Abstract

This report describes the design, deployment, configuration, and evaluation of an Intrusion Detection System (IDS) deployed in a VMware-based virtual lab. The lab uses Suricata as a Network-based IDS (NIDS) on an Ubuntu sensor VM to monitor traffic between an attacker VM (Kali Linux) and a target VM (Windows Computers). The objectives are to demonstrate IDS capabilities in detecting common attack patterns, present the steps required to install and tune Suricata in a virtual environment, and to document a reproducible simulation with expected outputs and analysis. The report concludes with recommendations for best practices in deployment and future improvements.

Table of Contents

1. Introduction

Virtualization platforms such as VMware are widely used to create reproducible and isolated environments for learning, testing, and demonstration of network security tools. An IDS deployed in a virtual environment offers a controlled way to observe attack behaviors and to tune detections without risking production networks. This report documents a lab in which Suricata is used as the NIDS to monitor traffic between a Kali attacker VM and a Windows target VM. The focus is practical**: installing Suricata**, **configuring capture methods within VMware**, **writing and tuning rules**, **running controlled attacks**, **collecting evidence**, and **interpreting alerts**.

2. Objectives

- Design a virtual lab topology in VMware suitable for IDS demonstration and testing.
- Install and configure Suricata on an Ubuntu VM and enable EVE JSON logging for easy analysis.
- Demonstrate detection of the following common attack behaviors: port scanning, exploit attempt (simulated), suspicious HTTP file download.
- Demonstrate attack scenarios and document expected alerts and evidence collection.

3. Background and Terminology

Intrusion Detection System (IDS): A tool that monitors network or host activity to detect suspicious behavior. IDSs can be signature-based (matching known patterns) or anomaly-based (detecting deviations from baseline).

Network-based IDS (NIDS): Monitors network traffic for signs of malicious behavior. Examples: Suricata, Snort.

Host-based IDS (HIDS): Monitors events on a single host (file changes, process activity, logs). Examples: OSSEC, Wazuh.
Suricata: A modern, high-performance IDS/IPS engine supporting multi-threading, protocol parsing, and rich logging (EVE JSON). Chosen for this lab due to its active development, native JSON output, and performance features.

False Positive / False Negative: False positive — benign activity triggers an alert; false negative — malicious activity goes undetected. A central part of IDS deployment is tuning rules to reduce false positives while retaining detection coverage.

Promiscuous Mode / Port Mirroring / SPAN: Techniques used to capture traffic in a virtual environment. Promiscuous mode allows a VM's virtual NIC to receive all traffic on a virtual switch. Port mirroring/SPAN duplicates packets from source ports to a monitoring port.

4. System Architecture and Design Choices

4.1. High-level Architecture

The lab consists of three virtual machines running on a single VMware host (Workstation Pro):

- **Kali Linux (Attacker)** it performs reconnaissance and simulated attacks.
- **Windows Computers (Target)** is the vulnerable or victim host for demonstrations.
- **GNS3 Server (Target)** is the vulnerable or victim host for demonstrations.
- **Ubuntu 24.04 LTS (Suricata sensor)** it runs Suricata in NIDS mode and collects logs.

A small management workstation (host OS) runs VMware. The network between VMs is isolated to avoid unintended impacts on the physical LAN.

4.2. Sensor Placement Decisions

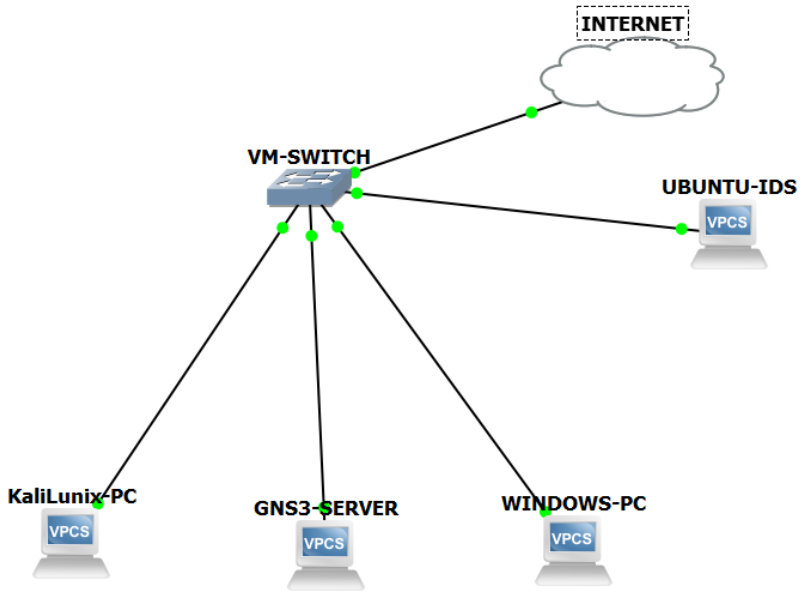
- **NIDS Sensor VM (Suricata)** placed on the same virtual network as attacker and target, and configured to capture mirrored traffic. This provides visibility into inter-VM traffic without instrumenting the target.
- **HIDS (Optional)**: Installing a host-based agent (e.g., Sysmon + Windows Event Forwarding or a Wazuh agent) on the Windows Server enhances detection of post-exploitation behaviors and provides host context.

4.3. Capture Mechanisms Compared

- **Host-only / Internal Networking:** Good for isolated labs; simplest to configure. Can be combined with promiscuous mode on the sensor interface.
- **Bridged Networking:** Connects VMs to the physical LAN; useful when testing interaction with external services but not recommended for class demos due to safety and noise.
- **Port Mirroring / SPAN (ESXi):** Best practice in production-like environments because it replicates exact traffic to the sensor VM. Requires vSwitch configuration on ESXi.

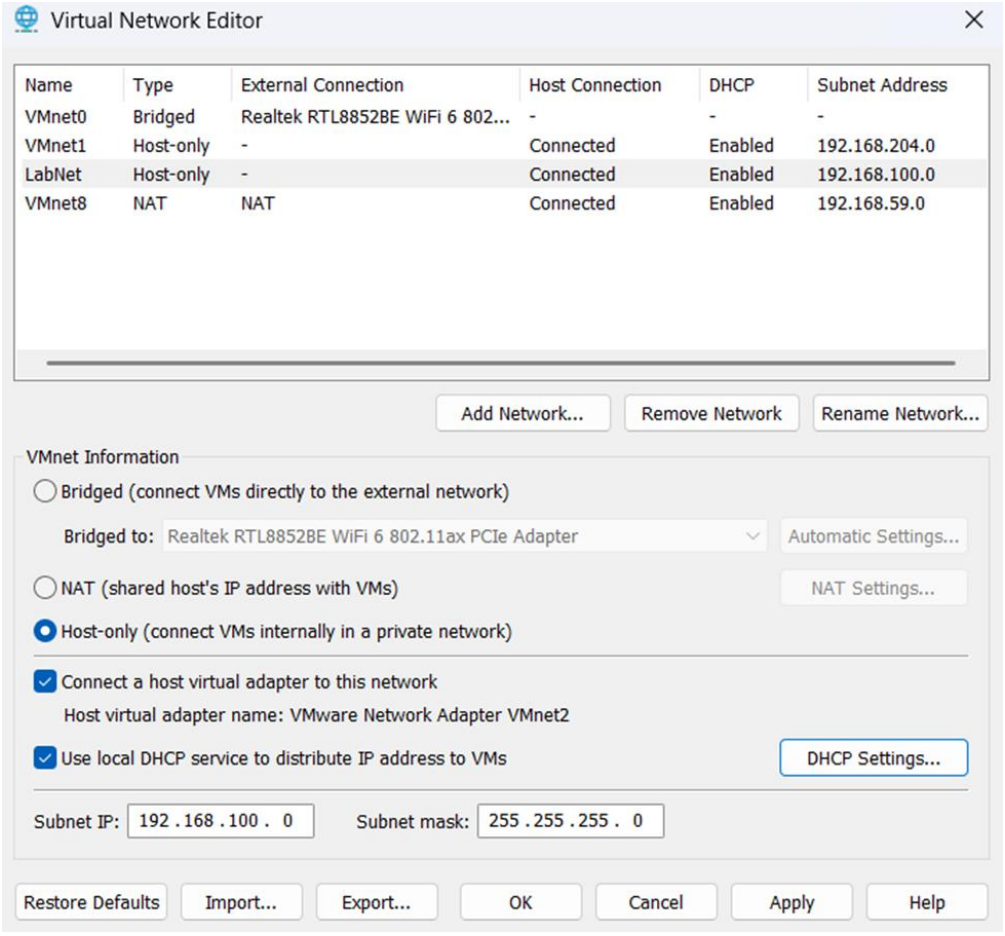
5. VMware Lab Topology and Networking Options

5.1. Topology Diagram

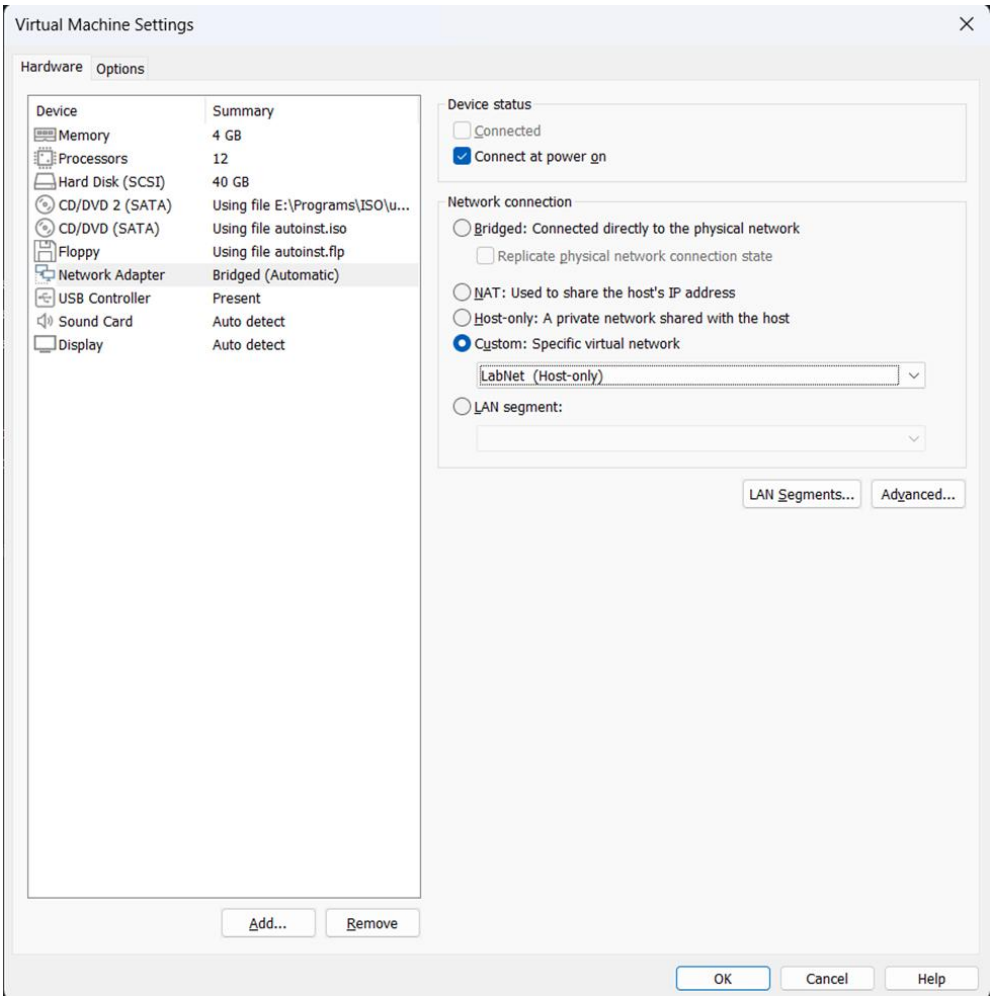


5.2. Practical Networking Configuration Steps

1. Create an **internal/host-only** virtual network named LabNet.



2. Attach Kali, GNS3 Server and Ubuntu Server to LabNet as default NICs.



3. Attach Ubuntu Suricata VM to LabNet and configure the Sonar/sensor interface to accept promiscuous traffic (VMware Workstation setting) or configure the virtual switch to mirror ports to the sensor (ESXi).
4. Ensure firewalls on the sensor do not block capture (Suricata listens on a raw interface; firewall typically not relevant, but ensure libpcap/af_packet can access interfaces).

6. Suricata: Installation, Configuration, and Rule Management

6.1. Installation (Ubuntu) — summary of required commands

This section lists the conceptual steps followed during lab preparation. Exact commands are provided in the appendix.

1. Update packages and install dependencies (apt update && apt upgrade).

```
# update and upgrade
sudo apt update && sudo apt upgrade -y
```

2. Install Suricata either from the OS package repositories (if current enough) or from the official packages/sources for the target Suricata version.

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

3. Install suricata-update and configure it to fetch rulesets (Emerging Threats open ruleset recommended for lab use).

```
# Get latest suricata-update if needed
sudo pip3 install --upgrade suricata-update
```

4. Configure Suricata to use af-packet where supported for best performance in virtual NICs, or pcap for simplicity.

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

# make sure the af-packet is like this:
af-packet:
    - interface: ens33
      threads: auto
      cluster-id: 99
      cluster-type: cluster_flow
      defrag: yes
      use-mmap: yes
      buffer-size: 64536
```

5. Configure suricata.yaml to enable EVE JSON logging (eve-log) with relevant modules (alerts, http, dns, tls, files).

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

# make sure the outputs section is like this:
outputs:
  - eve-log:
      enabled: yes
      filetype: regular
      filename: /var/log/suricata/eve.json
      rotate-interval: day
      types:
        - alert
        - http
        - dns
        - tls
        - files
        - flow
```

6. Start Suricata in IDS mode and verify it is capturing traffic.

```
# Test Suricata config
sudo suricata -T -c /etc/suricata/suricata.yaml

# Start/Restart system service
sudo systemctl restart suricata

# show only alert objects as they arrive
sudo tail -n 200 -f /var/log/suricata/eve.json | jq  'select(.alert != null)'
```

6.2. Rule Management and Updates

- Use suricata-update to pull rules and keep the ruleset fresh. For a lab, pulling Emerging Threats (ET Open) provides a wide set of signatures for reconnaissance and common exploits.
- Maintain a local.rules file for custom rules and testing; ensure Suricata is configured to load it.

6.3. Logging and Output

- **EVE JSON Output:** The EVE JSON log format is the preferred method for structured alerts and metadata. EVE can include alerts, HTTP logs, DNS logs, file extraction metadata, and TLS metadata.
- **PCAPs / Full Packet Logs:** For forensic validation, tcpdump can capture pcap files for sessions of interest, or Suricata can be configured to dump session pcaps.

7. Detection Techniques, Rules, and Tuning

7.1. Local rules file

Create /etc/suricata/rules/local.rules and ensure suricata.yaml includes the local rules path under rule-files:

```
# create local.rules
sudo nano /etc/suricata/rules/local.rules

# ensure suricata.yaml include them
sudo nano /etc/suricata/suricata.yaml
rule-files :
   - local.rules
  # - suricata.rules # This is commented to test local rules
```

7.2. Sample Custom Rules

ICMP sweep detection (high-level): triggers on unusual ICMP volume from a single source.

```
# DETECT ICMP PING SWEEP if more then 6 ping ATTEMPTS in 60 seconds

alert icmp any any -> any any (msg:"LAB ICMP Ping Sweep Detected"; \
itype:8; threshold: type both, track by_src, count 6, seconds 60; \
sid:1000002; rev:1;)
```

Nmap TCP SYN scan signature: detect rapid SYNs across many ports.

```
# SIMPLE Nmap port scan detection rule

alert tcp any any -> any any (msg:"LAB TCP SYN scan detected"; \
flags:S; threshold: type both, track by_src, count 10,\
seconds 60; sid:1000008; rev:1;)
```

HTTP suspicious download rule: match known malicious URL patterns or suspicious user-agent strings.

```
HTTP GET METHO ALERT
alert http any any -> any any (msg:"LAB HTTP GET detected";\
content:"GET"; http_method; sid:1000007; rev:1;)
```

7.2. Tuning Strategy

Effective tuning is iterative:

1. **Baseline:** Run the system under benign activity for 24–72 hours to collect baseline alerts and noise.
2. **Triaging:** Review alerts and label them as true positive, false positive, or benign.
3. **Adjustments:** Disable or lower priority of overly noisy rules; change thresholding; use flowbits or threshold options to aggregate repetitive alerts.
4. **Validation:** Re-run attack scenarios to ensure relevant detection rules still trigger.

7.3. False Positive Handling

- Use threshold to suppress noisy rules for specific flows.
- Customize rule conditions (e.g., require specific payload patterns) instead of generic signatures.
- Combine network alerts with host logs (Sysmon, Windows Event) to confirm suspicious activity.

8. Simulation Scenarios and Methodology

This section describes reproducible attack scenarios that the group can run during the lab demonstration. Each scenario includes an objective, expected Suricata alerts, relevant commands, and validation steps.

8.1. Scenario 1 — Port Scanning (Nmap)

Objective: Demonstrate detection of a typical reconnaissance scan.
Kali commands: `nmap -sS -p1-1024  192.168.100.3`
Expected Suricata output: TCP SYN scan alerts, metadata in EVE JSON showing multiple SYNs from attacker to many ports.
Validation: Use Suricata logs and a packet capture to show SYN sequences and correlate timestamps.

8.2. Scenario 2 — Suspicious Download / File Retrieval

Objective: Show file extraction and how Suricata logs file metadata.
Kali commands: Use `curl` to download a benign file that matches a test signature.
Expected Suricata output: HTTP log entry, file extraction record in EVE JSON, and an alert if the filename or payload matches a rule.
Validation: Inspect the extracted file artifact and cross-check with host logs.

9. Observations, Results, and Analysis

During initial lab runs, expect the following typical observations:

- **Noisy signatures:** Some rules will generate many false positives (e.g., benign port sweeps, background Windows traffic). Document and disable those rules for the demo.
- **Missing detections:** If Suricata runs in `pcap` mode or if virtualization isolates traffic incorrectly, Suricata may see limited traffic. Ensuring the correct capture configuration is essential.
- **Performance:** In a small lab, CPU and memory are sufficient for Suricata; however, thread settings and af-packet settings should be adjusted for higher throughput.

Example Analysis Steps:

1. Identify alert in EVE JSON and note timestamp, source IP, destination IP, and signature id.
2. Cross-check with tcpdump/Wireshark pcap for the same timestamp and flow to view packet-level evidence.
3. If an HIDS agent is installed on the Windows Server, check Windows Event logs or Sysmon entries for process creation, network connections, or suspicious file writes at the same timestamps.
4. Write a short incident summary linking the network alert to host artifacts.

10. Incident Response Playbook

This short runbook is intended for the class demonstration and to illustrate proper investigative steps.

1. **Identify & Validate:** Confirm alerts by reviewing Suricata EVE logs and pcap evidence.
2. **Contain:** In the lab, isolate the attacker VM or disable its network adapter to stop the activity.
3. **Collect Evidence:** Save Suricata logs, EVE JSON, and the pcap of the suspicious flow. Document timestamps and commands executed during the demo.
4. **Analyze:** Correlate network and host logs to determine the scope and mechanism of the activity.
5. **Remediate & Recover:** In a controlled lab this typically means cleaning test artifacts and restoring snapshots of VMs.
6. **Review & Tune:** Update rules to reduce false positives and add necessary signatures to improve detection.

11. Limitations, Risks and Mitigations

11.1. Limitations

- **Virtualization artifacts:** Some network characteristics in virtual networks differ from physical networks — timing, offloading, and NIC behavior can alter signatures.
- **Scope:** This lab demonstrates detection of common patterns but does not simulate advanced persistent threats or encrypted malicious traffic at scale.

11.2. Risks

- **Accidental exposure:** Avoid bridging the lab to production networks to prevent accidental scanning or exploits from propagating.
- **Tool misuse:** Ensure the class understands ethical use; do not run destructive payloads.

11.3. Mitigations

- Use host-only networks and snapshots. Keep the lab isolated. Use non-destructive test payloads.

12. Recommendations and Future Work

- **Hybrid Deployment:** Combine Suricata NIDS with a host-based agent (Sysmon + Wazuh/Winevent, or OSSEC) to improve correlation and detection of post-exploitation.
- **Centralized Logging & Visualization:** Forward EVE JSON to an ELK/Opensearch or Splunk instance for search and dashboarding. A simple Kibana dashboard improves presentation clarity.
- **Automation:** Use `suricata-update` in a scheduled job to keep rulesets updated; version control local rules.
- **Advanced Simulation:** Use virtualization tools (GNS3 / EVE-NG) for more complex topologies or to integrate network devices and higher-fidelity traffic flows.

14. References

The following resources provide authoritative information useful to reproduce the lab and to deepen your understanding of the technologies discussed:

1. **Suricata Official Documentation** — Installation, configuration, and `suricata.yaml` references. https://docs.suricata.io/en/latest/configuration/suricata-yaml.html
2. **suricata-update** — Rule management tool documentation and Emerging Threats rules information. https://docs.suricata.io/en/latest/rule-management/suricata-update.html
3. **VMware Documentation** — Workstation and ESXi guides on virtual networking, promiscuous mode, and port mirroring. https://community.spiceworks.com/t/vmware-promiscuous-mode/260055/6
4. **Nmap Documentation** — For scan types and usage flags used in reconnaissance simulations. https://nmap.org/book/man.html
5. **Sysinternals / Sysmon Documentation** — For host-based event capture and correlation on Windows systems. https://learn.microsoft.com/en-us/sysinternals/downloads/sysmon
6. **Wireshark / tcpdump Documentation** — For pcap capture and forensic analysis. https://www.wireshark.org/docs/wsug_html_chunked/AppToolstcpdump.html
7. **Snort / Zeek Documentation** — For comparative study and alternative IDS architectures. https://docs.zeek.org/en/master/