Network Security Lab

Building a Lightweight SIEM and IDPS with Suricata, Loki, Promtail, and LogCLI

Learning Objectives

By the end of this lab you will be able to:

1. Explain the core components of a modern SIEM and IDPS.
2. Install and configure **Suricata** to detect and log network events.
3. Configure **Promtail** to collect those logs and send them to **Loki**
4. Query and correlate alerts using **LogCLI**.
5. Reflect on the detection pipeline and incident-response concepts.

Key Concepts & Tools

1. Suricata — IDPS

Purpose: An open-source intrusion detection & prevention system that inspects packets and emits alerts in JSON.
Docs: https://suricata.io/documentation/

You'll use it to generate realistic network-security events for your SIEM.

---

2. Loki — Log Database

Purpose: Grafana's lightweight log aggregation system. It stores logs with labels instead of heavy indexes. This is similar to Elasticsearch but is much lighter.
Docs: https://grafana.com/docs/loki/latest/

Loki is our **SIEM-like backend**. It centralizes and indexes log data for searching and correlation.

---

3. Promtail — Log Shipper

Purpose: A small agent that tails log files and sends them to Loki.
Docs: https://grafana.com/docs/loki/latest/clients/promtail/

Promtail reads Suricata's JSON log (`eve.json`) and forwards it to Loki.

---

4. LogCLI — Query Tool

Purpose: Command-line client for Loki queries, written by Grafana Labs.
Docs: https://grafana.com/docs/loki/latest/tools/logcli/

You'll use it to run searches, extract JSON fields, and perform quick analyses. There is no heavy web UI needed.

5. Docker — Container Platform

Purpose: A lightweight virtualization platform that lets you run applications in isolated **containers**. Each container bundles an app and all its dependencies, so it runs the same way everywhere.
Docs: https://docs.docker.com/

In this lab, Docker is used to run **Loki** and **Promtail** without installing them directly on Ubuntu. It keeps your VM clean and ensures everyone runs the same environment.

Setup and Tasks

Part 1 – Prepare System

To complete this lab, you'll need to install **curl**, **jq**, **unzip**, and **docker**. What each tool does:

curl
- Command-line tool for transferring data via HTTP/S, FTP, etc.
- Used to trigger HTTP requests (for Suricata's custom alert) and to download files like LogCLI.
jq
- Lightweight JSON processor.
- Used to pretty-print and filter Suricata's `eve.json` logs — for example:
`tail -f /var/log/suricata/eve.json | jq .`
unzip
- Utility to extract `.zip` archives.
- Needed because the LogCLI binary you download comes packaged as a zip file.

1. To install curl, jq and unzip, run the following:

```
sudo apt update && sudo apt upgrade -y
sudo apt -y install curl jq unzip
```

2. To install docker:

```
curl -fsSL https://get.docker.com | sudo sh
```

3. The next command adds you to the docker group which allows you to run docker without sudo. The second command starts a fresh shell with the new group, docker.

```
sudo usermod -aG docker "$USER"
newgrp docker
```

4. Enable and start the docker service (some images need this).

- **Note**: `systemd` is an init system and system manager that has widely become the new standard for Linux distributions. The following article goes in depth with systemctl if interested: https://www.digitalocean.com/community/tutorials/how-to-use-systemctl-to-manage-systemd-services-and-units

```
sudo systemctl enable --now docker
docker --version
```

Part 2 – Suricata

Suricata is a high performance Network IDS, IPS and Network Security Monitoring engine. To better understand the purpose of Suricata, read the following article before continuing: https://medium.com/@redfanatic7/suricata-vs-snort-detailed-guide-to-the-programs-c331cff452a1

In your lab: After reading the article, **in your own words**, add a brief description of what you learned.

Preflight Setup for Suricata

Before creating your custom rules, you need to make sure Suricata has a valid set of default rules and the correct file paths configured. This setup step prevents startup errors by downloading the community rule set, creating a local rules file, and aligning Suricata's configuration so it can load and apply both sets of rules successfully.

The following commands download the community rule set:

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

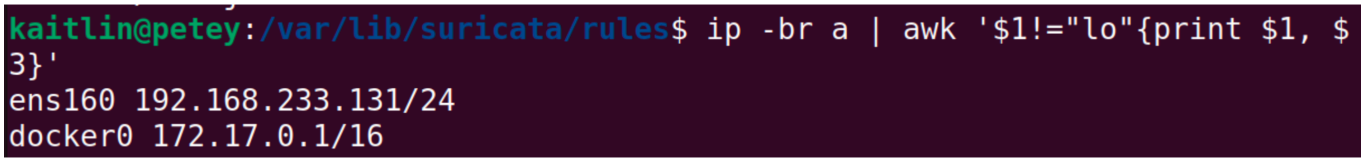
Check out `/var/lib/suricata/rules/suricata.rules`. This is the **main ruleset file** that gets downloaded and managed by the **Suricata Update utility**. This file contains thousands of prewritten signatures for:

- Exploit attempts
- Malware traffic
- Network scans
- Common vulnerabilities
- Suspicious protocols or payloads

We will need our interface name for later. To find this, enter the command:

```
ip -br a | awk '$1!="lo"{print $1, $3}'
```

Your interface name will most likely start with an e. Mine is **ens160**:



```
kaitlin@petey:/var/lib/suricata/rules$ ip -br a | awk '$1!="lo"{print $1, $
3}'
ens160 192.168.233.131/24
docker0 172.17.0.1/16
```

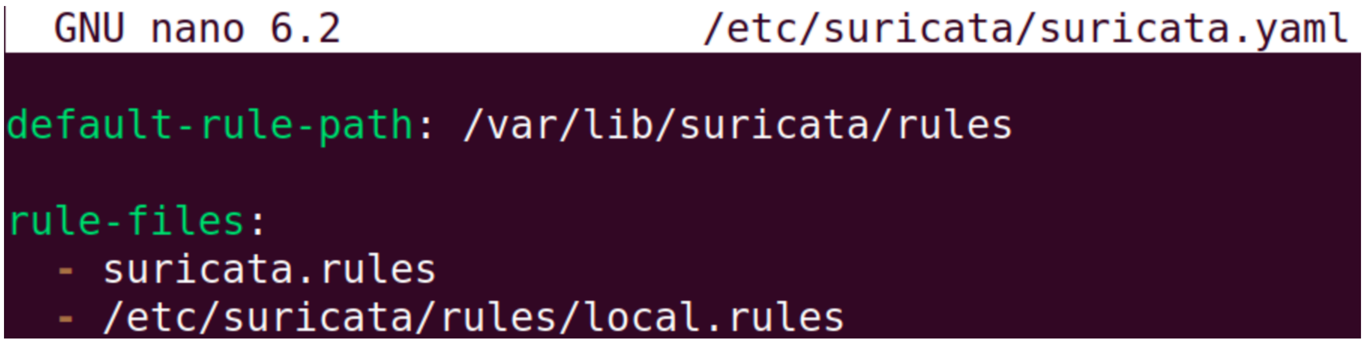
Next, we'll create a directory and file for our custom rules:

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

We now need to update the **suricata.yaml** file to the correct rule path (yaml is a **human-readable data format** used to define configuration settings similar to JSON or XML). Open the file:

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

Then, search for **default-rule-path** using **control + w** in nano. Update the rule path to **/var/lib/suricata/rules** and add **/etc/suricata/rules/local.rules** to rule-files:

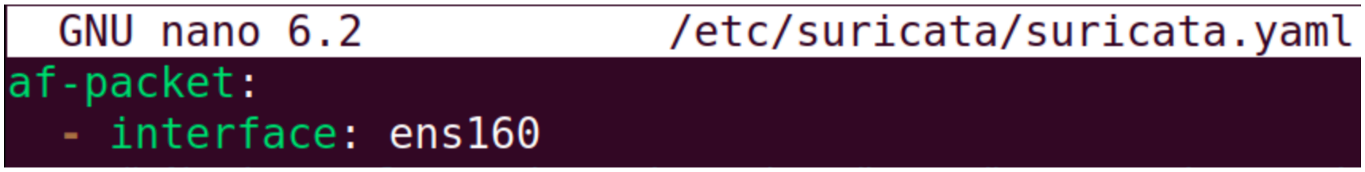


```
  GNU nano 6.2                    /etc/suricata/suricata.yaml

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

rule-files:
  - suricata.rules
  - /etc/suricata/rules/local.rules
```

Now search for **af-packet** and replace value with your interface name from above:



```
  GNU nano 6.2                   /etc/suricata/suricata.yaml
af-packet:
  - interface: ens160
```

Enter the following command to test and validate your Suricata connection:

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

Explain what the -T, -c and -v flags do in the command above. You can use the man command to read the manual on Suricata to answer this question:

```
man suricata
```

Run Suricata

Run the following commands. The first command stops the default service so you can control it manually, and the second starts Suricata in the background on your VM's primary network interface, allowing it to monitor network traffic and generate alerts.

```
sudo systemctl stop suricata
sudo suricata -i $(ip -br a | awk '$1!="lo"{print $1; exit}') -D
```

Confirm it writes logs. To exit, hit **control + c**:

```
sudo tail -f /var/log/suricata/eve.json | jq .
```

An important Suricata file is `eve.json`. This is Suricata's **main log file**, and it records all events detected by the IDS/IPS. It's written in JSON (JavaScript Object Notation) format so other tools (like Promtail, Loki, or SIEMs) can easily parse and analyze it. Each line in `eve.json` is a separate JSON object describing a single event.

You can find eve.json under the /var/log folder. The following command will make reading the file much easier by separating each object:

```
sudo jq . /var/log/suricata/eve.json
```

Question 1: What types of events (fields under `"event_type"`) do you see in `eve.json`?

Part 3 - Loki

Loki acts as the **central log database** in your SIEM setup.
It receives logs, like Suricata's eve.json, from agents such as **Promtail**, stores them efficiently, and lets you search and analyze them using simple queries.

Unlike traditional log systems that index all text, which uses lots of resources, Loki indexes only **metadata labels**, such as `job`, `host`, or `filename`.

You're using Loki to:

- Store Suricata's network and alert logs
- Enable fast searching and filtering (using **LogCLI**)
- Demonstrate how a SIEM collects and analyzes security events

Before continuing, read the following article: https://medium.com/@gpiechnik/loki-effective-logging-and-log-aggregation-with-grafana-c3356e7f13ad

In your lab: After reading the article, **in your own words**, add a brief description of what you learned.

Preflight Setup for Loki

Before running Loki, we need to make sure its configuration file and data directories exist and have the correct permissions. If these steps are skipped, the container will exit with code 1 because it can't find `/etc/loki/loki-config.yml` or write to its data directory.

1. Create directories for Loki:
    - **/etc/loki**: where Loki's **configuration file** will live.
    - **/var/lib/loki**: Loki's **data directory**. Used for storing log chunks and index files.
    - **{chunks,rules}**: creates two subdirectories:
        - **chunks/**: where Loki stores compressed log data.

  - **rules/**: where Loki stores alerting or recording rules (not used in this lab but required by the config).
- The `-p` flag ensures that missing directories are created without errors.

```
sudo mkdir -p /etc/loki /var/lib/loki/{chunks,rules}
```

2. Create a default Loki configuration file:

```
# Create a default Loki config file
cat <<'EOF' | sudo tee /etc/loki/loki-config.yml
auth_enabled: false
server:
  http_listen_port: 3100
common:
  path_prefix: /var/lib/loki
  storage:
    filesystem:
      chunks_directory: /var/lib/loki/chunks
      rules_directory: /var/lib/loki/rules
  replication_factor: 1
  ring:
    kvstore:
      store: inmemory
schema_config:
  configs:
    - from: 2020-10-24
      store: boltdb-shipper
      object_store: filesystem
      schema: v13
      index:
        prefix: index_
        period: 24h
EOF
```

To break this down:

- **auth_enabled: false** - turns off authentication for simplicity.
- **server.http_listen_port: 3100** - tells Loki to listen on port **3100** for incoming connections.
- **common.path_prefix** and **storage.filesystem** - define where Loki should store logs (/var/lib/loki/chunks and /var/lib/loki/rules).
- **schema_config**: tells Loki how to structure and index the stored logs.

In short: This YAML file is Loki's instruction manual. It defines how it runs and where it saves data.

3. Fix permissions so Loki can write data. Give Loki's container user (UID 10001) permission to write to /var/lib/loki:

```
sudo chown -R 10001:10001 /var/lib/loki
sudo chmod -R u+rwX /var/lib/loki
```

Run Loki

4. Once setup is complete, start Loki in Docker:

```
sudo docker run -d --name loki -p 3100:3100 \
  -v /etc/loki:/etc/loki \
  -v /var/lib/loki:/var/lib/loki \
  grafana/loki:2.9.8 -config.file=/etc/loki/loki-config.yml
```

To break this down:

- **-d**: runs in detached (background) mode.
- **--name loki**: names the container "loki" for easy reference.
- **-p 3100:3100**: maps port **3100** from the container to your VM, so you can reach Loki at `http://localhost:3100`.
- **-v /etc/loki:/etc/loki**: mounts your host's config directory into the container.
- **-v /var/lib/loki:/var/lib/loki**: mounts the data directory so logs persist even if the container restarts.
- **grafana/loki:2.9.8**: specifies the Loki image and version.

5. Check that Loki is running and ready. Should get output "Ready" if no issues.

```
docker ps
curl -s http://localhost:3100/ready; echo
```

Question 2: What port does Loki expose, and what API path receives log data?

Part 4 – Run Promtail (Log Shipper)

Promtail is an agent that collects logs from your system and **sends them to Loki**. It's designed to work hand-in-hand with Loki.

Promtail's main job is to **find**, **read**, and **forward log files**; In this lab, that's Suricata's eve.json. It attaches helpful **labels**, such as `job="suricata"`, to each log line before sending it to Loki. These labels make your logs easier to filter, group, and search later.

Promtail also keeps track of which parts of a file it has already read using a small positions file, so if you restart it, it won't resend old logs or miss new ones.

Read the following documentation page for Promtail: https://grafana.com/docs/loki/latest/send-data/promtail/

OPTIONAL: Promtail is deprecated and will be EOL March 2026. If interested, the following explains how to migrate from Promtail to Alloy: https://grafana.com/docs/alloy/latest/set-up/migrate/from-promtail/

---

1. Create Promtail folders:
    - **/etc/promtail**: where we'll store Promtail's **config** file.
    - **/var/lib/promtail**: where Promtail keeps its positions file. This is a bookmark so it knows how far it has read in each log and doesn't resend lines as mentioned above.

```
sudo mkdir -p /etc/promtail /var/lib/promtail
```

2. Write the Promtail config file:

```
cat <<'EOF' | sudo tee /etc/promtail/promtail-config.yml
server:
  http_listen_port: 9080
  grpc_listen_port: 0
clients:
  - url: http://localhost:3100/loki/api/v1/push
positions:
  filename: /var/lib/promtail/positions.yaml
scrape_configs:
  - job_name: suricata
    static_configs:
      - targets: [localhost]
        labels:
          job: suricata
          __path__: /var/log/suricata/eve.json
EOF
```

To break this down:

- **server.http_listen_port: 9080**: small local status server (Promtail's own HTTP endpoint).
    - **clients.url**: where Promtail sends logs — Loki's HTTP push API on `localhost:3100`.
    - **positions.filename**: the file that stores read offsets (prevents duplicates and missed lines).
    - **scrape_configs**: what to read and how to label it.
        - **job_name: suricata**: a logical name for this scrape job.
        - **static_configs**: we're tailing a known file path.
            - **labels.job: suricata**: adds a queryable label (`{job="suricata"}`).
            - **labels.path: /var/log/suricata/eve.json**: **the file to tail** (Promtail reads this path).

3. Run the Promtail container:

```
sudo docker run -d --name promtail -p 9080:9080 \
  -v /etc/promtail:/etc/promtail \
  -v /var/log/suricata:/var/log/suricata:ro \
  -v /var/lib/promtail:/var/lib/promtail \
  grafana/promtail:2.9.8 \
  -config.file=/etc/promtail/promtail-config.yml
```

To break this down:

- **-d**: run in background (detached).
- **--name promtail**: easy to reference later (`docker logs promtail`).
- **-p 9080:9080**: maps container's port **9080** to the VM's **9080** (Promtail's HTTP status page).
- **-v /etc/promtail:/etc/promtail**: mount your config into the container.
- **-v /var/log/suricata:/var/log/suricata:ro**: mount Suricata logs read-only (:ro) so Promtail can read `eve.json`.
- **-v /var/lib/promtail:/var/lib/promtail**: mount the positions storage so progress persists across restarts.
- **grafana/promtail:2.9.8**: the Promtail image and version.
- **-config.file=...**: tell Promtail where the config is inside the container (the mounted path).

In your own words:
Question 3: What role does Promtail play compared to Loki?
Question 4: Why does Promtail track a "position file"? What problem does it solve?

Part 5 – Install LogCLI and Test Queries

LogCLI is Grafana Loki's command-line tool. It lets you **query**, **view**, and **analyze** logs directly from your terminal. There is no web dashboard needed.

You'll use it to verify that Promtail is successfully sending Suricata logs into Loki.

1. Download and install LogCLI:

```
curl -L https://github.com/grafana/loki/releases/download/v2.9.8/logcli-
linux-arm64.zip -o /tmp/logcli.zip
sudo unzip -o /tmp/logcli.zip -d /usr/local/bin
sudo mv /usr/local/bin/logcli-linux-arm64 /usr/local/bin/logcli
sudo chmod +x /usr/local/bin/logcli
logcli --version
```

Explain in your own words what the above commands are doing.

2. **Verify connectivity and list available labels**. The following command tests if LogCLI can connect to the Loki server running on `localhost:3100`. It Lists all the **labels** Loki has seen so far.

```
logcli labels --addr=http://localhost:3100
```

3. **Query recent logs**. Explain in your own words what the command below is doing.

```
logcli query --addr=http://localhost:3100 --limit=10 '{job="suricata"}'
```

Question 5: What labels do you see attached to your logs?
Question 6: How do labels differ from full-text indexes?

Part 6 – Generate Alerts and Analyze

Now that everything is running (Suricata, Promtail, Loki and LogCLI), you'll trigger a test alert to verify that your SIEM pipeline works from end to end.

1. Add a custom Suricata rule to guarantee an alert:

```
echo 'alert http any any -> any any (msg:"LAB UA hit"; http.user_agent;
content:"CPS-NETSEC-LAB"; sid:9900001; rev:1;)' \
  | sudo tee -a /etc/suricata/rules/local.rules
```

- **echo 'alert ...'**: creates a custom Suricata detection rule that matches specific HTTP requests. The rule means:
  "If any HTTP request (any source/destination) contains the User-Agent string `CPS-NETSEC-LAB`, generate an alert named 'LAB UA hit.'"
- **sid:9900001**: gives the rule a unique **signature ID** (used to identify alerts).
- **rev:1**: the rule's revision number (used if you edit the rule later).
- **sudo tee -a**: appends this rule to `/etc/suricata/rules/local.rules`, the file for your custom or local rules.

2. Restart Suricata to load the new rule
   - Suricata must reload its rules to recognize the one you just added.
   - Restarting the service applies all rule changes immediately.

```
sudo systemctl restart suricata
```

If Suricata fails to start, check the rule syntax with:

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

3. Trigger the alert:

```
curl -A "CPS-NETSEC-LAB" http://example.com/ || true
```

- This command sends an HTTP request with the User-Agent header `"CPS-NETSEC-LAB"`.
- Because that header matches your custom rule, Suricata will immediately generate an alert in `eve.json`.

4. Query alerts in Loki:

```
logcli query --addr=http://localhost:3100 --limit=50 \
  '{job="suricata"} |= "event_type\":\"alert\"" | json | line_format "
{{.alert.signature}}"'
```

Question 7: What is the command above doing?
Question 8: What alert message appears?

Part 7 – Correlation Challenge

Find top source IPs in the last 5 minutes:

```
logcli query --addr=http://localhost:3100 --limit=1000 --since=5m \
  '{job="suricata"} |= "event_type\":\"alert\"" | json | line_format "
{{.src_ip}} "' \
  | sort | uniq -c | sort -nr | head
```

Question 9: What does this simple command illustrate about correlation and aggregation in SIEMs?
Question 10: How might a SOC use this information in an investigation?

Part 8 – Create and Test Your Own Custom Rule

Write a custom Suricata rule that detects a specific pattern or condition different from the one in this lab, then trigger it and verify it appears in your logs and in Loki.

Take screenshots of each step and answer the following questions:

1. What condition did your rule detect?
2. How did you test and confirm that it triggered correctly?
3. How would you modify your rule to make it more specific (to reduce false positives)?
4. Why is fine-tuning rules important in real-world intrusion detection?

Part 9 – Cleanup

When you are done, stop and remove your docker containers:

```
sudo docker stop promtail loki
sudo docker rm promtail loki
sudo apt purge -y suricata
sudo docker system prune -a -f
```

Submission

Your lab should include the following components below. Add your lab to your GitHub repo, and submit the link. Be sure to add the lab to your README table of contents. Everything should be in your own words and include references where applicable.

1. A brief introduction describing the purpose of your lab

2. Screenshots of all work
3. Answers to all questions
4. A brief summary at the end of your lab

References & Sources

Tool	Official Documentation	Additional Reading
Suricata	https://suricata.io/documentation/	OISF Docs Configuration Guide
Grafana Loki	https://grafana.com/docs/loki/latest/	Loki Architecture
Promtail	https://grafana.com/docs/loki/latest/clients/promtail/	Promtail Configuration Reference
LogCLI	https://grafana.com/docs/loki/latest/tools/logcli/	Query Examples
cURL	https://curl.se/docs/manpage.html	Used to trigger HTTP traffic for Suricata
Docker	https://docs.docker.com/	Container runtime for Loki/Promtail