VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY

UNIVERSITY OF INFORMATION TECHNOLOGY

FINAL REPORT

LIBRENMS+SURICATA

Instructor: **MSc. Trần Thị Dung**

Student: **Đào Minh Châu - 24560025**

HO CHI MINH CITY

August 24, 2025

Table of Contents

Table of Figures

Table of Tables

List of Abbreviations

Abbreviation	Full Term
API	Application Programming Interface
CPU	Central Processing Unit
DHCP	Dynamic Host Configuration Control
HTTP/HTTPS	Hyper Text Transfer Protocol (Secure)
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
JSON	JavaScript Object Notation
NAT	Network Address Translation
NMS	Network Monitor System
REST	Representational State Transfer
SNMP	Simple Network Management Protocol
SSH	Secure Shell
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VM	Virtual Machine

1. Introduction

1.1. Suricata Definition

Suricata is an open-source **Intrusion Detection System (IDS)**, **Intrusion Prevention System (IPS)**, and **Network Security Monitoring (NSM)** engine developed by the **Open Information Security Foundation (OISF)**. It performs real-time analysis of network traffic to detect and prevent threats like port scans, malware, and brute force attacks using rule-based detection, deep packet inspection, and protocol analysis. With support for over 20 protocols (e.g., HTTP, TLS, DNS), multi-threaded performance, and JSON-based logging, Suricata integrates seamlessly with tools like LibreNMS or SIEM systems, making it a versatile choice for securing small to enterprise-scale networks.

Initiated in **2008** by the OISF with funding from the U.S. Department of Homeland Security and industry partners, Suricata aimed to create a high-performance, community-driven alternative to tools like Snort. Its first stable release in **2010** introduced core IDS/IPS capabilities, followed by advancements like Lua scripting (2014) for complex threat detection and a Rust-based JSON generator (2016) for high-throughput logging. By **2025**, Suricata has evolved into version 7.0.8+, supporting SCADA protocols (e.g., EtherNet/IP), cloud deployments, and over 40,000 rules from the Emerging Threats Open ruleset, solidifying its role in modern network security.

Key aspects to understand include its multi-threaded architecture for scalability, extensive protocol parsing, and flexible operation modes (IDS, IPS, NSM). Suricata's strengths lie in its performance, community support, and integration capabilities, but it requires robust hardware and careful rule tuning to minimize false positives. Regular updates via suricata-update, monitoring resource usage, and starting in IDS mode for initial tuning are critical for effective deployment.

1.2. Suricata Components

Components below are necessary to implement a Suricata-based IDS:

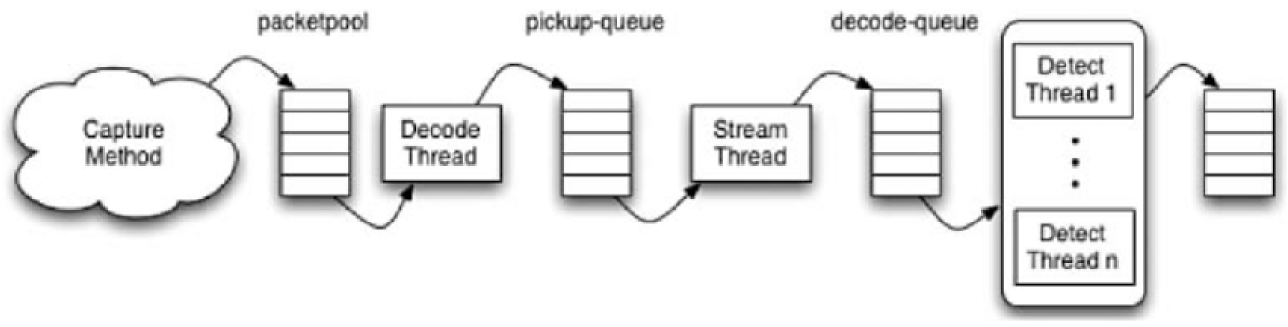


Figure 1: Suricata Architecture

Packet Acquisition Engine

The Packet Acquisition Engine is responsible for capturing network packets for analysis, serving as the entry point for Suricata's processing pipeline. It interfaces with network hardware or software to collect raw packets from interfaces like eth0, supporting technologies such as AF_PACKET, PF_RING, DPDK, and Netmap for high-performance capture. This component ensures Suricata can handle high-speed traffic, up to 100 Gbps in optimized setups, by leveraging multi-threading to distribute packet capture across CPU cores.

Packet Decoder

The **Packet Decoder** processes raw packets captured by the acquisition engine, extracting meaningful data by decoding network protocols at various layers (e.g., Ethernet, IP, TCP, UDP). It parses packet headers and payloads, reconstructing streams and identifying protocols (e.g., HTTP, TLS, DNS) even on non-standard ports, enabling Suricata to analyze application-layer content for threats. This component works by dissecting packets into their constituent parts, such as source/destination IPs, ports, and payload data, which are then passed to the detection engine. The decoder uses Suricata's multi-threaded architecture to process packets in parallel, ensuring low latency, allowing fine-tuned analysis based on network needs.

Detection Engine

The Detection Engine is the core of Suricata's threat identification, analyzing decoded packets against a set of rules to detect malicious activity. It uses signature-based detection (via rules like those from the Emerging Threats Open ruleset) and supports advanced techniques like anomaly detection and Lua scripting for custom logic, enabling identification of threats such as port scans, DDoS attacks, or SSH brute force attempts.

Output and Logging System

The Output and Logging System handles the recording and reporting of detected events, generating logs in formats like EVE JSON, fast.log, or pcap for forensic analysis. It supports multiple output types, including alerts, HTTP logs, TLS certificate details, and full packet captures, making it essential for integration with SIEM systems (e.g., Splunk, Elastic Stack) and post-incident analysis. The system supports conditional logging to reduce storage needs and uses a Rust-based JSON generator (introduced in Suricata 6) for high-performance logging, capable of handling thousands of events per second.

1.3 Suricata Operation

1.3.1. Suricata Architecture and Workflow

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

The Packet Acquisition Engine is the first component in Suricata's workflow, responsible for capturing raw network packets from a specified interface. It interfaces with network hardware or software using technologies like AF_PACKET, PF_RING, DPDK, or Netmap, enabling high-speed packet capture, even in environments handling 10-100 Gbps traffic. The engine distributes captured packets across multiple threads for parallel processing, using load balancing modes like cluster_flow to optimize performance. The engine captures traffic from the protected network and forwards it to the next component. Hardware acceleration can further enhance performance by bypassing the OS kernel, reducing packet loss under heavy loads.

Next, the Packet Decoder processes raw packets from the acquisition engine, decoding them into structured data by parsing network protocol layers (e.g., Ethernet, IP, TCP, UDP). It extracts critical information like source/destination IPs, ports, and payloads, reconstructing packet streams to enable analysis of higher-layer protocols, even on non-standard ports, which is essential for identifying threats like HTTP-based exploits or SSH brute force attempts. This component operates by dissecting each packet's headers and payloads, passing the decoded data to the detection engine. The decoder's multi-threaded design, ensures efficient processing, minimizing latency even in high-traffic scenarios.

As the core analytical component, the Detection Engine is the core of Suricata's threat identification, analyzing decoded packets against a set of rules to detect malicious activities such as port scans, ICMP floods, or SSH brute force attacks. It uses signature-based detection with rules from sources like the Emerging Threats Open ruleset (over 40,000 rules) or custom rules for complex threat patterns.

Finally, The Application Layer Parsers analyze application-layer protocols (e.g., HTTP, DNS, TLS, FTP, SMTP) by parsing their specific structures, extracting metadata like HTTP URLs, DNS queries, or TLS certificates. This enables Suricata to detect sophisticated threats, such as SQL injection in HTTP requests or malicious TLS sessions, which require content-level inspection beyond packet headers. The parsers reconstruct application sessions from packet streams, and work in tandem with the detection engine, providing enriched data for precise threat identification, and support automatic protocol detection on non-standard ports, ensuring robust coverage in diverse network environments.

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.

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

To ensure Suricata rules are effective and minimize false positives, they must be precisely crafted based on specific criteria. Key considerations include:

- Protocol and Port Precision: Define rules for specific protocols (e.g., TCP, UDP, ICMP) and ports (e.g., 22 for SSH, 80 for HTTP) to target relevant traffic.

- IP Address Specificity: Use network variables (e.g., $HOME_NET for 192.168.100.0/24) instead of generic any to focus on known network segments.

- Content-Based Detection: Include unique strings or byte patterns in malicious payloads (e.g., content:"SELECT * FROM" for SQL injection) to enhance accuracy.

- Flow and TCP Flags: Leverage flow direction (to_server, to_client) and TCP flags (e.g., flags:S for SYN) to pinpoint attack behaviors.

- Thresholding for Noise Reduction: Apply thresholds (e.g., threshold:type both,track by_src,count 5,seconds 60) to filter benign high-frequency events, ensuring alerts are meaningful.

d. Incorporate Metadata and Documentation

To improve Suricata rule effectiveness and maintainability, incorporate comprehensive metadata and documentation. Each rule should include:

- Message (msg) Field: A concise, clear description of the detected threat (e.g., msg:"SSH Brute Force Attempt Detected").

- SID (Suricata ID): A unique identifier (e.g., sid:1000001) for rule tracking and management.

- Revision Number (rev): A version control number (e.g., rev:1) to track rule updates.

1.4. LibreNMS Definition

LibreNMS is an open-source, web-based network monitoring system designed to provide detailed insights into network performance, device health, and operational status. Built on PHP and leveraging a MariaDB database for storage, LibreNMS uses Simple Network Management Protocol (SNMP) to poll devices, collecting metrics such as bandwidth usage, CPU load, memory utilization, and uptime. These metrics are stored in RRDtool for time-series analysis and displayed on a customizable, user-friendly dashboard accessible via a web interface. Its ability to auto-discover devices, support a wide range of hardware, and integrate with external systems like Suricata via REST API makes it a versatile tool for network administrators.

Originally forked from Observium in 2013, LibreNMS was created to offer a fully open-source, community-driven alternative with enhanced flexibility and frequent updates. Licensed under GPLv3, it has grown into a robust platform supported by a global community of contributors who add features, device support, and integrations. LibreNMS supports monitoring of diverse devices, including routers, switches, servers, and IoT devices, across vendors like Cisco, Juniper, and MikroTik. Its auto-discovery feature uses protocols like CDP, LLDP, or ARP to identify devices, reducing manual configuration. In the lab environment, LibreNMS polls devices every 5 minutes, generating graphs for traffic (e.g., 2.3 MB/s baseline on the Kali VM) and alerts for anomalies, which are critical for real-time network oversight and integration with Suricata's intrusion detection alerts.

Key features of LibreNMS include its scalable architecture, customizable alerting, and extensive integration capabilities. It supports distributed polling for large networks, ensuring performance in enterprise settings, and offers alerting via email, Slack, or custom scripts. The REST API enables seamless integration with external tools, such as the Python daemon in the lab setup that forwards Suricata's EVE JSON alerts to LibreNMS for unified visualization. LibreNMS also provides detailed reporting, billing modules for bandwidth usage, and support for advanced protocols like NetFlow for traffic analysis. While resource-efficient (e.g., running on 8GB RAM in the lab), it requires proper configuration of SNMP agents on monitored devices (e.g., Ubuntu 22.04 VMs) and periodic maintenance to manage database growth. For further details, refer to the LibreNMS Documentation.

1.5. LibreNMS Components&Operation

Its modular architecture comprises several key components that work together to collect, process, store, and visualize network metrics:

-Polling Engine: the core component responsible for collecting metrics from network devices using protocols like SNMP (Simple Network Management Protocol), ICMP, and agent-based methods. It periodically queries devices to gather data such as CPU usage, memory utilization, bandwidth, and interface status, ensuring real-time monitoring of network health. LibreNMS supports distributed polling for scalability, using cron jobs to schedule tasks. The engine's multi-threaded design ensures efficient processing, storing raw data in MariaDB for further analysis and visualization.

-Database Backend: typically MariaDB, stores all collected metrics, device configurations, and alert data for persistence and querying. It organizes data into tables for devices, interfaces, and performance metrics, enabling historical analysis and trend reporting. The backend supports efficient querying for dashboard displays, with schema optimizations ensuring performance even with large datasets.

-RRDtool Storage: component manages time-series data for network metrics, storing them in Round-Robin Database (RRD) files for efficient, fixed-size storage and graphing. It handles metrics like bandwidth and uptime, generating compact data structures that enable long-term trend analysis without excessive disk usage. RRDtool operates by aggregating polled data into time-based archives, configured via /opt/librenms/config.php. The polling engine writes metrics to RRD files, which are then used by the web interface to render graphs.

-Web Interface: powered by Apache2 and PHP, provides a user-friendly dashboard for visualizing device metrics, alerts, and network status. It displays graphs, device lists, and integrated alerts in a customizable format, the interface supports user authentication and role-based access for secure management.

-Alerting System: monitors collected metrics and external inputs against predefined rules to generate notifications for anomalies, such as high CPU usage or security threats. The system's flexibility supports custom rules, ensuring precise alerting tailored to the network's needs.

-Discovery Engine: automatically identifies devices and services on the network using protocols like CDP, LLDP, or ARP, reducing manual configuration. It uses SNMP to query device details (e.g., hostname, OS) and updates the database with new devices.

-API Integration Layer: enables LibreNMS to interface with external systems, such as Suricata, via a REST API, facilitating data exchange and automation. This component ensures seamless integration, enhancing the system's ability to correlate network metrics with security events.

2. Implementation

2.1. Topology

Basic Topology: The NMS and IDS work separately from each other

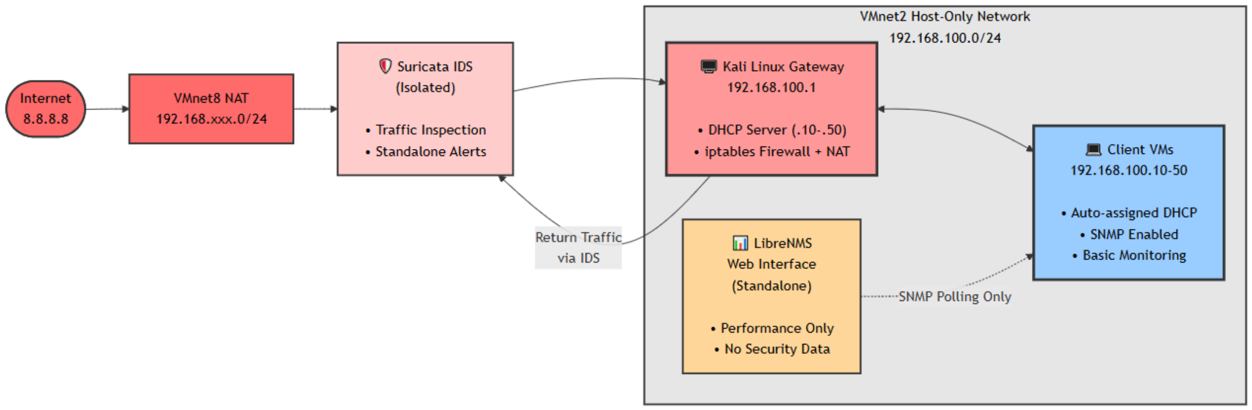


Figure 2: Basic Topology

VM Name	Role	IP Address
Kali-vm	DHCP Server, Libre host, Suricata host, Gateway	192.168.100.1
Client-vm	Monitored Device	192.168.100.10
Attacker-vm	External Network Attacker	

Table 1: Basic Components

Advanced Topology:

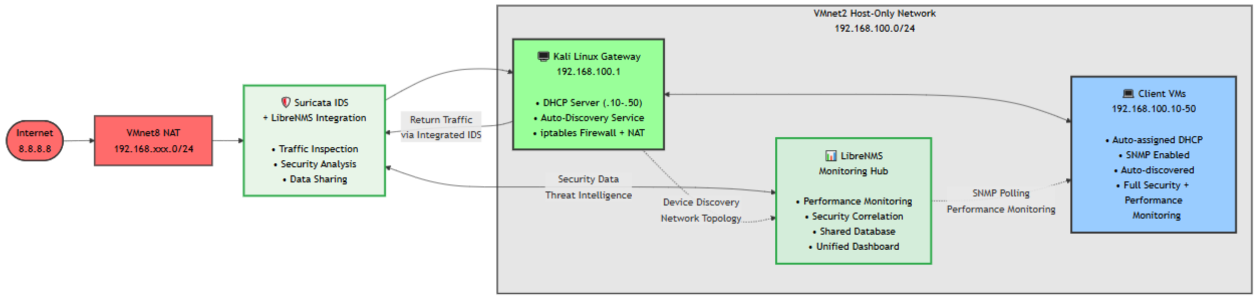


Figure 3: Advanced Topology

VM Name	Role	IP Address
Kali-vm	DHCP Server, Libre host, Suricata host, Gateway	192.168.100.1
Client-vm	Monitored Device	192.168.100.10
Attacker-vm	External Network Attacker	

Table 2: Advanced Component

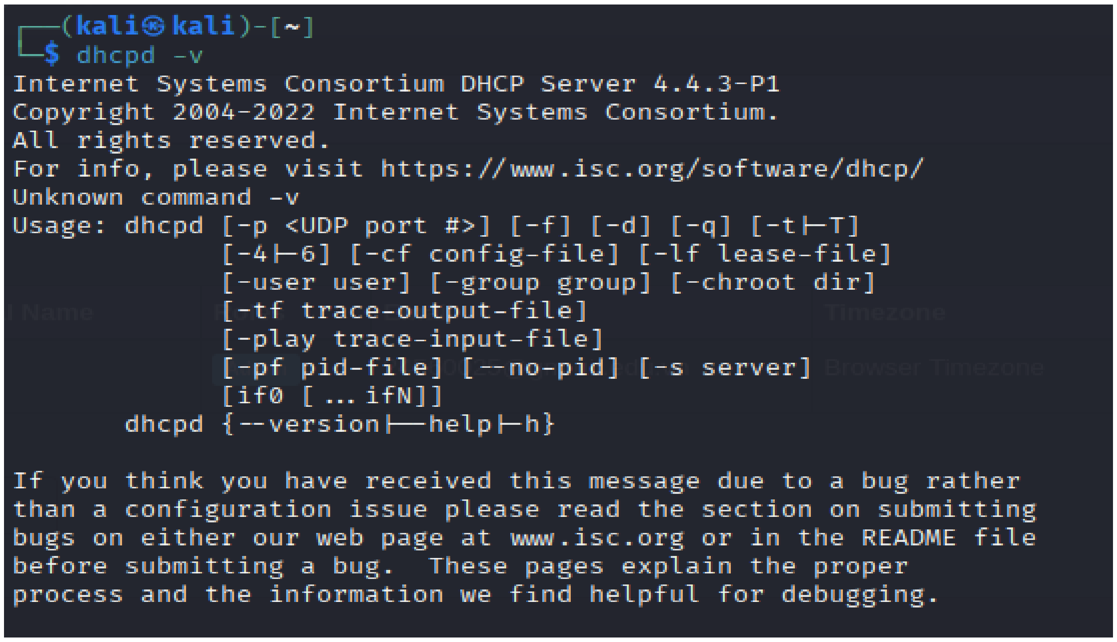
2.2 Installation

2.2.1 DHCP Installation

On Kali-vm, use these command to update the system and install DHCP:

```
sudo apt update && sudo apt upgrade -y
sudo apt dist-upgrade -y
sudo apt install isc-dhcp-server -y
```

Then verify the installation like below:



```
┌──(kali㉿kali)-[~]
└─$ dhcpd -v
Internet Systems Consortium DHCP Server 4.4.3-P1
Copyright 2004-2022 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/
Unknown command -v
Usage: dhcpd [-p <UDP port #>] [-f] [-d] [-q] [-t|-T]
             [-4|-6] [-cf config-file] [-lf lease-file]
             [-user user] [-group group] [-chroot dir]
             [-tf trace-output-file]
             [-play trace-input-file]
             [-pf pid-file] [--no-pid] [-s server]
             [if0 [ ... ifN]]
       dhcpd {--version|--help|-h}

If you think you have received this message due to a bug rather
than a configuration issue please read the section on submitting
bugs on either our web page at www.isc.org or in the README file
before submitting a bug.  These pages explain the proper
process and the information we find helpful for debugging.
```

Figure 4: DHCP Server Verification.

Install other required packages by executing this command:

```
sudo apt install -y apache2 mariadb-server mariadb-client php php-cli php-common
php-curl php-fpm php-gd php-gmp php-json php-mbstring php-mysql php-snmp php-xml
php-zip libapache2-mod-php curl composer fping git graphviz imagemagick mtr-tiny
nmap rrdtool snmp snmp-mibs-downloader unzip python3-pymysql python3-dotenv
python3-redis python3-setuptools python3-systemd python3-pip acl whois iptables-
persistent
```

2.2.2. LibreNMS Installation:

-Create Librenms User and Directory:

```
sudo useradd librenms -d /opt/librenms -M -r -s "$(which bash)"
sudo mkdir -p /opt/librenms
sudo chown librenms:librenms /opt/librenms
```

-Download LibreNMS Source:

```
cd /opt
sudo git clone https://github.com/librenms/librenms.git
    sudo chown -R librenms:librenms /opt/librenms
sudo chmod 771 /opt/librenms
sudo setfacl -d -m g::rwx /opt/librenms/rrd /opt/librenms/logs
/opt/librenms/bootstrap/cache/ /opt/librenms/storage/
```

```
sudo setfacl -R -m g::rwx /opt/librenms/rrd /opt/librenms/logs
/opt/librenms/bootstrap/cache/ /opt/librenms/storage/
```

-Install PHP Dependencies:

```
sudo su - librenms
cd /opt/librenms
./scripts/composer_wrapper.php install --no-dev
exit
```

2.2.3. Suricata Installation:

Suricata serves as the primary Intrusion Detection System (IDS) component, providing real-time network traffic analysis and threat detection capabilities.

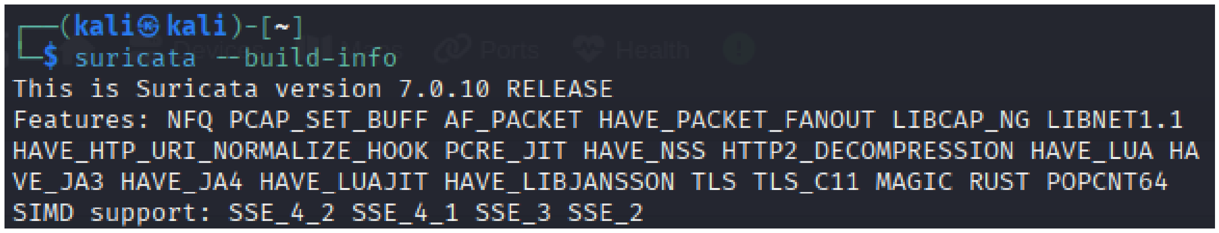
Issue the following command to install Suricata and its dependencies:

```
sudo apt install -y suricata suricata-update jq
```

The installation includes:

- suricata: Core IDS/IPS engine
- suricata-update: Rule management utility for maintaining current threat signatures
- jq: JSON processor for parsing Suricata's EVE JSON output format

Verify if Suricata is installed or not :



```
┌──(kali㉿kali)-[~]
└─$ suricata --build-info
This is Suricata version 7.0.10 RELEASE
Features: NFQ PCAP_SET_BUFF AF_PACKET HAVE_PACKET_FANOUT LIBCAP_NG LIBNET1.1
HAVE_HTP_URI_NORMALIZE_HOOK PCRE_JIT HAVE_NSS HTTP2_DECOMPRESSION HAVE_LUA HA
VE_JA3 HAVE_JA4 HAVE_LUAJIT HAVE_LIBJANSSON TLS TLS_C11 MAGIC RUST POPCNT64
SIMD support: SSE_4_2 SSE_4_1 SSE_3 SSE_2
```

Figure 5: Suricata Installation.

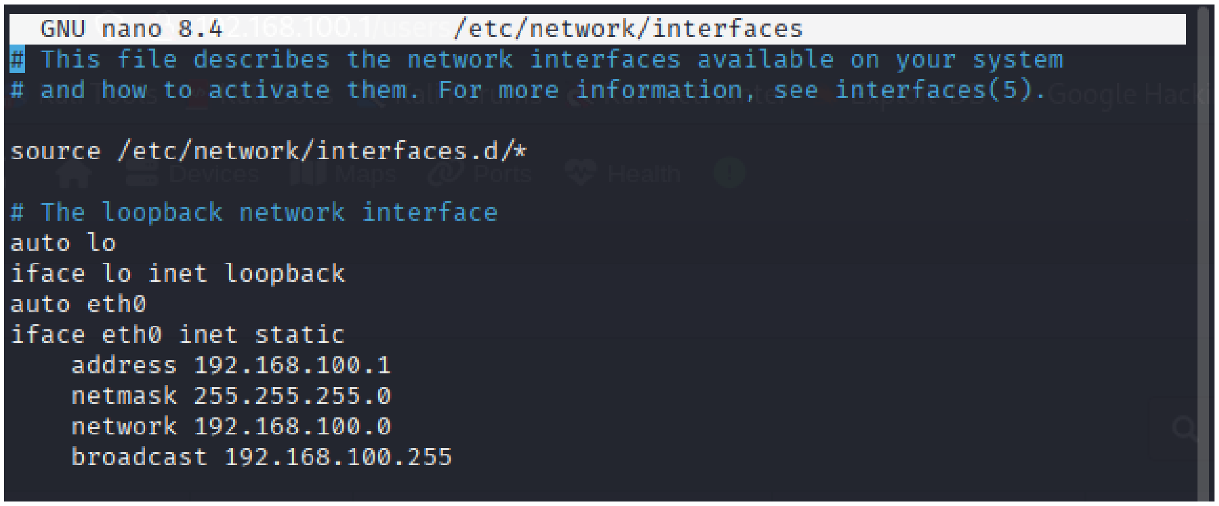
2.3. Configuration

2.3.1. Basic Configuration:

Network Interface Configuration:

The foundational step involves configuring the network interfaces on the Kali-vm to support both internal network management and external internet connectivity.

```
sudo nano /etc/network/interfaces
```
Make sure that the file content is like below:

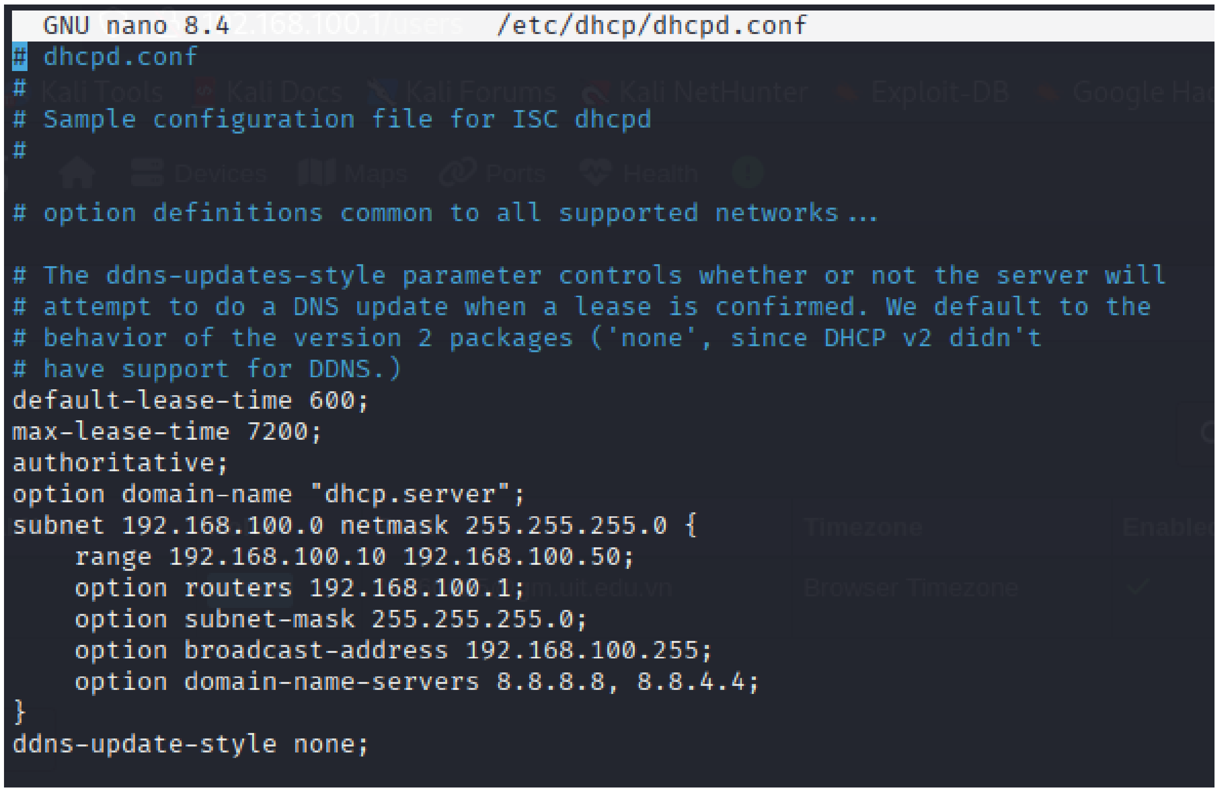


```
  GNU nano 8.4                    /etc/network/interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback
auto eth0
iface eth0 inet static
    address 192.168.100.1
    netmask 255.255.255.0
    network 192.168.100.0
    broadcast 192.168.100.255
```

Figure 6: Interfaces Configuration

DHCP Server Basic Configuration:

```
sudo nano /etc/dhcp/dhcpd.conf
```



```
  GNU nano 8.4                    /etc/dhcp/dhcpd.conf
# dhcpd.conf
#
# Sample configuration file for ISC dhcpd
#

# option definitions common to all supported networks...

# The ddns-updates-style parameter controls whether or not the server will
# attempt to do a DNS update when a lease is confirmed. We default to the
# behavior of the version 2 packages ('none', since DHCP v2 didn't
# have support for DDNS.)
default-lease-time 600;
max-lease-time 7200;
authoritative;
option domain-name "dhcp.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 subnet-mask 255.255.255.0;
    option broadcast-address 192.168.100.255;
    option domain-name-servers 8.8.8.8, 8.8.4.4;
}
ddns-update-style none;
```

Figure 7 : DHCP Configuration

Interface Assignment:

```
sudo nano /etc/default/isc-dhcp-server
```

Set :

```
INTERFACESv4="eth0"
INTERFACESv6=""
```

Enable IP Forwarding:

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

Add or uncomment this :

```
net.ipv4.ip_forward=1
```

Apply changes:

```
Sudo sysctl -p
```

IPTables NAT Configuration:

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

Save the iptables permanently:

```
sudo netfilter-persistent save
```

LibreNMS Database Configuration:

MariaDB Setup:

```
sudo systemctl enable mariadb
sudo systemctl start mariadb
sudo mysql -u root -p
```

In MySQL, add these commands:

```
CREATE DATABASE librenms CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
CREATE USER 'librenms'@'localhost' IDENTIFIED BY 'LibreNMS2024!';
GRANT ALL PRIVILEGES ON librenms.* TO 'librenms'@'localhost';
FLUSH PRIVILEGES;
exit
```

Configure MariaDB for Libre:

```
sudo nano /etc/mysql/mariadb.conf.d/50-server.cnf
```

Add this to the file :

```
[mysqld]
```

```
innodb_file_per_table=1
lower_case_table_names=0
innodb_buffer_pool_size=256M
innodb_flush_log_at_trx_commit=2
innodb_log_file_size=128M
```

Restart MariaDB:

```
sudo systemctl restart mariadb
```

Apache2 Virtual Host Configuration:

Enable Apache2 modules:

```
sudo a2enmod rewrite
sudo a2enmod headers
sudo a2enmod php8.2
```

Create LibreNMS virtual host:

```
sudo nano /etc/apache2/sites-available/librenms.conf
```

Add this to the file :

```
<VirtualHost *:80>
    ServerName 192.168.100.1
    DocumentRoot /opt/librenms/html/

    CustomLog /var/log/apache2/librenms_access.log combined
    ErrorLog /var/log/apache2/librenms_error.log

    AllowEncodedSlashes NoDecode

    <Directory "/opt/librenms/html/">
        Require all granted
        AllowOverride All
        Options FollowSymLinks MultiViews
    </Directory>

    # Deny access to sensitive directories
    <Directory "/opt/librenms/config/">
        Require all denied
    </Directory>

    <Directory "/opt/librenms/includes/">
        Require all denied
    </Directory>

    <Directory "/opt/librenms/logs/">
```

```
        Require all denied
    </Directory>
</VirtualHost>
```

Enable the site and disable the default page:

```
sudo a2ensite librenms
sudo a2dissite 000-default
sudo systemctl restart apache2
```

Configure PHP:

Check PHP installation and adjust the file:

```
php -v
sudo nano /etc/php/8.4/apache2/php.ini
```

My PHP version is 8.4, might be different for other machines

Modify these settings:

```
date.timezone = Asia/Ho_Chi_Minh
memory_limit = 256M
upload_max_filesize = 16M
post_max_size = 16M
max_execution_time = 300
max_input_vars = 3000
```

Modify the same with PHP CLI:

```
sudo nano /etc/php/8.4/cli/php.ini
```

Restart apache to make changes :

```
sudo systemctl restart apache2
```

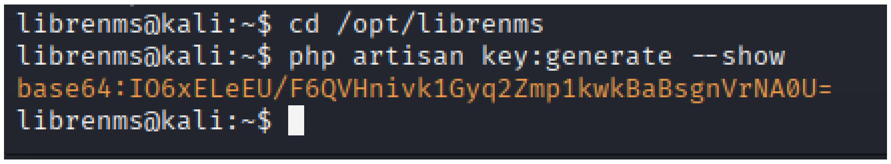
Configure LibreNMS Environment :

Copy the example file and adjust permission:

```
sudo cp /opt/librenms/.env.example /opt/librenms/.env
sudo chown librenms:librenms /opt/librenms/.env
sudo chmod 640 /opt/librenms/.env
```

Generate application key(This is critical, LibreNMS won’t work without APP_KEY):

```
sudo su - librenms
cd /opt/librenms
php artisan key:generate --show
```



```
librenms@kali:~$ cd /opt/librenms
librenms@kali:~$ php artisan key:generate --show
base64:IO6xELeEU/F6QVHnivk1Gyq2Zmp1kwkBaBsgnVrNA0U=
librenms@kali:~$
```

Figure 8: LibreNMS Key Generation.

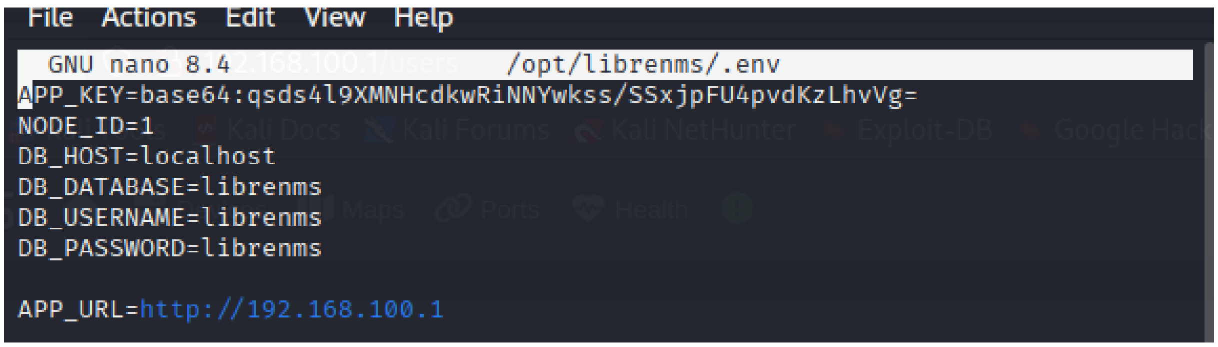
After generating the key, note it, and edit the environment file:

```
sudo nano /opt/librenms/.env
```

Adjust the file content:

```
NODE_ID=1
DB_HOST=localhost
DB_DATABASE=librenms
DB_USERNAME=librenms
DB_PASSWORD=librenms
APP_KEY=base64:YOUR_GENERATED_KEY_HERE
APP_URL=http://192.168.100.1
```

APP_KEY is the key we've just created, APP_URL is the localhost page.



```
  GNU nano 8.4                  /opt/librenms/.env
APP_KEY=base64:qsds4l9XMNHcdkwRiNNYwkss/SSxjpFU4pvdKzLhvVg=
NODE_ID=1
DB_HOST=localhost
DB_DATABASE=librenms
DB_USERNAME=librenms
DB_PASSWORD=librenms

APP_URL=http://192.168.100.1
```

Figure 9: LibreNMS Environment Configuration.

Initialize the database :

```
sudo su - librenms
cd /opt/librenms
php artisan migrate --seed
```

Create the database user:

```
php artisan user:add librenms -p librenms -e 24560025@gm.uit.edu.vn -r admin
```

This add a new user with username: librenms, password: librenms, email: 24560025@gm.uit.edu.vn with the role of an admin of the database.

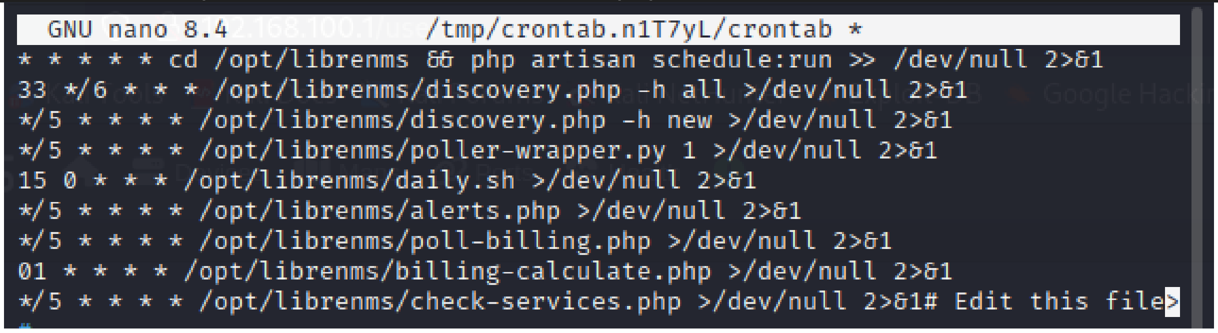
Set up Cron Job:

```
crontab -e
```

Add these lines to the cron configuration file:

```
33 */6 * * * /opt/librenms/discovery.php -h all >/dev/null 2>&1
```

```
*/5 * * * * /opt/librenms/discovery.php -h new >/dev/null 2>&1
*/5 * * * * /opt/librenms/poller-wrapper.py 1 >/dev/null 2>&1
15 0 * * * /opt/librenms/daily.sh >/dev/null 2>&1
*/5 * * * * /opt/librenms/alerts.php >/dev/null 2>&1
*/5 * * * * /opt/librenms/poll-billing.php >/dev/null 2>&1
01 * * * * /opt/librenms/billing-calculate.php >/dev/null 2>&1
*/5 * * * * /opt/librenms/check-services.php >/dev/null 2>&1
```



```
  GNU nano 8.4              /tmp/crontab.n1T7yL/crontab *
* * * * * cd /opt/librenms && php artisan schedule:run >> /dev/null 2>&1
33 */6 * * * /opt/librenms/discovery.php -h all >/dev/null 2>&1
*/5 * * * * /opt/librenms/discovery.php -h new >/dev/null 2>&1
*/5 * * * * /opt/librenms/poller-wrapper.py 1 >/dev/null 2>&1
15 0 * * * /opt/librenms/daily.sh >/dev/null 2>&1
*/5 * * * * /opt/librenms/alerts.php >/dev/null 2>&1
*/5 * * * * /opt/librenms/poll-billing.php >/dev/null 2>&1
01 * * * * /opt/librenms/billing-calculate.php >/dev/null 2>&1
*/5 * * * * /opt/librenms/check-services.php >/dev/null 2>&1# Edit this file>
```

Figure 10: Cron Job Configuration.

Set file with proper permission:

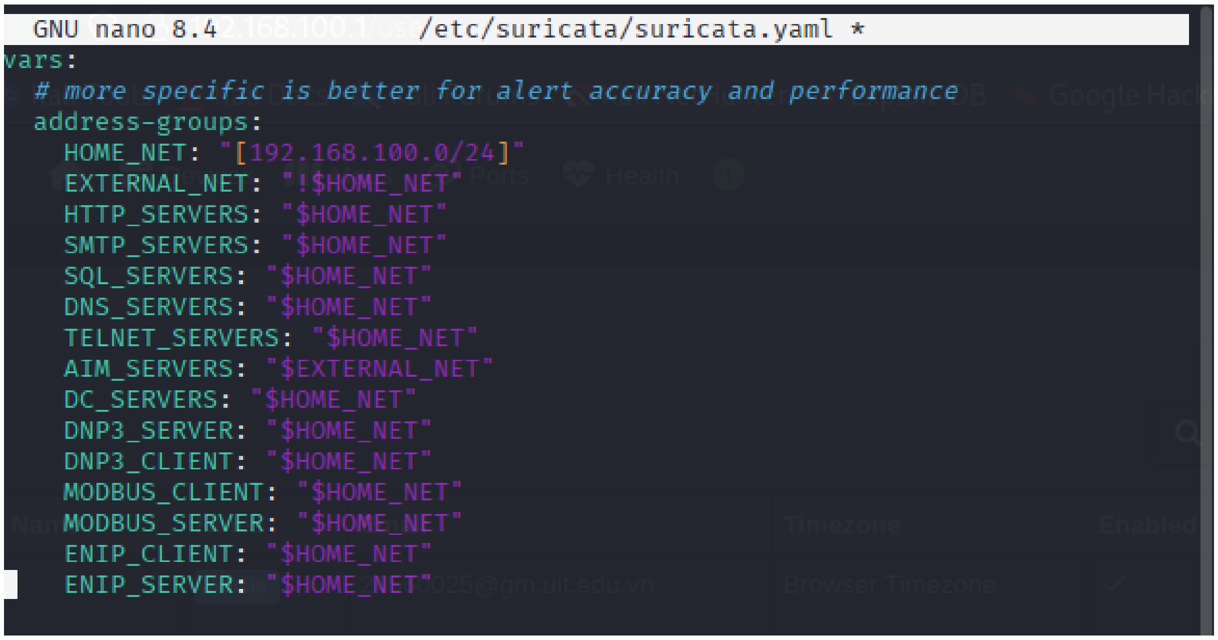
```
sudo chown -R librenms:librenms /opt/librenms
sudo setfacl -d -m g::rwx /opt/librenms/rrd /opt/librenms/logs
/opt/librenms/bootstrap/cache/ /opt/librenms/storage/
sudo setfacl -R -m g::rwx /opt/librenms/rrd /opt/librenms/logs
/opt/librenms/bootstrap/cache/ /opt/librenms/storage/
```

Configure Suricata IDS:

Backup original configuration and edit Suricata configuration:

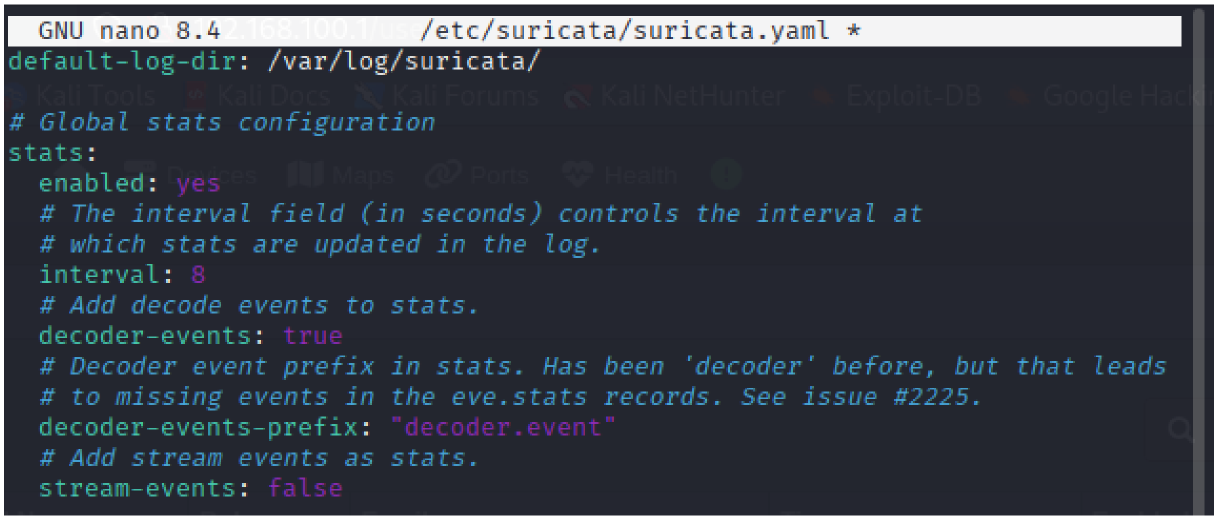
```
sudo cp /etc/suricata/suricata.yaml /etc/suricata/suricata.yaml.backup
sudo nano /etc/suricata/suricata.yaml
```

Edit the file like below:



```
  GNU nano 8.4                /etc/suricata/suricata.yaml *
vars:
  # more specific is better for alert accuracy and performance
  address-groups:
    HOME_NET: "[192.168.100.0/24]"
    EXTERNAL_NET: "!$HOME_NET"
    HTTP_SERVERS: "$HOME_NET"
    SMTP_SERVERS: "$HOME_NET"
    SQL_SERVERS: "$HOME_NET"
    DNS_SERVERS: "$HOME_NET"
    TELNET_SERVERS: "$HOME_NET"
    AIM_SERVERS: "$EXTERNAL_NET"
    DC_SERVERS: "$HOME_NET"
    DNP3_SERVER: "$HOME_NET"
    DNP3_CLIENT: "$HOME_NET"
    MODBUS_CLIENT: "$HOME_NET"
    MODBUS_SERVER: "$HOME_NET"
    ENIP_CLIENT: "$HOME_NET"
    ENIP_SERVER: "$HOME_NET"
```

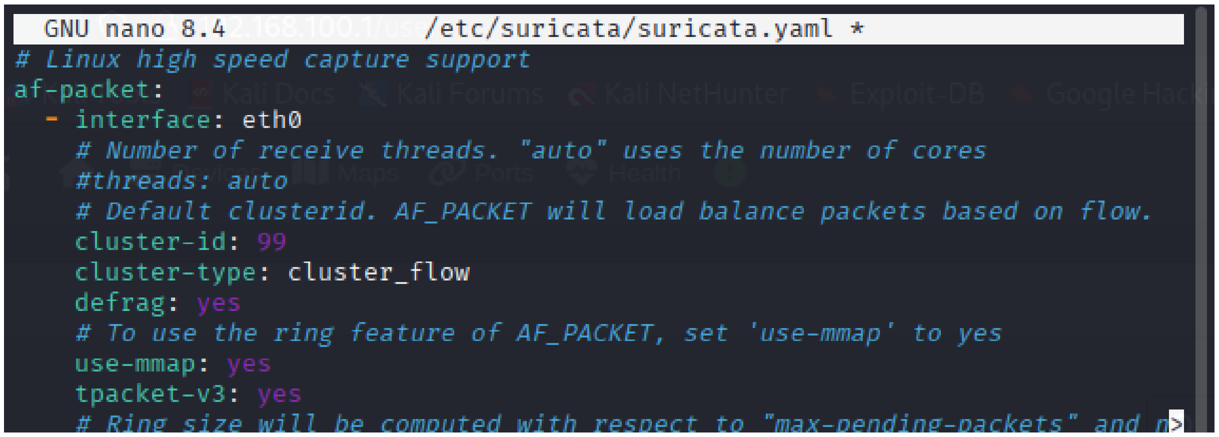
Figure 11: Suricata Configuration.1



```
  GNU nano 8.4                /etc/suricata/suricata.yaml *
default-log-dir: /var/log/suricata/

# Global stats configuration
stats:
  enabled: yes
  # The interval field (in seconds) controls the interval at
  # which stats are updated in the log.
  interval: 8
  # Add decode events to stats.
  decoder-events: true
  # Decoder event prefix in stats. Has been 'decoder' before, but that leads
  # to missing events in the eve.stats records. See issue #2225.
  decoder-events-prefix: "decoder.event"
  # Add stream events as stats.
  stream-events: false
```

Figure 12: Suricata Configuration.2

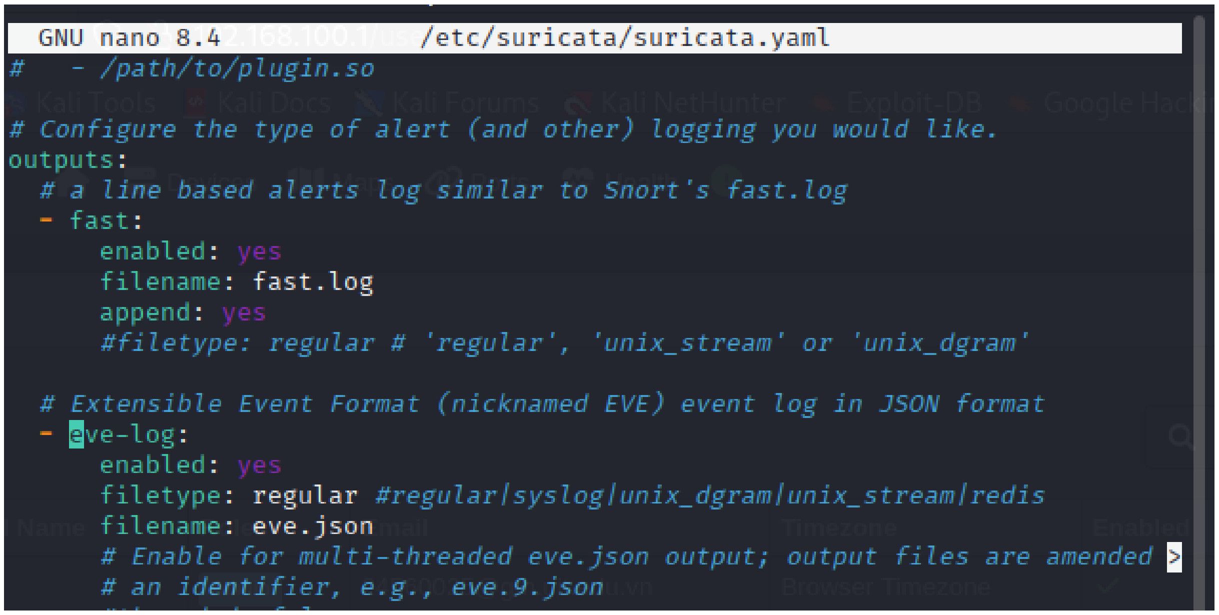
Configure af-packet:



```
  GNU nano 8.4              /etc/suricata/suricata.yaml *
# Linux high speed capture support
af-packet:
  - interface: eth0
    # Number of receive threads. "auto" uses the number of cores
    #threads: auto
    # Default clusterid. AF_PACKET will load balance packets based on flow.
    cluster-id: 99
    cluster-type: cluster_flow
    defrag: yes
    # To use the ring feature of AF_PACKET, set 'use-mmap' to yes
    use-mmap: yes
    tpacket-v3: yes
    # Ring size will be computed with respect to "max-pending-packets" and n>
```

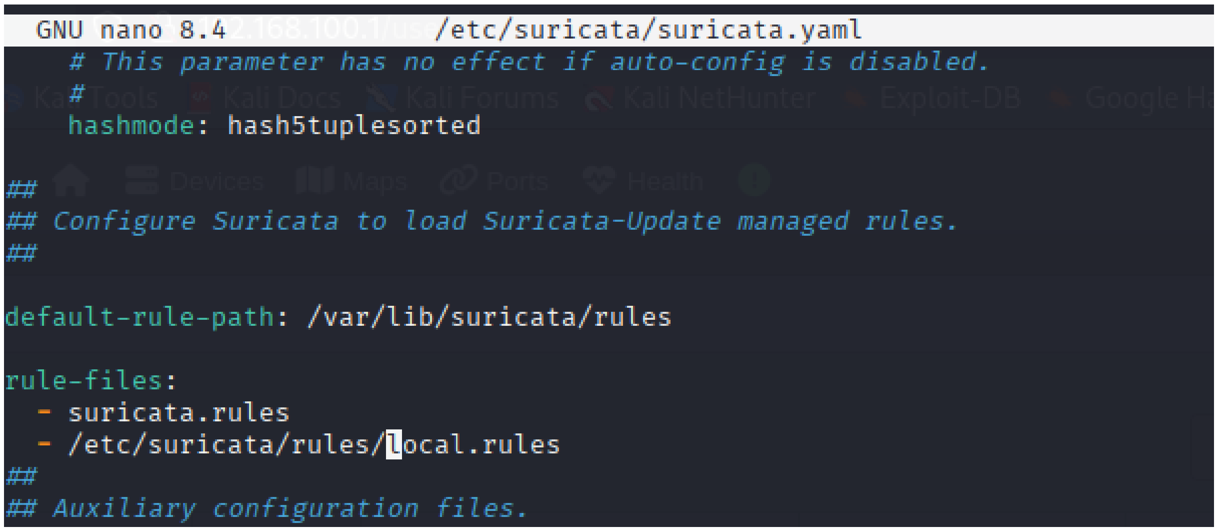
Figure 13: Suricata Configuration.3

Set Suricata output:



```
  GNU nano 8.4              /etc/suricata/suricata.yaml
#   - /path/to/plugin.so

# Configure the type of alert (and other) logging you would like.
outputs:
  # a line based alerts log similar to Snort's fast.log
  - fast:
      enabled: yes
      filename: fast.log
      append: yes
      #filetype: regular # 'regular', 'unix_stream' or 'unix_dgram'

  # Extensible Event Format (nicknamed EVE) event log in JSON format
  - eve-log:
      enabled: yes
      filetype: regular #regular|syslog|unix_dgram|unix_stream|redis
      filename: eve.json
      # Enable for multi-threaded eve.json output; output files are amended >
      # an identifier, e.g., eve.9.json
```

Figure 14: Suricata Configuration.4



```
  GNU nano 8.4                 /etc/suricata/suricata.yaml
    # This parameter has no effect if auto-config is disabled.
    #
    hashmode: hash5tuplesorted

##
## Configure Suricata to load Suricata-Update managed rules.
##

default-rule-path: /var/lib/suricata/rules

rule-files:
  - suricata.rules
  - /etc/suricata/rules/local.rules
##
## Auxiliary configuration files.
```

Figure 15: Suricata Configuration.5

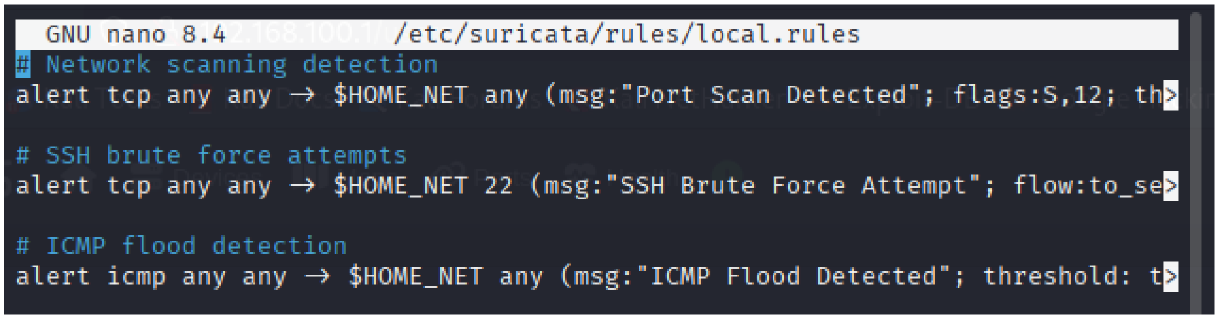
Custom Rules for Attack Detection:

Create custom rules directory and file:

```
sudo mkdir -p /etc/suricata/rules
sudo nano /etc/suricata/rules/local.rules
```

Add these following rules :

```
# Network scanning detection
alert tcp any any -> $HOME_NET any (msg:"Port Scan Detected"; flags:S,12;
threshold: type both, track by_src, count 20, seconds 60; sid:1000001; rev:1;)

# SSH brute force attempts
alert tcp any any -> $HOME_NET 22 (msg:"SSH Brute Force Attempt";
flow:to_server; threshold: type both, track by_src, count 5, seconds 60;
sid:1000002; rev:1;)

# ICMP flood detection
alert icmp any any -> $HOME_NET any (msg:"ICMP Flood Detected"; threshold: type
both, track by_src, count 10, seconds 5; sid:1000003; rev:1;)
```



```
  GNU nano 8.4                /etc/suricata/rules/local.rules
# Network scanning detection
alert tcp any any → $HOME_NET any (msg:"Port Scan Detected"; flags:S,12; th>

# SSH brute force attempts
alert tcp any any → $HOME_NET 22 (msg:"SSH Brute Force Attempt"; flow:to_se>

# ICMP flood detection
alert icmp any any → $HOME_NET any (msg:"ICMP Flood Detected"; threshold: t>
```

Figure 16: Suricata Local Rules

Update the rules and activate Suricata:

```
sudo suricata-update update-sources
sudo suricata-update list-sources
sudo suricata-update
```

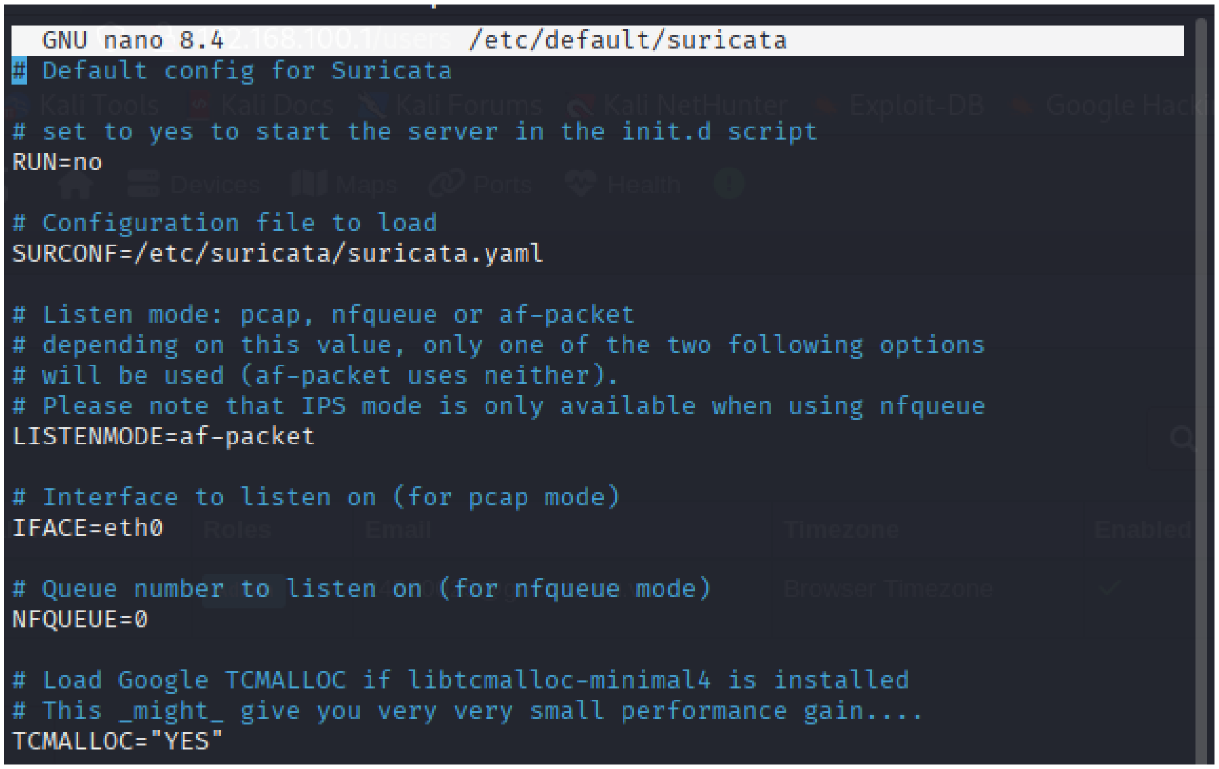
Configure Suricata service:

Issue the command below:

```
sudo nano /etc/default/suricata
```

Adjust the file content :

```
# Interface to listen on
IFACE=eth0

# Additional interfaces (space separated)
LISTENMODE=af-packet
```



```
  GNU nano 8.4                    /etc/default/suricata
# Default config for Suricata

# set to yes to start the server in the init.d script
RUN=no

# Configuration file to load
SURCONF=/etc/suricata/suricata.yaml

# Listen mode: pcap, nfqueue or af-packet
# depending on this value, only one of the two following options
# will be used (af-packet uses neither).
# Please note that IPS mode is only available when using nfqueue
LISTENMODE=af-packet

# Interface to listen on (for pcap mode)
IFACE=eth0

# Queue number to listen on (for nfqueue mode)
NFQUEUE=0

# Load Google TCMALLOC if libtcmalloc-minimal4 is installed
# This _might_ give you very very small performance gain....
TCMALLOC="YES"
```

Figure 17: Suricata Service Configuration.

3. Result

3.1. Basic Scenario:

-In this basic scenario, Suricata is configured to detect both internal and external attacks, and LibreNMS can monitor physical status of devices in Vmnet2 network.

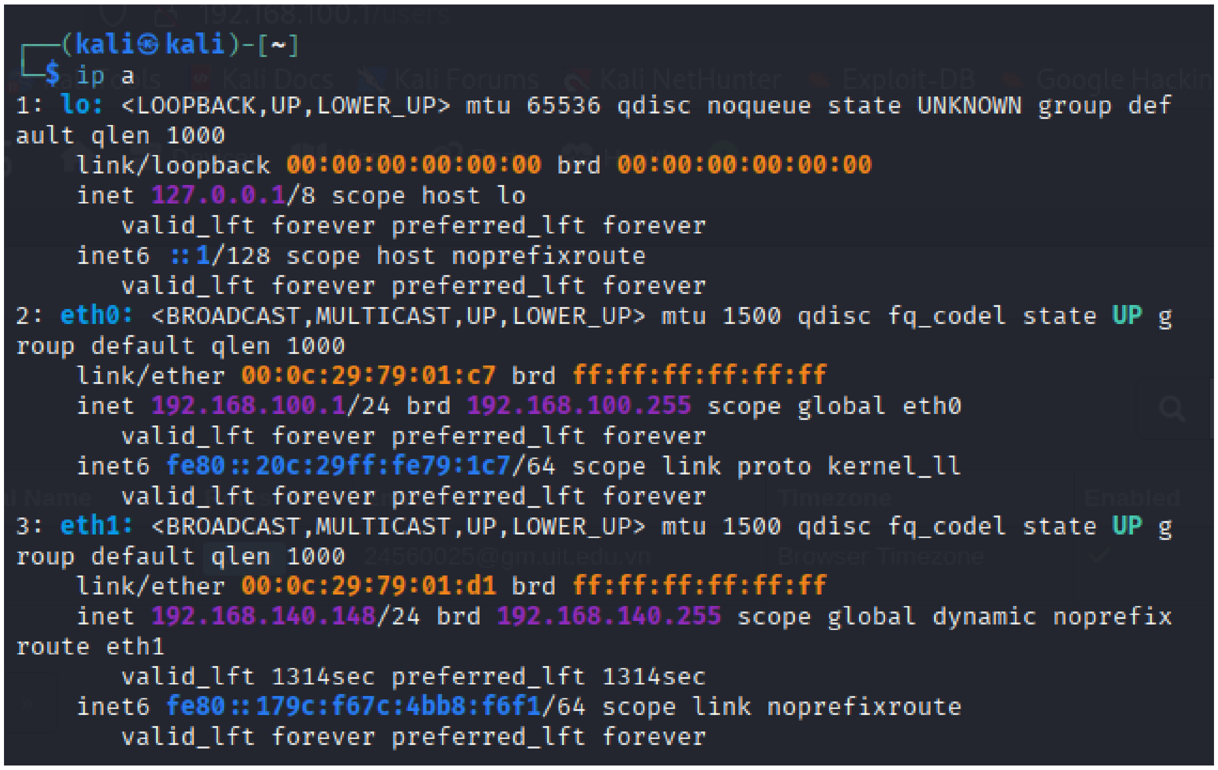
Suricata test:

```
sudo tail -f /var/log/suricata/eve.json | jq 'select(.event_type=="alert")
```

Intrusion can be detected by issuing the command above

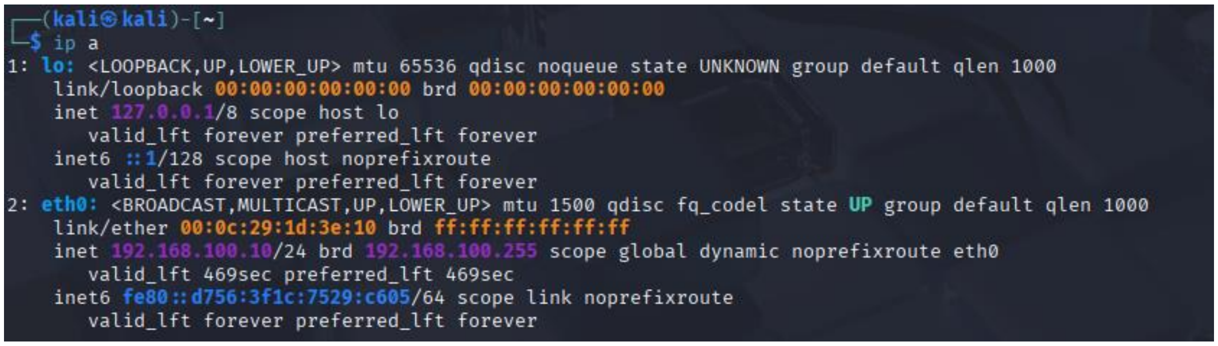
This case I will use the kali-vm to attack client-vm since every traffic must go through the DHCP server, which is kali-vm so the source of the attack is not important.

Let's clarify that kali-vm's IP address is 192.168.100.1, and client-vm's IP address is 192.168.100.10:



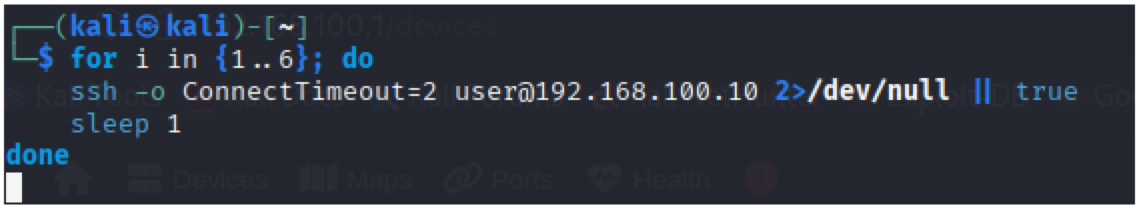
```
┌──(kali㉿kali)-[~]
└─$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group def
ault qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP g
roup default qlen 1000
    link/ether 00:0c:29:79:01:c7 brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.1/24 brd 192.168.100.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe79:1c7/64 scope link proto kernel_ll
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP g
roup default qlen 1000
    link/ether 00:0c:29:79:01:d1 brd ff:ff:ff:ff:ff:ff
    inet 192.168.140.148/24 brd 192.168.140.255 scope global dynamic noprefix
route eth1
       valid_lft 1314sec preferred_lft 1314sec
    inet6 fe80::179c:f67c:4bb8:f6f1/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

Figure 18: Kali-vm IP Address.



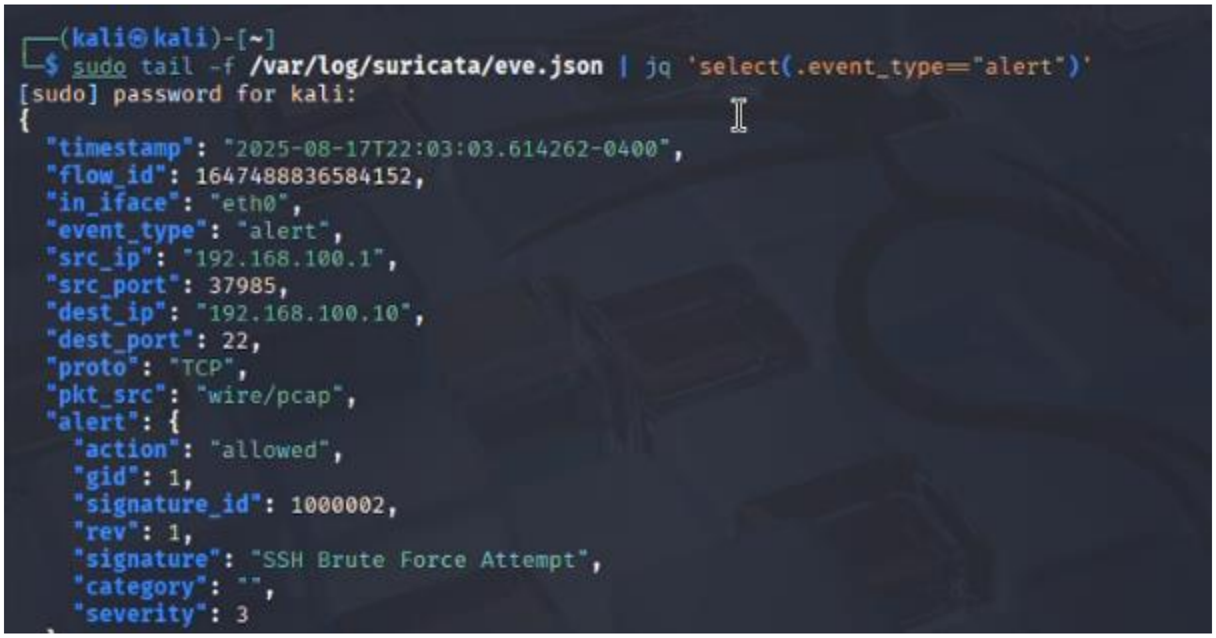
```
┌──(kali㉿kali)-[~]
└─$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       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:29:1d:3e:10 brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.10/24 brd 192.168.100.255 scope global dynamic noprefixroute eth0
       valid_lft 469sec preferred_lft 469sec
    inet6 fe80::d756:3f1c:7529:c605/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

Figure 19: Client-vm IP Address.



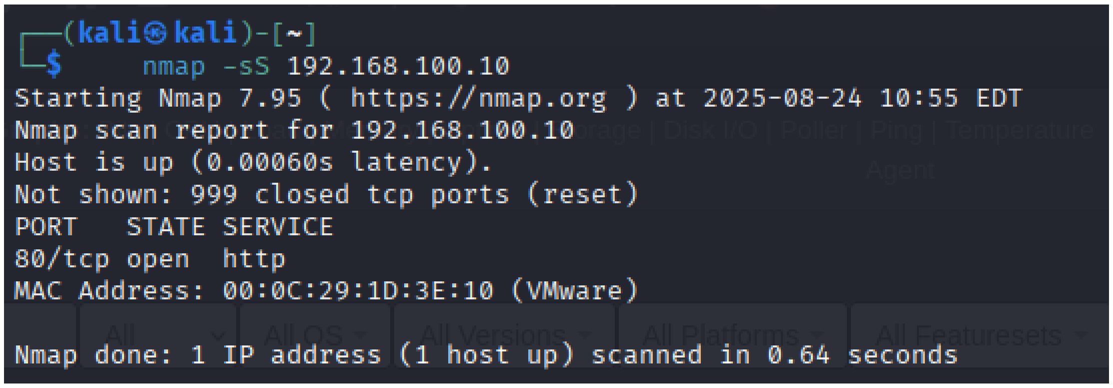
```
┌──(kali㉿kali)-[~]
└─$ for i in {1..6}; do
    ssh -o ConnectTimeout=2 user@192.168.100.10 2>/dev/null || true
    sleep 1
done
```

Figure 20: BruteForce SSH Attack.



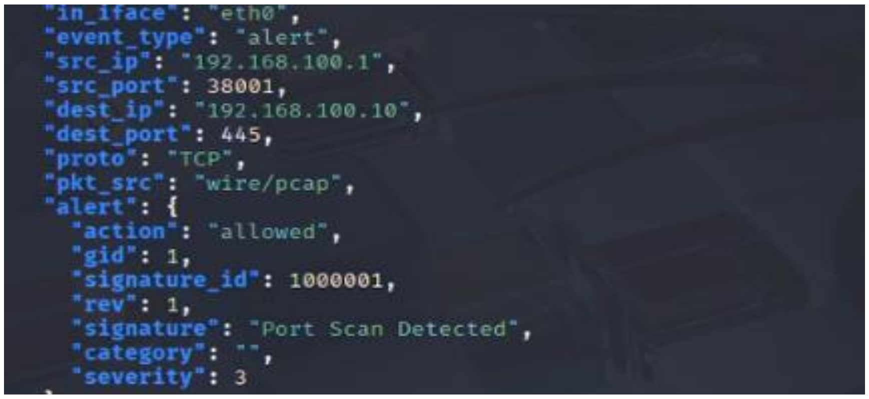
```
┌──(kali㉿kali)-[~]
└─$ sudo tail -f /var/log/suricata/eve.json | jq 'select(.event_type=="alert")'
[sudo] password for kali:
{
  "timestamp": "2025-08-17T22:03:03.614262-0400",
  "flow_id": 1647488836584152,
  "in_iface": "eth0",
  "event_type": "alert",
  "src_ip": "192.168.100.1",
  "src_port": 37985,
  "dest_ip": "192.168.100.10",
  "dest_port": 22,
  "proto": "TCP",
  "pkt_src": "wire/pcap",
  "alert": {
    "action": "allowed",
    "gid": 1,
    "signature_id": 1000002,
    "rev": 1,
    "signature": "SSH Brute Force Attempt",
    "category": "",
    "severity": 3
```

Figure 21: BruteForce SSH Detected.



```
┌──(kali㉿kali)-[~]
└─$     nmap -sS 192.168.100.10
Starting Nmap 7.95 ( https://nmap.org ) at 2025-08-24 10:55 EDT
Nmap scan report for 192.168.100.10
Host is up (0.00060s latency).
Not shown: 999 closed tcp ports (reset)
PORT    STATE SERVICE
80/tcp open  http
MAC Address: 00:0C:29:1D:3E:10 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 0.64 seconds
```

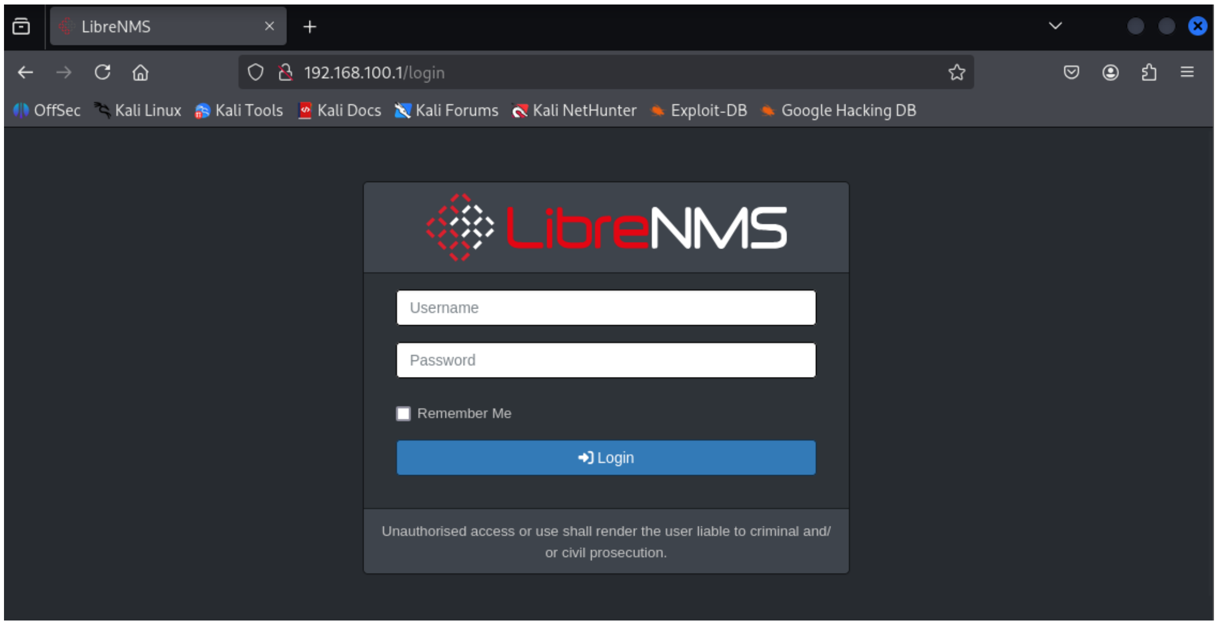
Figure 22: Port Scan Attack.



```
"in_iface": "eth0",
"event_type": "alert",
"src_ip": "192.168.100.1",
"src_port": 38001,
"dest_ip": "192.168.100.10",
"dest_port": 445,
"proto": "TCP",
"pkt_src": "wire/pcap",
"alert": {
  "action": "allowed",
  "gid": 1,
  "signature_id": 1000001,
  "rev": 1,
  "signature": "Port Scan Detected",
  "category": "",
  "severity": 3
```

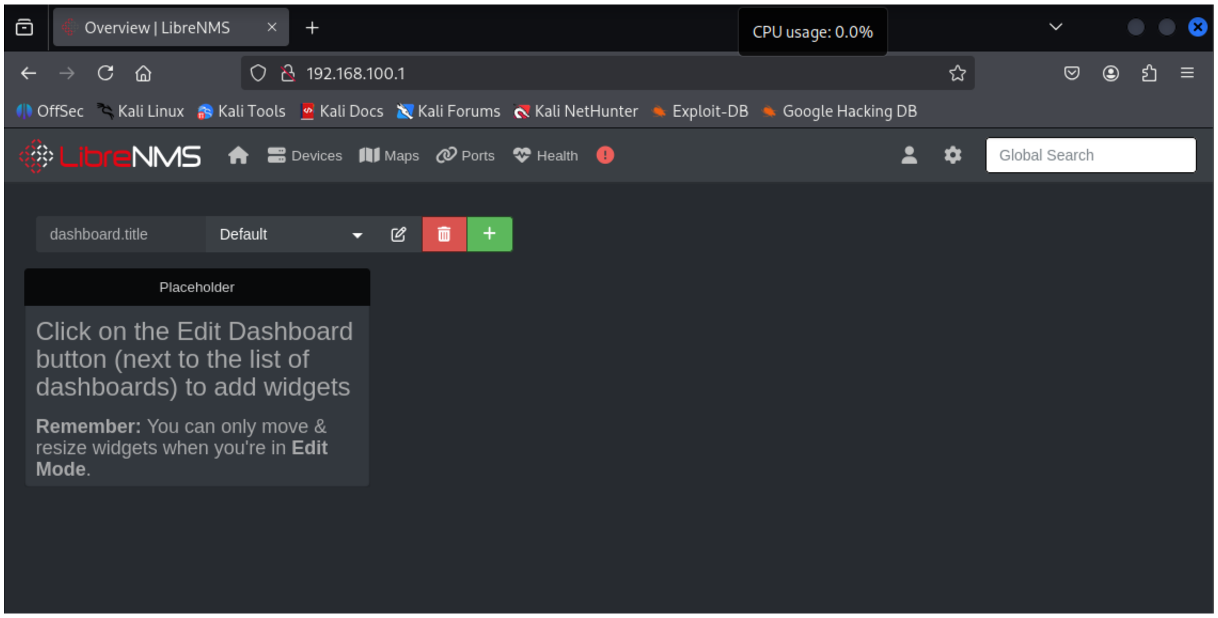
Figure 23: Port Scan Detected.

LibreNMS test:



Unauthorised access or use shall render the user liable to criminal and/ or civil prosecution.

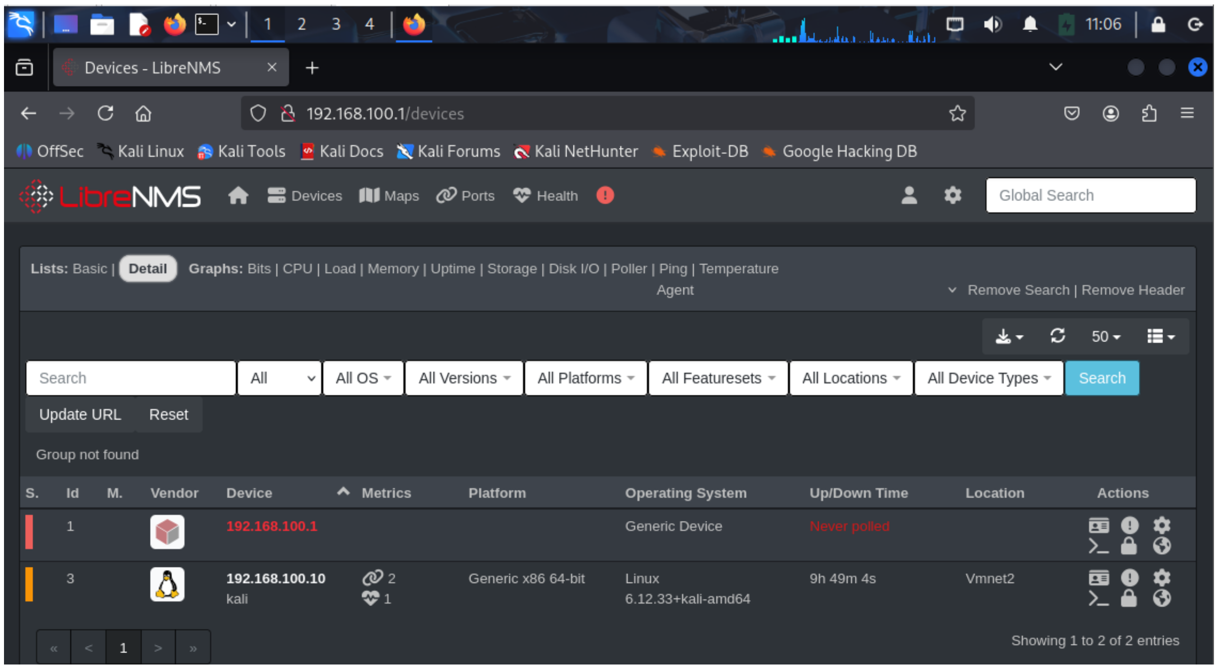
Figure 24: LibreNMS Login Page.



Click on the Edit Dashboard button (next to the list of dashboards) to add widgets

Remember: You can only move & resize widgets when you're in **Edit Mode**.

Figure 25: LibreNMS Home Page.

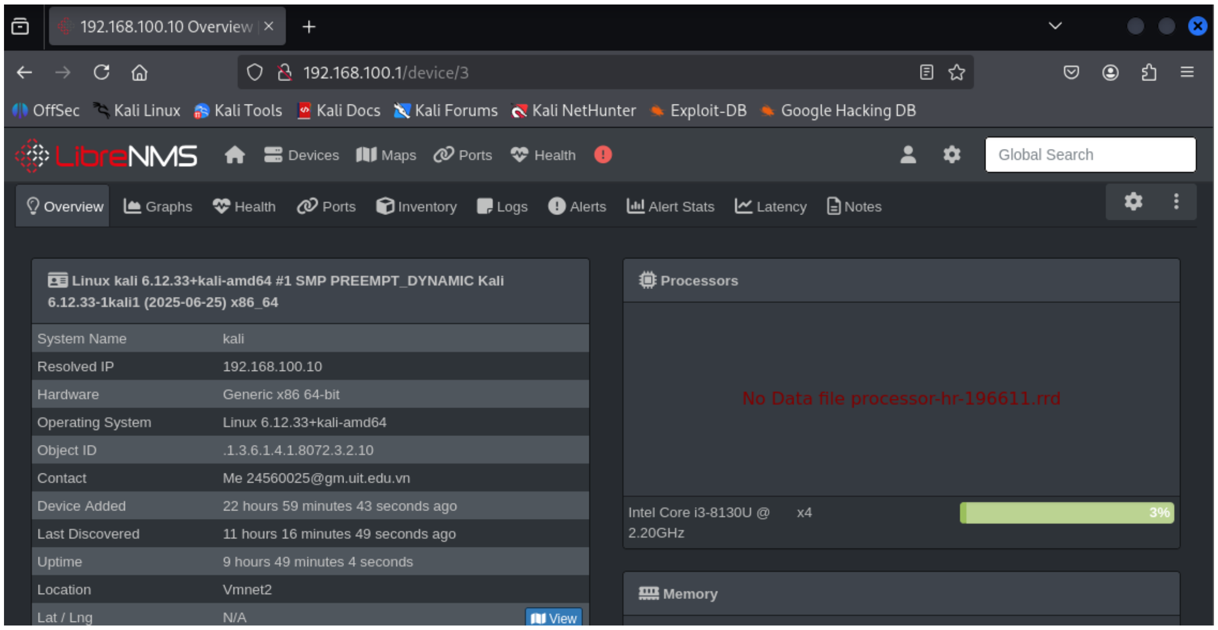


Group not found

S.	Id	M.	Vendor	Device	Metrics	Platform	Operating System	Up/Down Time	Location	Actions
🟥	1			192.168.100.1			Generic Device	Never polled		
🟧	3			192.168.100.10 kali	2 1	Generic x86 64-bit	Linux 6.12.33+kali-amd64	9h 49m 4s	Vmnet2	

Showing 1 to 2 of 2 entries

Figure 26: LibreNMS Device List.



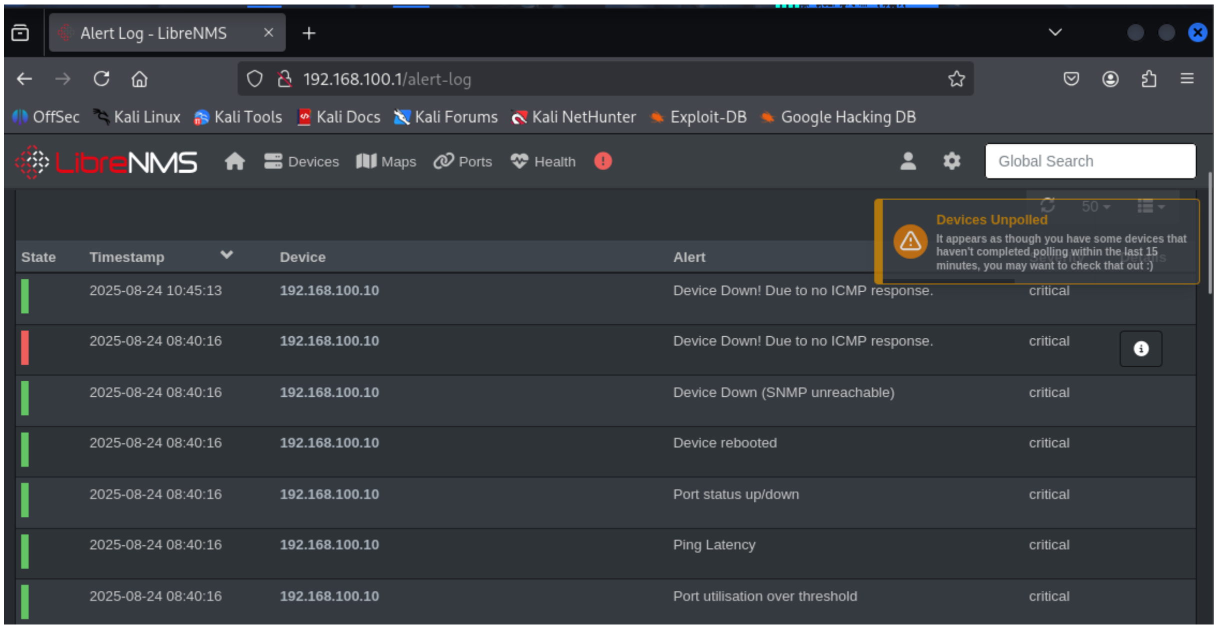
Linux kali 6.12.33+kali-amd64 #1 SMP PREEMPT_DYNAMIC Kali 6.12.33-1kali1 (2025-06-25) x86_64

System Name	kali
Resolved IP	192.168.100.10
Hardware	Generic x86 64-bit
Operating System	Linux 6.12.33+kali-amd64
Object ID	.1.3.6.1.4.1.8072.3.2.10
Contact	Me 24560025@gm.uit.edu.vn
Device Added	22 hours 59 minutes 43 seconds ago
Last Discovered	11 hours 16 minutes 49 seconds ago
Uptime	9 hours 49 minutes 4 seconds
Location	Vmnet2
Lat / Lng	N/A

No Data file processor-hr-196611.rrd

Intel Core i3-8130U @ 2.20GHz x4 3%

Figure 27: LibreNMS Device Information.



Devices Unpolled
It appears as though you have some devices that haven't completed polling within the last 15 minutes, you may want to check that out :)

State	Timestamp	Device	Alert	
🟩	2025-08-24 10:45:13	192.168.100.10	Device Down! Due to no ICMP response.	critical
🟥	2025-08-24 08:40:16	192.168.100.10	Device Down! Due to no ICMP response.	critical
🟩	2025-08-24 08:40:16	192.168.100.10	Device Down (SNMP unreachable)	critical
🟩	2025-08-24 08:40:16	192.168.100.10	Device rebooted	critical
🟩	2025-08-24 08:40:16	192.168.100.10	Port status up/down	critical
🟩	2025-08-24 08:40:16	192.168.100.10	Ping Latency	critical
🟩	2025-08-24 08:40:16	192.168.100.10	Port utilisation over threshold	critical

Figure 28: LibreNMS Alerts.

3.2. Advanced Scenario:

In this Advanced Scenario, Suricata logs should be displayed on LibreNMS Dashboard. Unfortunately, I failed on integrating Suricata and LibreNMS so this part is not available.

4. References

[1] Scarfone, K., & Mell, P. (2007). Guide to Intrusion Detection and Prevention Systems (IDPS). NIST Special Publication 800-94, National Institute of Standards and Technology.

[2] Open Information Security Foundation. (2023). Suricata User Guide. Retrieved from https://suricata.readthedocs.io/

[3] LibreNMS Community. (2023). LibreNMS Documentation. Retrieved from https://docs.librenms.org/

[4] García-Teodoro, P., Díaz-Verdejo, J., Maciá-Fernández, G., & Vázquez, E. (2009). Anomaly-based network intrusion detection: Techniques, systems and challenges. Computers & Security, 28(1-2), 18-28.

[5] Stallings, W. (2017). Network Security Essentials: Applications and Standards (6th ed.). Pearson Education.