve DHCP requests?
#       Separate multiple interfaces with spaces, e.g. "eth0 eth1".
INTERFACESv4="eth0"
INTERFACESv6=""
```

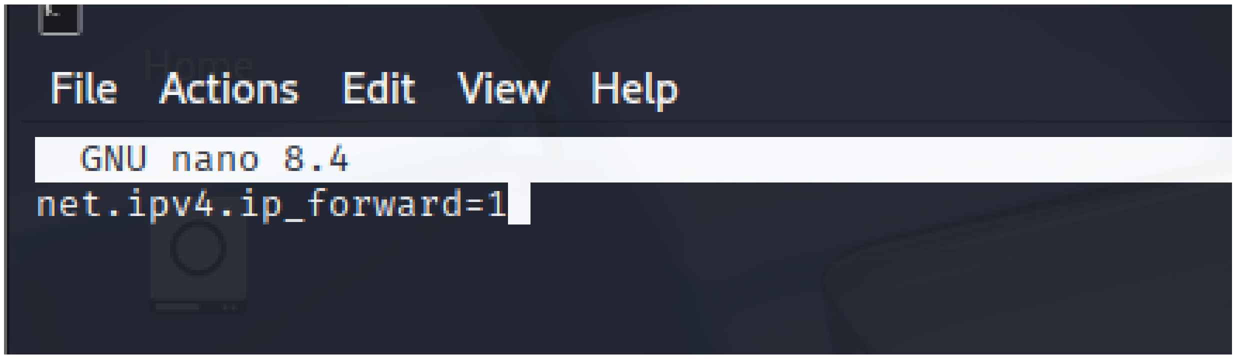
Figure 9: LAN Interface setting

After all restart service:

```
sudo systemctl restart isc-dhcp-server
```

b. Configuring IP Forwarding, NAT and Port Forwarding via iptables

To enable the Kali router to forward packets between interfaces, IP forwarding is enabled by editing /etc/sysctl.conf file, uncomment:

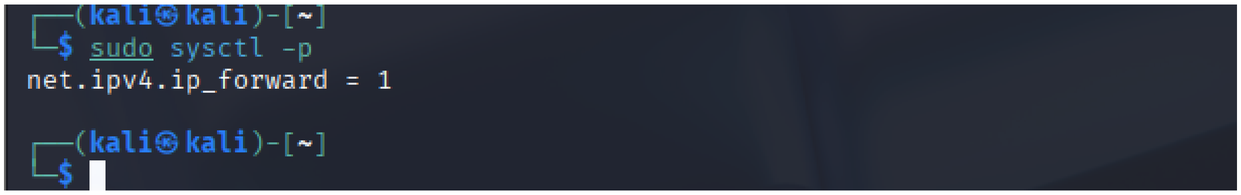


```
GNU nano 8.4
net.ipv4.ip_forward=1
```

Figure 10: Forward packets setting

After applying, this configuration allows packet forwarding and lays the foundation for implementing NAT and port forwarding rules via iptables, ensuring proper routing and monitoring of traffic between the LAN and external networks.

```
sudo sysctl -p
```



```
┌──(kali㉿kali)-[~]
└─$ sudo sysctl -p
net.ipv4.ip_forward = 1

┌──(kali㉿kali)-[~]
└─$
```

Figure 11: Saving forwarding packets setting

c. Configure Iptables rules for all the task:

Based on the objectives of the advanced deployment, iptables rules were configured to allow external attackers to reach the internal web server (on ports 80 for HTTP and 22 for SSH), permit ICMP traffic, and provide NAT functionality so that devices within the LAN can access external networks. The rules are organized to ensure both security and proper traffic forwarding.

- **Allow loopback, established and related connections**

```
sudo iptables -A INPUT -i lo -j ACCEPT
sudo iptables -A OUTPUT -o lo -j ACCEPT

sudo iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
sudo iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j
ACCEPT
```

The first two rules explicitly permit traffic on the loopback interface (lo), which is essential for local inter-process communication, such as Snort logging and monitoring. The next two rules utilize connection tracking to allow packets belonging to established or related connections to bypass explicit port rules. This ensures that legitimate return traffic is not dropped, improving both performance and security.

- **NAT for Internal-to-External Traffic**

```
sudo iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
sudo iptables -A FORWARD -i eth0 -o eth1 -j ACCEPT
```

These rules configure NAT so that outbound traffic from the internal network (eth0) to the NAT network (eth1) is masqueraded. This enables internal devices to share the NAT IP for internet access while concealing their private IP addresses. Forwarding rules allow packets to traverse between internal and external interfaces seamlessly.

- **SSH port forwarding**

```
sudo iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 22 -j DNAT -
-to-destination 192.168.100.10:22
sudo iptables -A FORWARD -p tcp -d 192.168.100.10 --dport 22 -j ACCEPT
```

These rules forward SSH traffic from the external network to the internal web server (192.168.100.10). By mapping TCP port 22 on the NAT interface to the internal host, remote administration is enabled securely.

- **HTTP port forwarding**

```
sudo iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 80 -j DNAT -
-to-destination 192.168.100.10:80
sudo iptables -A FORWARD -p tcp -d 192.168.100.10 --dport 80 -j ACCEPT
```

Similarly, HTTP traffic on TCP port 80 is forwarded to the internal web server, allowing external users to access web services hosted on the LAN.

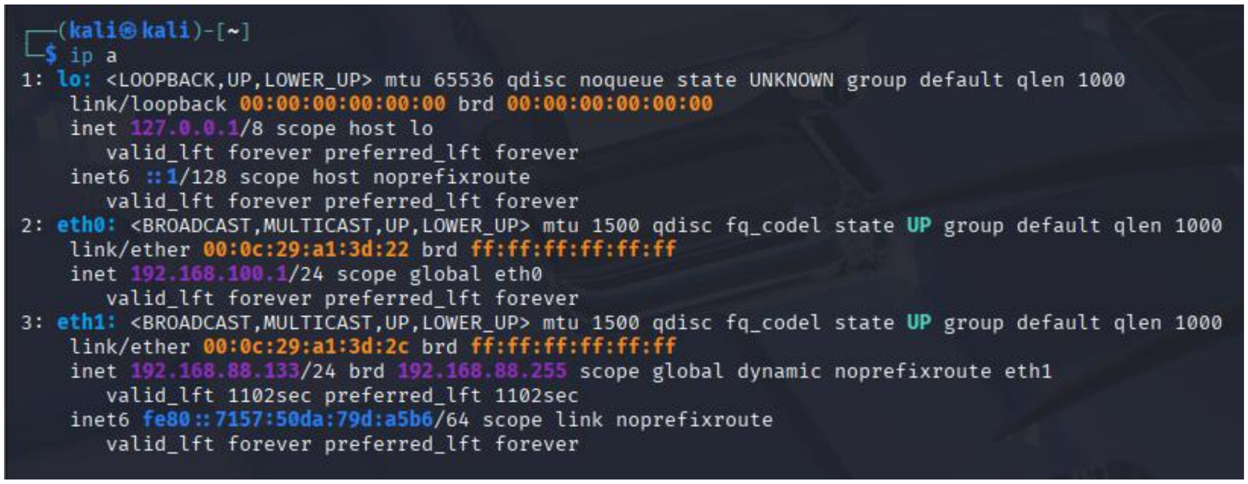
- **ICMP forwarding**

```
sudo iptables -A FORWARD -p icmp -j ACCEPT
sudo iptables -A INPUT -p icmp -j ACCEPT
```

ICMP traffic is explicitly permitted to allow ping requests and network diagnostics from external hosts. This also enables testing of ICMP flood attacks for monitoring and detection purposes.

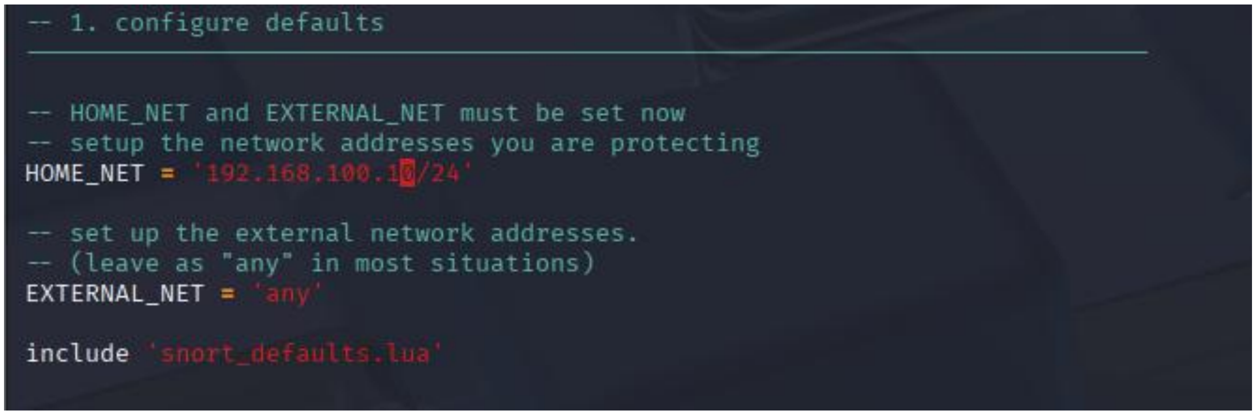
d. Network Confguration

To accommodate multiple roles on the Snort VM, it was configured with two network interfaces: one connected to the Host-Only network for internal monitoring and the other connected to the NAT network to simulate external traffic. The Kali VM acting as a router also serves as a DHCP server, providing IP addresses to clients within the LAN. In this topology, the web server VM is assigned the IP 192.168.100.10, which becomes the new *HOME_NET* – in snort.lua file**,** that Snort will protect and monitor for inbound attacks, as shown in the figures below.



```
┌──(kali㉿kali)-[~]
└─$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:a1:3d:22 brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.1/24 scope global eth0
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:a1:3d:2c brd ff:ff:ff:ff:ff:ff
    inet 192.168.88.133/24 brd 192.168.88.255 scope global dynamic noprefixroute eth1
       valid_lft 1102sec preferred_lft 1102sec
    inet6 fe80::7157:50da:79d:a5b6/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

Figure 12: Kali Router IP



```
-- 1. configure defaults
---------------------------------------------------------------------------

-- HOME_NET and EXTERNAL_NET must be set now
-- setup the network addresses you are protecting
HOME_NET = '192.168.100.10/24'

-- set up the external network addresses.
-- (leave as "any" in most situations)
EXTERNAL_NET = 'any'

include 'snort_defaults.lua'
```

Figure 13: Advanced Snort HOME_NET

2.3.3. Local Rules designed for each attacks

In this project, all prebuilt Snort rules were disabled, and custom local rules were created specifically for the simulated attacks. Each rule is designed to capture the targeted activity and generate alerts according to the intended security objective.

a. Basic Rules

ICMP Ping Detection

To detect ICMP ping activity, Snort rules are crafted to alert on ICMP Echo Request packets directed toward any monitored destination. In the basic setup, the rule does not specify a particular internal IP, allowing detection from any source to any destination within the network.

```
alert icmp any any -> any any (msg:"ICMP Ping detected"; icmp_type:8;
sid:1000001; rev:1;)
```

Snort generates repeated alerts indicating the detection of ICMP Echo Requests from the attacker’s source IP.

ICMP Flood Detection

An attack of ICMP flood may look like

```
Sudo ping – f 192.168.88.132
```

ICMP Flood represents high-frequency bursts of ICMP Echo Requests, indicative of a volumetric denial-of-service attempt. The basic rule incorporates a detection filter to track the number of ICMP packets per source within a short time window. For example,

```
alert icmp any any -> any any (msg:"ICMP Flood detected"; icmp_type:8;
detection_filter:track by_src, count 15, seconds 5; sid:1000002;
rev:1;)
```

Snort triggers alerts when the threshold is exceeded, demonstrating its ability to detect flooding while also generating alerts for normal ICMP traffic.

SYN Flood Detection

SYN Flood attacks generate rapid bursts of TCP SYN packets to exhaust server resources or hide malicious activity. The rule pattern focuses on stateless SYN flags and high packet rate to detect flooding without relying on established connections.

Attack Command for SYN flood

```
hping3 -S -p 80 --flood --rand-source192.168.100.10
```

Snort rules

```
alert tcp any any -> any 80 (msg:"SYN Flood detected"; flags:S;
flow:stateless; detection_filter:track by_dst, count 2000, seconds 1;
sid:1000003; rev:2;)
```

Expected Output:

High-rate SYN packets trigger alerts showing source IP, destination IP, and destination port, allowing the administrator to identify volumetric SYN flood behavior.

b. Advanced Rules

ICMP Ping and Flood Detection (Advanced)

Advanced rules specify the internal host IP (192.168.100.10) to limit monitoring to the protected network segment. This reduces unnecessary alerts from unrelated traffic and enhances performance. The detection patterns remain the same: low-frequency for Ping and high-frequency for Flood.

ICMP Ping:

```
alert icmp any any -> 192.168.100.10 any (msg:"ICMP Ping detected";
icmp_type:8; sid:1000001; rev:1;)
```

ICMP Flood:

```
alert icmp any any -> 192.168.100.10 any (msg:"ICMP Flood detected";
icmp_type:8; detection_filter:track by_src, count 15, seconds 5;
sid:1000002; rev:1;)
```

Expected Output:

Alerts are triggered only for traffic targeting the protected host, reflecting both the low-frequency reconnaissance pattern and the high-frequency flood pattern separately. Specifically, the ICMP Flood attacks will trigger both of the flood and the normal one.

SYN Flood Detection

SYN Flood attacks generate rapid bursts of TCP SYN packets to exhaust server resources or hide malicious activity. The rule pattern focuses on stateless SYN flags and high packet rate to detect flooding without relying on established connections.

SYN Flood command from attacker

```
hping3 -S -p 80 --flood --rand-source192.168.100.10
```

Snort rules

```
alert tcp any any -> 192.168.100.10 80 (msg:"SYN Flood detected";
flags:S; flow:stateless; detection_filter:track by_dst, count 2000,
seconds 1; sid:1000003; rev:2;)
```

Expected Output:

High-rate SYN packets trigger alerts showing source IP, destination IP, and destination port, allowing the administrator to identify volumetric SYN flood behavior.

SQL Injection Detection

SQL Injection attacks embed suspicious SQL statements within HTTP requests. The rule pattern looks for characteristic content strings (' OR '1'='1) in the HTTP URI, leveraging flow tracking to focus on requests sent to the web server.

An attacker sends crafted requests like

```
curl -X POST "http://192.168.88.133/DVWA/login.php" \
     -d "username=' OR '1'='1&password=anything&Login=Login"
```

Rule designed for SQL injection

```
alert tcp any any -> 192.168.100.10 80 (msg:"SQL Injection attempt";
flow:to_server,established; content:"' OR '1'='1"; http_uri;
sid:1000004; rev:3;)
```

Expected Output:

Snort generates alerts for any HTTP request containing the SQL injection pattern, indicating the attacker’s source IP and target host.

SSH Connection Attempt

SSH attempts are identified by TCP packets with the SYN flag directed to the SSH port. The rule pattern monitors connection initiation attempts, which is critical in environments with port forwarding, enabling detection of both legitimate and potentially malicious login attempts.

SSH command:

```
ssh user@192.168.88.133
```

Rule designed for SSH

```
alert tcp 192.168.88.129 any -> 192.168.100.10 22 (msg:"SSH attempt
from external to web server"; flags:S; sid:1000009; rev:1;)
```

Expected Output:

Snort alerts indicate each SSH connection attempt with source and destination details, providing clear detection of attempts over the forwarded port.

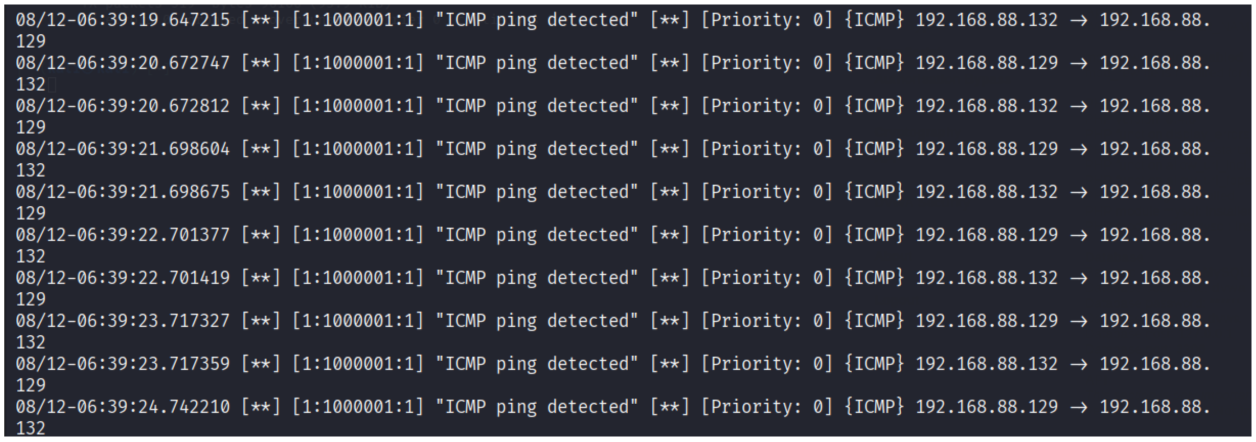
3. Results

3.1 Scenario 1: Ping and ICMP flood detection

ICMP Detection

```
Ping 192.168.88.132
```

Snort output:



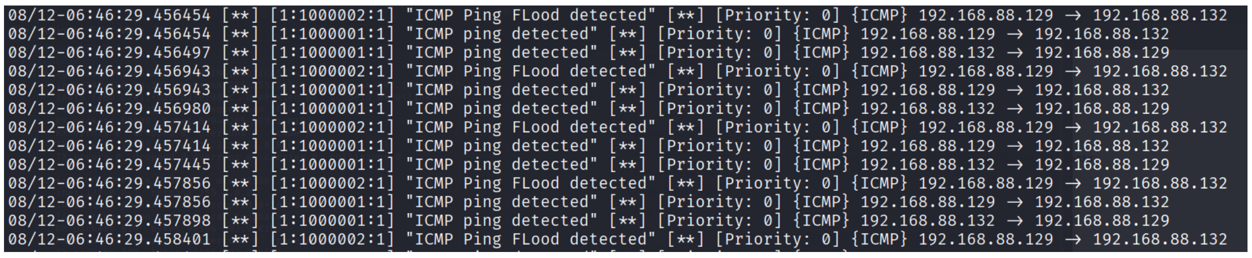
```
08/12-06:39:19.647215 [**] [1:1000001:1] "ICMP ping detected" [**] [Priority: 0] {ICMP} 192.168.88.132 → 192.168.88.
129
08/12-06:39:20.672747 [**] [1:1000001:1] "ICMP ping detected" [**] [Priority: 0] {ICMP} 192.168.88.129 → 192.168.88.
132
08/12-06:39:20.672812 [**] [1:1000001:1] "ICMP ping detected" [**] [Priority: 0] {ICMP} 192.168.88.132 → 192.168.88.
129
08/12-06:39:21.698604 [**] [1:1000001:1] "ICMP ping detected" [**] [Priority: 0] {ICMP} 192.168.88.129 → 192.168.88.
132
08/12-06:39:21.698675 [**] [1:1000001:1] "ICMP ping detected" [**] [Priority: 0] {ICMP} 192.168.88.132 → 192.168.88.
129
08/12-06:39:22.701377 [**] [1:1000001:1] "ICMP ping detected" [**] [Priority: 0] {ICMP} 192.168.88.129 → 192.168.88.
132
08/12-06:39:22.701419 [**] [1:1000001:1] "ICMP ping detected" [**] [Priority: 0] {ICMP} 192.168.88.132 → 192.168.88.
129
08/12-06:39:23.717327 [**] [1:1000001:1] "ICMP ping detected" [**] [Priority: 0] {ICMP} 192.168.88.129 → 192.168.88.
132
08/12-06:39:23.717359 [**] [1:1000001:1] "ICMP ping detected" [**] [Priority: 0] {ICMP} 192.168.88.132 → 192.168.88.
129
08/12-06:39:24.742210 [**] [1:1000001:1] "ICMP ping detected" [**] [Priority: 0] {ICMP} 192.168.88.129 → 192.168.88.
132
```

Figure 14: Scenario 1 Ping alert

ICMP Flood:

```
Sudo ping – f 192.168.88.132
```

Snort Ouput:



```
08/12-06:46:29.456454 [**] [1:1000002:1] "ICMP Ping FLood detected" [**] [Priority: 0] {ICMP} 192.168.88.129 → 192.168.88.132
08/12-06:46:29.456454 [**] [1:1000001:1] "ICMP ping detected" [**] [Priority: 0] {ICMP} 192.168.88.129 → 192.168.88.132
08/12-06:46:29.456497 [**] [1:1000001:1] "ICMP ping detected" [**] [Priority: 0] {ICMP} 192.168.88.132 → 192.168.88.129
08/12-06:46:29.456943 [**] [1:1000002:1] "ICMP Ping FLood detected" [**] [Priority: 0] {ICMP} 192.168.88.129 → 192.168.88.132
08/12-06:46:29.456943 [**] [1:1000001:1] "ICMP ping detected" [**] [Priority: 0] {ICMP} 192.168.88.129 → 192.168.88.132
08/12-06:46:29.456980 [**] [1:1000001:1] "ICMP ping detected" [**] [Priority: 0] {ICMP} 192.168.88.132 → 192.168.88.129
08/12-06:46:29.457414 [**] [1:1000002:1] "ICMP Ping FLood detected" [**] [Priority: 0] {ICMP} 192.168.88.129 → 192.168.88.132
08/12-06:46:29.457414 [**] [1:1000001:1] "ICMP ping detected" [**] [Priority: 0] {ICMP} 192.168.88.129 → 192.168.88.132
08/12-06:46:29.457445 [**] [1:1000001:1] "ICMP ping detected" [**] [Priority: 0] {ICMP} 192.168.88.132 → 192.168.88.129
08/12-06:46:29.457856 [**] [1:1000002:1] "ICMP Ping FLood detected" [**] [Priority: 0] {ICMP} 192.168.88.129 → 192.168.88.132
08/12-06:46:29.457856 [**] [1:1000001:1] "ICMP ping detected" [**] [Priority: 0] {ICMP} 192.168.88.129 → 192.168.88.132
08/12-06:46:29.457898 [**] [1:1000001:1] "ICMP ping detected" [**] [Priority: 0] {ICMP} 192.168.88.132 → 192.168.88.129
08/12-06:46:29.458401 [**] [1:1000002:1] "ICMP Ping FLood detected" [**] [Priority: 0] {ICMP} 192.168.88.129 → 192.168.88.132
```

Figure 15: Scenario 1 Ping Flood alert

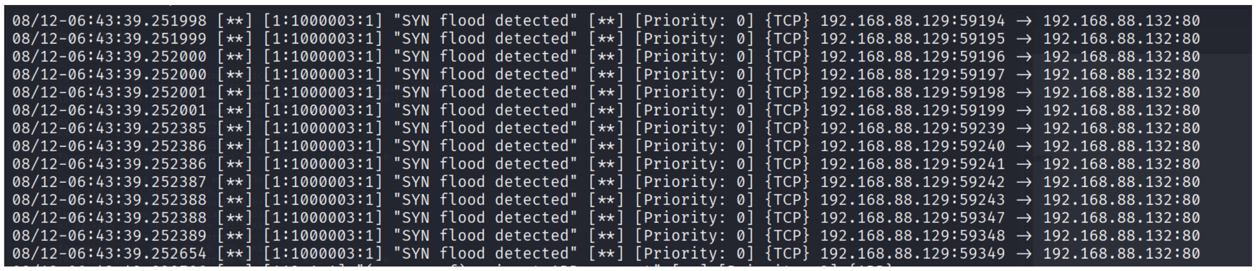
Compared to expected output: Both ping and ICMP flood alerts showed exactly what I wanted when creating rules. Especially the ICMP flood, it triggered both the normal and the flood ICMP alerts.

3.2: Scenario 2: Hping3 SYN Flood detection from internal network

Hping3 SYN Flood:

```
Sudo ping hping3 -S -p 80 192.168.88.132
```

Snort Ouput:



```
08/12-06:43:39.251998 [**] [1:1000003:1] "SYN flood detected" [**] [Priority: 0] {TCP} 192.168.88.129:59194 → 192.168.88.132:80
08/12-06:43:39.251999 [**] [1:1000003:1] "SYN flood detected" [**] [Priority: 0] {TCP} 192.168.88.129:59195 → 192.168.88.132:80
08/12-06:43:39.252000 [**] [1:1000003:1] "SYN flood detected" [**] [Priority: 0] {TCP} 192.168.88.129:59196 → 192.168.88.132:80
08/12-06:43:39.252000 [**] [1:1000003:1] "SYN flood detected" [**] [Priority: 0] {TCP} 192.168.88.129:59197 → 192.168.88.132:80
08/12-06:43:39.252001 [**] [1:1000003:1] "SYN flood detected" [**] [Priority: 0] {TCP} 192.168.88.129:59198 → 192.168.88.132:80
08/12-06:43:39.252001 [**] [1:1000003:1] "SYN flood detected" [**] [Priority: 0] {TCP} 192.168.88.129:59199 → 192.168.88.132:80
08/12-06:43:39.252385 [**] [1:1000003:1] "SYN flood detected" [**] [Priority: 0] {TCP} 192.168.88.129:59239 → 192.168.88.132:80
08/12-06:43:39.252386 [**] [1:1000003:1] "SYN flood detected" [**] [Priority: 0] {TCP} 192.168.88.129:59240 → 192.168.88.132:80
08/12-06:43:39.252386 [**] [1:1000003:1] "SYN flood detected" [**] [Priority: 0] {TCP} 192.168.88.129:59241 → 192.168.88.132:80
08/12-06:43:39.252387 [**] [1:1000003:1] "SYN flood detected" [**] [Priority: 0] {TCP} 192.168.88.129:59242 → 192.168.88.132:80
08/12-06:43:39.252388 [**] [1:1000003:1] "SYN flood detected" [**] [Priority: 0] {TCP} 192.168.88.129:59243 → 192.168.88.132:80
08/12-06:43:39.252388 [**] [1:1000003:1] "SYN flood detected" [**] [Priority: 0] {TCP} 192.168.88.129:59347 → 192.168.88.132:80
08/12-06:43:39.252389 [**] [1:1000003:1] "SYN flood detected" [**] [Priority: 0] {TCP} 192.168.88.129:59348 → 192.168.88.132:80
08/12-06:43:39.252654 [**] [1:1000003:1] "SYN flood detected" [**] [Priority: 0] {TCP} 192.168.88.129:59349 → 192.168.88.132:80
```

Figure 16: Scenario 2 SYN Flood alert

Figure 16: Scenario 2 SYN Flood result

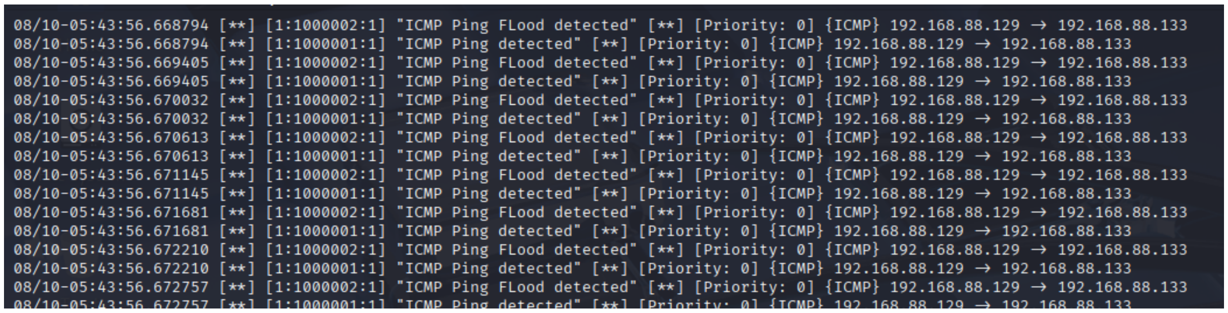
Compared to the expected output: Snort detected SYN flood packets on the outside interface (eth1) due to 1-to-1 NAT mapping. Although running Snort on the internal interface provides precise VM-level detection, using the external interface is sufficient in this topology, as alerts indicate the internal host under attack.

3.3 Scenerio 3: Hping3 and ICMP Ping Flood Detection from external network

ICMP Flood:

```
Sudo ping – f 192.168.88.133
```

Snort output:



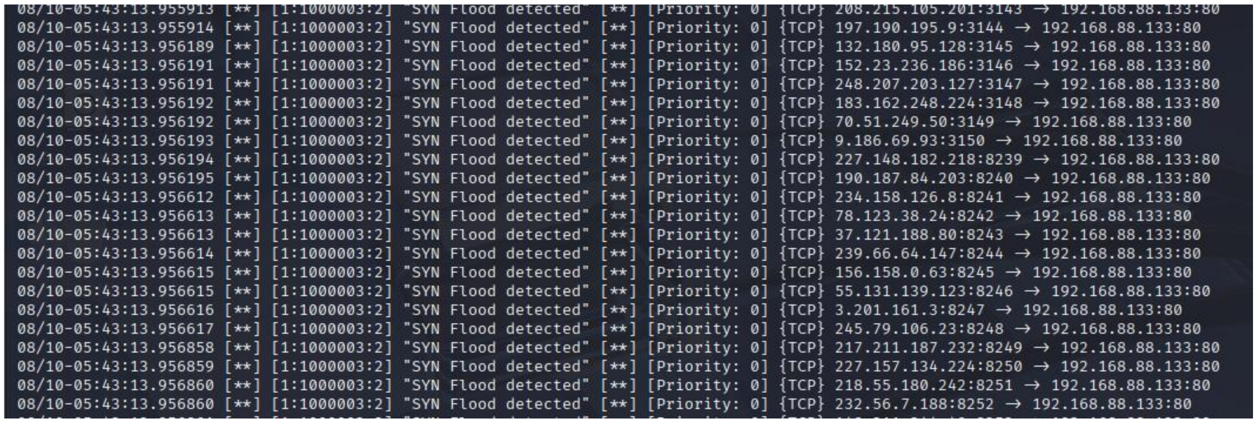
```
08/10-05:43:56.668794 [**] [1:1000002:1] "ICMP Ping FLood detected" [**] [Priority: 0] {ICMP} 192.168.88.129 → 192.168.88.133
08/10-05:43:56.668794 [**] [1:1000001:1] "ICMP Ping detected" [**] [Priority: 0] {ICMP} 192.168.88.129 → 192.168.88.133
08/10-05:43:56.669405 [**] [1:1000002:1] "ICMP Ping FLood detected" [**] [Priority: 0] {ICMP} 192.168.88.129 → 192.168.88.133
08/10-05:43:56.669405 [**] [1:1000001:1] "ICMP Ping detected" [**] [Priority: 0] {ICMP} 192.168.88.129 → 192.168.88.133
08/10-05:43:56.670032 [**] [1:1000002:1] "ICMP Ping FLood detected" [**] [Priority: 0] {ICMP} 192.168.88.129 → 192.168.88.133
08/10-05:43:56.670032 [**] [1:1000001:1] "ICMP Ping detected" [**] [Priority: 0] {ICMP} 192.168.88.129 → 192.168.88.133
08/10-05:43:56.670613 [**] [1:1000002:1] "ICMP Ping FLood detected" [**] [Priority: 0] {ICMP} 192.168.88.129 → 192.168.88.133
08/10-05:43:56.670613 [**] [1:1000001:1] "ICMP Ping detected" [**] [Priority: 0] {ICMP} 192.168.88.129 → 192.168.88.133
08/10-05:43:56.671145 [**] [1:1000002:1] "ICMP Ping FLood detected" [**] [Priority: 0] {ICMP} 192.168.88.129 → 192.168.88.133
08/10-05:43:56.671145 [**] [1:1000001:1] "ICMP Ping detected" [**] [Priority: 0] {ICMP} 192.168.88.129 → 192.168.88.133
08/10-05:43:56.671681 [**] [1:1000002:1] "ICMP Ping FLood detected" [**] [Priority: 0] {ICMP} 192.168.88.129 → 192.168.88.133
08/10-05:43:56.671681 [**] [1:1000001:1] "ICMP Ping detected" [**] [Priority: 0] {ICMP} 192.168.88.129 → 192.168.88.133
08/10-05:43:56.672210 [**] [1:1000002:1] "ICMP Ping FLood detected" [**] [Priority: 0] {ICMP} 192.168.88.129 → 192.168.88.133
08/10-05:43:56.672210 [**] [1:1000001:1] "ICMP Ping detected" [**] [Priority: 0] {ICMP} 192.168.88.129 → 192.168.88.133
08/10-05:43:56.672757 [**] [1:1000002:1] "ICMP Ping FLood detected" [**] [Priority: 0] {ICMP} 192.168.88.129 → 192.168.88.133
08/10-05:43:56.672757 [**] [1:1000001:1] "ICMP Ping detected" [**] [Priority: 0] {ICMP} 192.168.88.129 → 192.168.88.133
```

Figure 17: Scenario 3 ICMP Flood alert

SYN Flood attack:

```
hping3 -S -p 80 --flood --rand-source192.168.100.10
```

Snort Output:



```
08/10-05:43:13.955913 [**] [1:1000003:2] "SYN Flood detected" [**] [Priority: 0] {TCP} 208.215.105.201:3143 → 192.168.88.133:80
08/10-05:43:13.955914 [**] [1:1000003:2] "SYN Flood detected" [**] [Priority: 0] {TCP} 197.190.195.9:3144 → 192.168.88.133:80
08/10-05:43:13.956189 [**] [1:1000003:2] "SYN Flood detected" [**] [Priority: 0] {TCP} 132.180.95.128:3145 → 192.168.88.133:80
08/10-05:43:13.956191 [**] [1:1000003:2] "SYN Flood detected" [**] [Priority: 0] {TCP} 152.23.236.186:3146 → 192.168.88.133:80
08/10-05:43:13.956191 [**] [1:1000003:2] "SYN Flood detected" [**] [Priority: 0] {TCP} 248.207.203.127:3147 → 192.168.88.133:80
08/10-05:43:13.956192 [**] [1:1000003:2] "SYN Flood detected" [**] [Priority: 0] {TCP} 183.162.248.224:3148 → 192.168.88.133:80
08/10-05:43:13.956192 [**] [1:1000003:2] "SYN Flood detected" [**] [Priority: 0] {TCP} 70.51.249.50:3149 → 192.168.88.133:80
08/10-05:43:13.956193 [**] [1:1000003:2] "SYN Flood detected" [**] [Priority: 0] {TCP} 9.186.69.93:3150 → 192.168.88.133:80
08/10-05:43:13.956194 [**] [1:1000003:2] "SYN Flood detected" [**] [Priority: 0] {TCP} 227.148.182.218:8239 → 192.168.88.133:80
08/10-05:43:13.956195 [**] [1:1000003:2] "SYN Flood detected" [**] [Priority: 0] {TCP} 190.187.84.203:8240 → 192.168.88.133:80
08/10-05:43:13.956612 [**] [1:1000003:2] "SYN Flood detected" [**] [Priority: 0] {TCP} 234.158.126.8:8241 → 192.168.88.133:80
08/10-05:43:13.956613 [**] [1:1000003:2] "SYN Flood detected" [**] [Priority: 0] {TCP} 78.123.38.24:8242 → 192.168.88.133:80
08/10-05:43:13.956613 [**] [1:1000003:2] "SYN Flood detected" [**] [Priority: 0] {TCP} 37.121.188.80:8243 → 192.168.88.133:80
08/10-05:43:13.956614 [**] [1:1000003:2] "SYN Flood detected" [**] [Priority: 0] {TCP} 239.66.64.147:8244 → 192.168.88.133:80
08/10-05:43:13.956615 [**] [1:1000003:2] "SYN Flood detected" [**] [Priority: 0] {TCP} 156.158.0.63:8245 → 192.168.88.133:80
08/10-05:43:13.956615 [**] [1:1000003:2] "SYN Flood detected" [**] [Priority: 0] {TCP} 55.131.139.123:8246 → 192.168.88.133:80
08/10-05:43:13.956616 [**] [1:1000003:2] "SYN Flood detected" [**] [Priority: 0] {TCP} 3.201.161.3:8247 → 192.168.88.133:80
08/10-05:43:13.956617 [**] [1:1000003:2] "SYN Flood detected" [**] [Priority: 0] {TCP} 245.79.106.23:8248 → 192.168.88.133:80
08/10-05:43:13.956858 [**] [1:1000003:2] "SYN Flood detected" [**] [Priority: 0] {TCP} 217.211.187.232:8249 → 192.168.88.133:80
08/10-05:43:13.956859 [**] [1:1000003:2] "SYN Flood detected" [**] [Priority: 0] {TCP} 227.157.134.224:8250 → 192.168.88.133:80
08/10-05:43:13.956860 [**] [1:1000003:2] "SYN Flood detected" [**] [Priority: 0] {TCP} 218.55.180.242:8251 → 192.168.88.133:80
08/10-05:43:13.956860 [**] [1:1000003:2] "SYN Flood detected" [**] [Priority: 0] {TCP} 232.56.7.188:8252 → 192.168.88.133:80
```

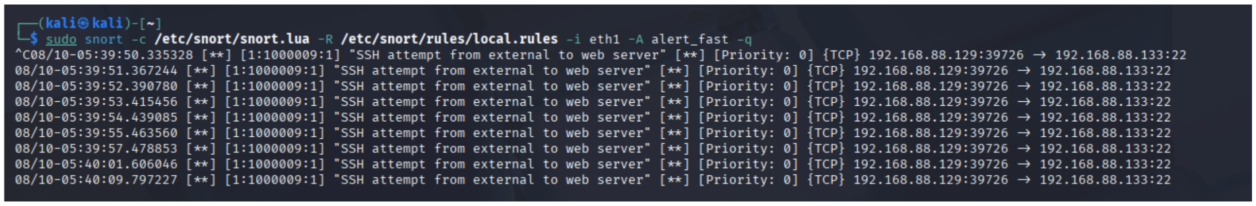
Figure 18: Scenario 3 SYN Flood alert

Compared to the expected output: The first attack triggered both ICMP ping and ICMP flood alerts, showing attacker source IP and target IP. The SYN flood also triggered expected alerts, displaying the attacker's IP, target IP, and destination port. Results matched the intended behavior of the configured rules.

3.4 Scenerio 4: SSH access detection

```
ssh user@192.168.88.133
```

Snort Output:



```
┌──(kali㉿kali)-[~]
└─$ sudo snort -c /etc/snort/snort.lua -R /etc/snort/rules/local.rules -i eth1 -A alert_fast -q
^C08/10-05:39:50.335328 [**] [1:1000009:1] "SSH attempt from external to web server" [**] [Priority: 0] {TCP} 192.168.88.129:39726 → 192.168.88.133:22
08/10-05:39:51.367244 [**] [1:1000009:1] "SSH attempt from external to web server" [**] [Priority: 0] {TCP} 192.168.88.129:39726 → 192.168.88.133:22
08/10-05:39:52.390780 [**] [1:1000009:1] "SSH attempt from external to web server" [**] [Priority: 0] {TCP} 192.168.88.129:39726 → 192.168.88.133:22
08/10-05:39:53.415456 [**] [1:1000009:1] "SSH attempt from external to web server" [**] [Priority: 0] {TCP} 192.168.88.129:39726 → 192.168.88.133:22
08/10-05:39:54.439085 [**] [1:1000009:1] "SSH attempt from external to web server" [**] [Priority: 0] {TCP} 192.168.88.129:39726 → 192.168.88.133:22
08/10-05:39:55.463560 [**] [1:1000009:1] "SSH attempt from external to web server" [**] [Priority: 0] {TCP} 192.168.88.129:39726 → 192.168.88.133:22
08/10-05:39:57.478853 [**] [1:1000009:1] "SSH attempt from external to web server" [**] [Priority: 0] {TCP} 192.168.88.129:39726 → 192.168.88.133:22
08/10-05:40:01.606046 [**] [1:1000009:1] "SSH attempt from external to web server" [**] [Priority: 0] {TCP} 192.168.88.129:39726 → 192.168.88.133:22
08/10-05:40:09.797227 [**] [1:1000009:1] "SSH attempt from external to web server" [**] [Priority: 0] {TCP} 192.168.88.129:39726 → 192.168.88.133:22
```

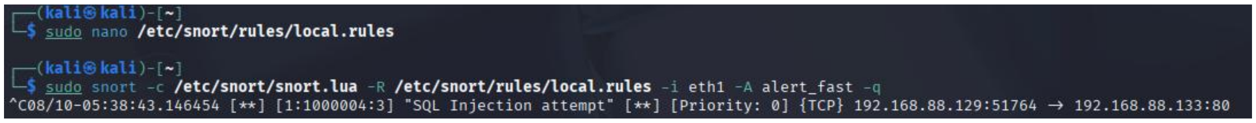
Figure 19: Scenario 4 SSH alert

Compared to the expected output: SSH attempt triggered the alert with correct source and destination IP and port information, confirming proper detection.

3.5 Scenerio 5: SQL injections detection.

```
curl -X POST "http://192.168.88.133/DVWA/login.php" \
     -d "username=' OR '1'='1&password=anything&Login=Login"
```

Snort Output:



```
┌──(kali㉿kali)-[~]
└─$ sudo nano /etc/snort/rules/local.rules

┌──(kali㉿kali)-[~]
└─$ sudo snort -c /etc/snort/snort.lua -R /etc/snort/rules/local.rules -i eth1 -A alert_fast -q
^C08/10-05:38:43.146454 [**] [1:1000004:3] "SQL Injection attempt" [**] [Priority: 0] {TCP} 192.168.88.129:51764 → 192.168.88.133:80
```

Figure 20: Scenario 5 SQL Injection alert

Compared to the expected output: The SQL injection attempts were successfully detected, triggering alerts with the attacker’s IP and the targeted service.

3.6. Summary Table of Results

Scenario	Attack Command	Listening Interface	Rule Purpose	Result	Snort Alert
1	ping 192.168.88.133	eth0	Normal ICMP	Yes	Normal ICMP alert
1	ping -f 192.168.88.133	eth0	ICMP Flood	Yes	ICMP flood alert + normal ICMP
2	hping3 -S -p 80 192.168.100.10 --flood	eth1	SYN Flood	Yes	SYN flood alert
3	sudo ping -f 192.168.88.133	eth1	ICMP Flood	Yes	ICMP flood alert + normal ICMP
3	hping3 -S -p 80 192.168.100.10 --flood	eth1	SYN Flood	Yes	SYN flood alert
4	ssh user@192.168.88.133	eth1	SSH Attempt	Yes	SSH alert
5	' OR '1'='1	eth1	SQL Injection	Yes	SQL Injection alert

Table 3: Table of Results

3.7. Conclusion

The choice of Snort interface significantly affects detection visibility, especially in NAT environments. Monitoring the internal interface provides direct visibility of which internal VM is targeted, whereas monitoring the external interface detects incoming attacks before NAT translation. In scenarios with 1-to-1 NAT mapping, external monitoring is sufficient, as alerts contain the translated internal destination. For environments with multiple internal hosts behind NAT, selecting the appropriate interface is crucial to accurately identify which host is under attack and to correlate alerts with the correct internal IP. Furthermore, advanced configurations such as port forwarding require careful alignment between Snort interface monitoring and iptables rules to ensure that forwarded services (e.g., HTTP, SSH) are correctly monitored without generating false negatives. This approach allows administrators to conduct both reactive incident response and proactive network security analysis.