Project 1 Report – Network Intrusion Detection System (NIDS) using Suricata

Full Name(s):
Shreyas Gurrapu, Folasade Nasir, Raphooko Phooko
Program: Infotact Solutions Cybersecurity Internship
Date: 17/09/2025
Mentor: Vasudev Jha

Introduction

This report document describes the setup and methodology of implementing project 1 of the Infotact Solutions Internship. The report will also list corresponding best practice rules and recommendations.

1. Objective

The objective of this was to create a virtualized security lab to use an open-source NIDS like Snort or Suricata to be deployed to monitor network traffic. The NIDS will then be configured with custom rules designed to detect said malicious activities, such as reconnaissance scans, brute-force login attempts, and known malware communication, providing immediate alerts to security analysts for investigation.

2. Scope

- Deploy and configure a NIDS Suricata in a virtual lab environment.
- Design and implement a robust set of custom detection rules to identify: reconnaissance scans (Nmap), brute-force login attempts (SSH/FTP), and malware-like C2 beaconing.
- Generate realistic attack traffic using common penetration testing tools and verify that the NIDS raises appropriate alerts.
- Document the setup, testing process, evidence (logs/screenshots), and recommendations for production hardening.

3. Tools and Lab Environment Used

Virtual Machines :

Attacker VM: Kali Linux (tools: Hydra)

Target/Services VM: Ubuntu Server or multi-service VM (SSH, FTP, HTTP)

NIDS VM: Ubuntu Server with Suricata installed

Virtualization – Oracle Virtual VM VirtualBox

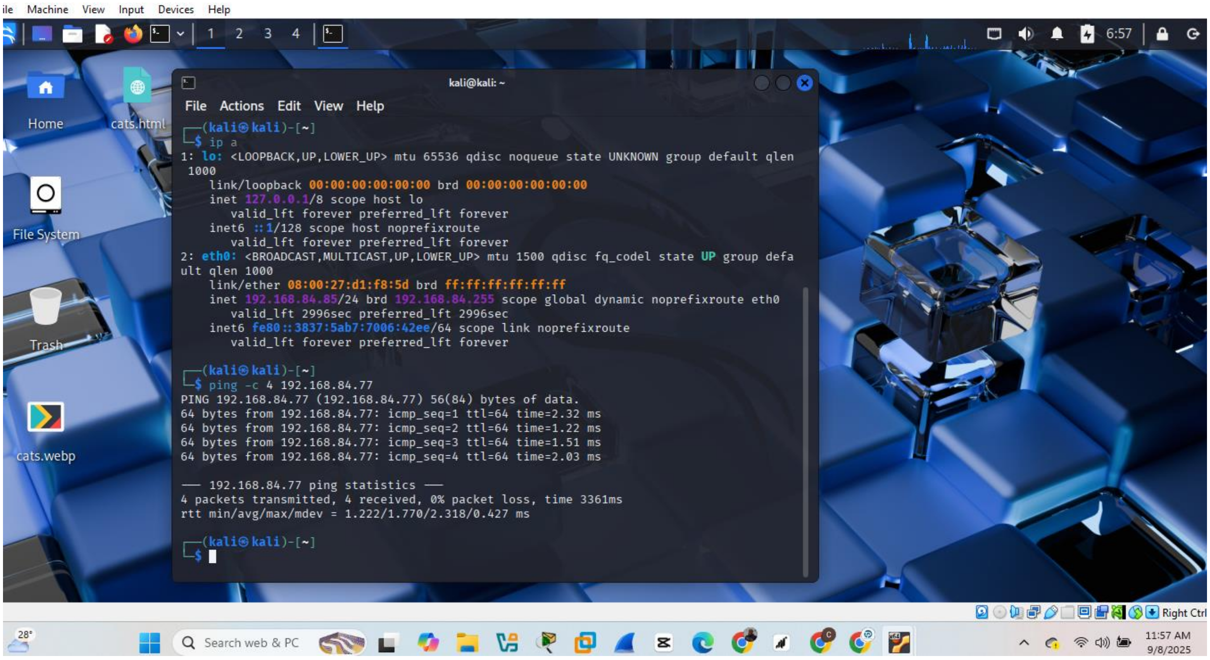
Primary Tools - NIDS Engine: Suricata - Traffic & Attack Tools: Nmap, Hydra, Metasploit Framework - Analysis & Scripting: Wireshark, tcpdump, Bash, Python

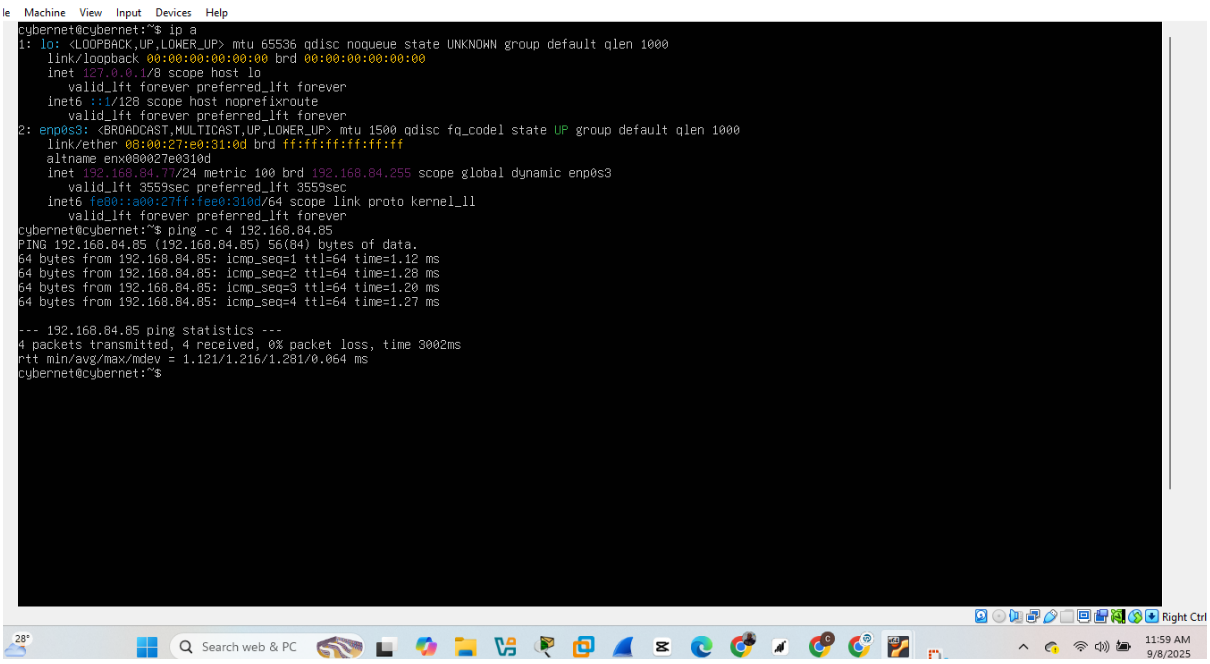
4. Methodology and Implementation (Week-by-week)

Week 1 — Lab Setup

- Chose Kali Linux (attacker) and Ubuntu Server(target + NIDS) VMs on VMware.
- Suricata is then installed on the NIDS VM and configured `suricata.yaml` for home/ lab network interface monitoring.
- Captured baseline traffic to establish normal behavior and tuned logging (eve.json, fast.log).

Suricata started successfully and generated baseline logs with no false positives for benign traffic.





Week 2 — Reconnaissance (Nmap) Rules

- Custom rules are written to detect common Nmap scans (SYN, FIN, Xmas/Null).
- Generated scan traffic from the Kali Linux (attacker) VM using Nmap and validated that Suricata produced alerts.

Test commands (examples):

- SYN scan: nmap -sS -p 1-65535 -T4 192.168.84.77 #ubuntu server IP
- FIN scan: nmap -sF 192.168.84.77 #ubuntu server IP
- Xmas scan: nmap -sX 192.168.84.77 #ubuntu server IP

Week 3 — Brute Force Detection

- Implemented rules aimed at detecting SSH and FTP brute-force patterns (there were multiple failed authentication attempts from a single source within a short interval).
- Simulated attacks using Hydra and Metasploit as they are well known brute forcing tools coming preinstalled with Kali Linux.

Test commands (examples):

- Hydra SSH brute-force: hydra -l root -P passwords.txt ssh://192.168.84.77

Week 4 — Malware Command and Control (C2) Beaconing

- Created signature-based rules for simple C2 beaconing characteristics (periodic HTTP beacons, suspicious user-agents, or fixed-length TCP heartbeat patterns).
- Attempted to emulate beaconing traffic using simple Python scripts or Metasploit to generate periodic outbound HTTP requests to a controlled C2 endpoint.

Test commands (examples):

-Python beacon (simple):

```
import requests

import time

TARGET_URL = "http://<Ubuntu VM address>/beacon"

HEADERS = { "User-Agent": "MalwareBot" }

def send_beacon():

    while True:
```

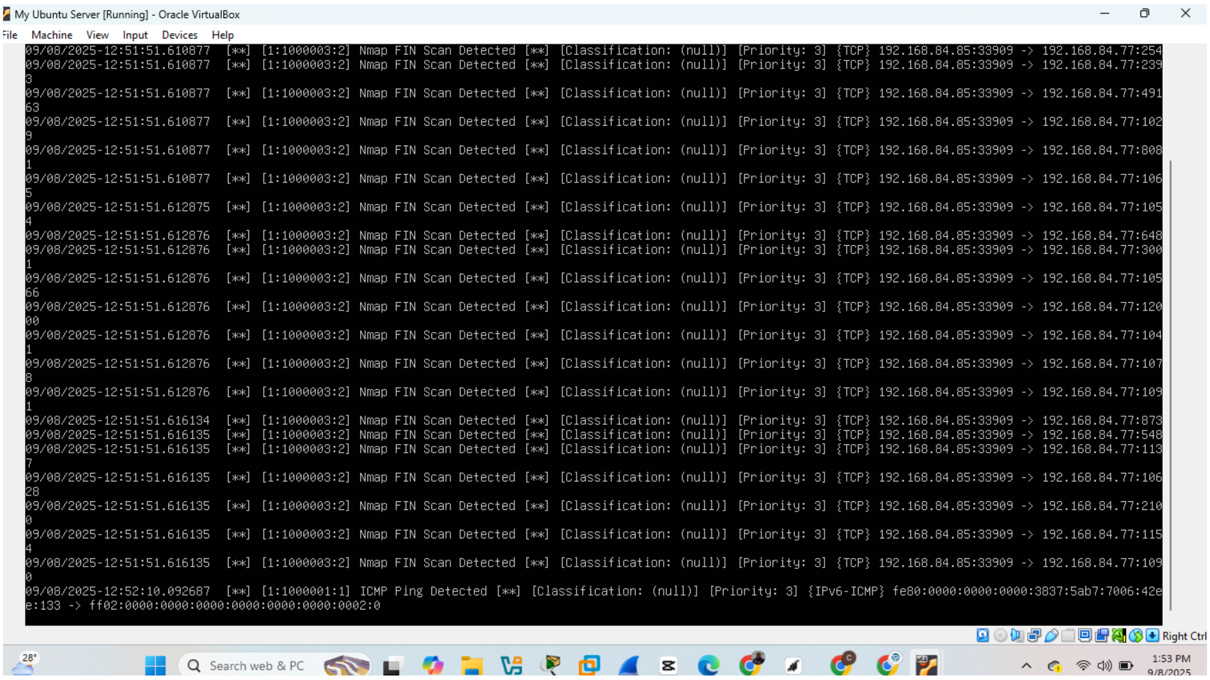
```
    try:
      response = requests.get(TARGET_URL, headers=HEADERS)
      print(f"Beacon sent: {response.status_code}")
    except Exception as e:
      print(f"Error sending beacon: {e}")
    time.sleep(10) # Beacon every 10 seconds


if __name__ == "__main__":
  send_beacon()
```

5. Example Custom Rules (found in /etc/suricata/rules/custom.rules)

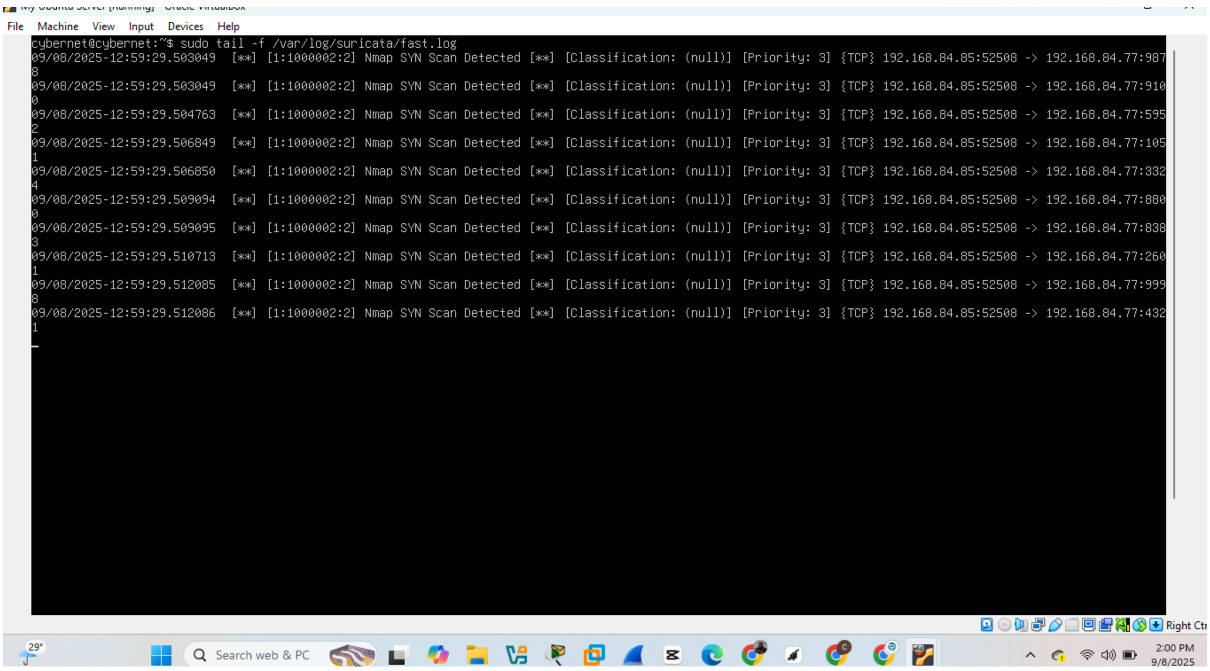
Detect Nmap SYN scan (TCP SYN to many ports)

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"NIDS: Possible Nmap SYN scan"; flags:S; threshold: type both, track by_src, count 20, seconds 60; sid:1000001; rev:1;)
```



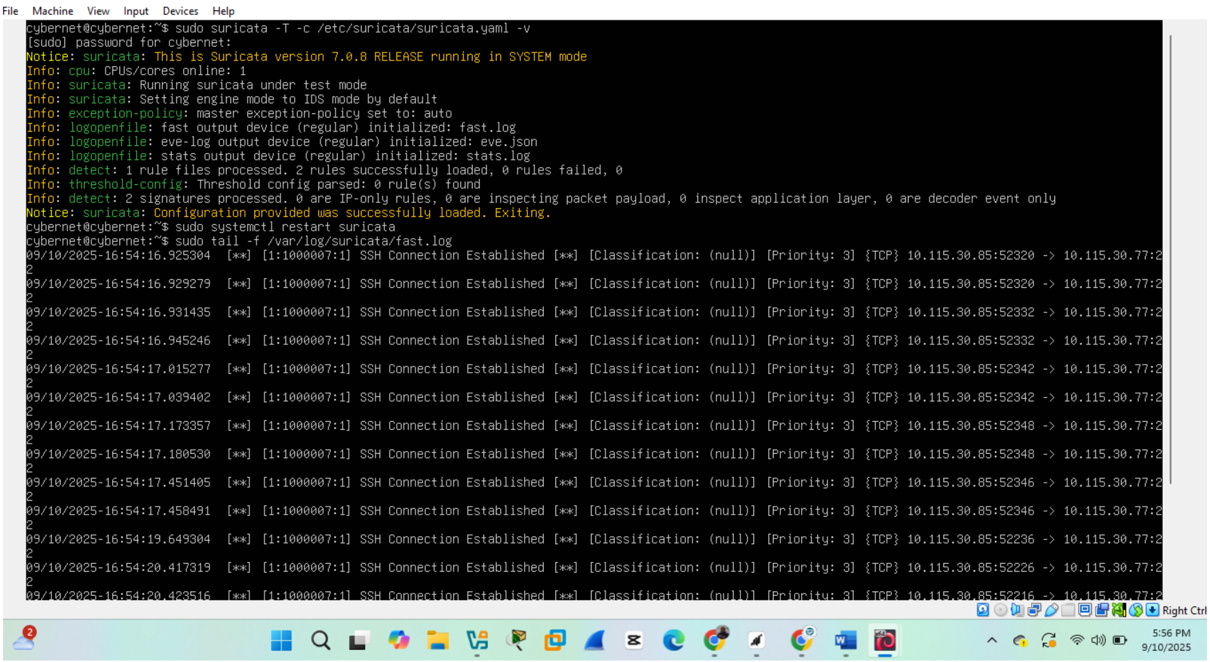
Detect Nmap FIN/XMAS/NULL scans (odd flags)

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"NIDS: Possible Nmap FIN /XMAS/NULL scan"; flags:FPU; threshold: type both, track by_src, count 10, seconds 60; sid:1000002; rev:1)
```



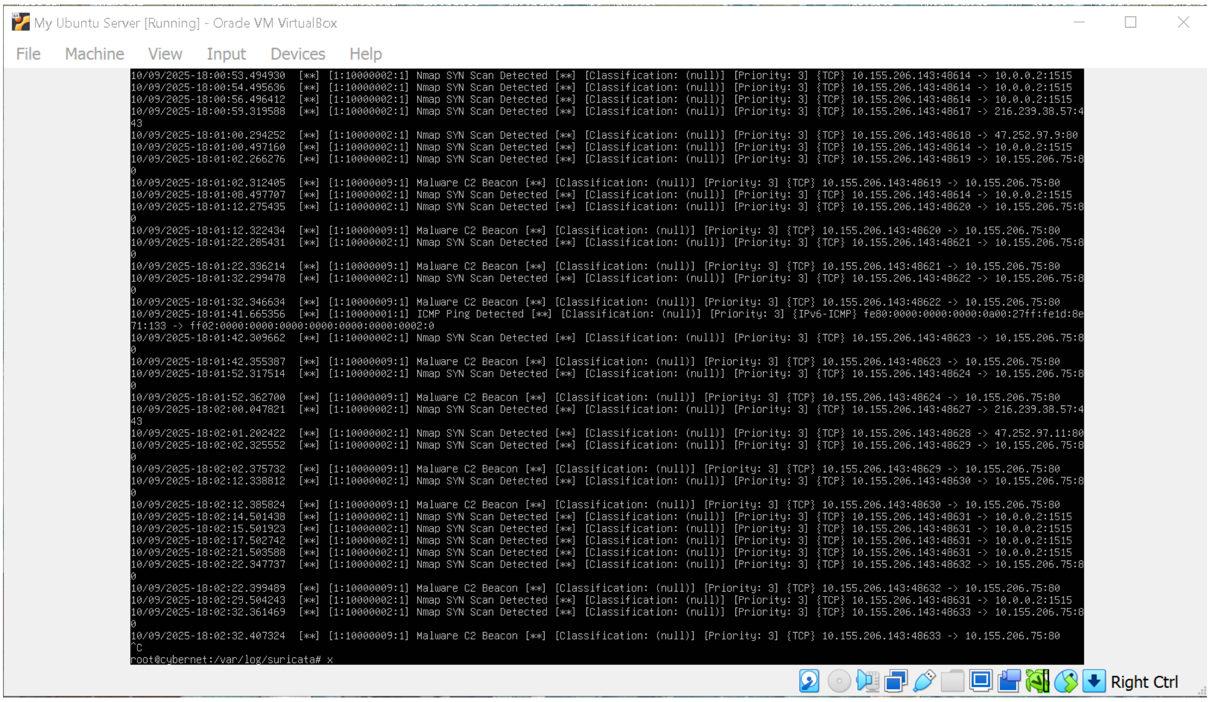
Detect SSH brute-force (many connections to port 22)

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 22 (msg:"NIDS: SSH brute-force attempt detected"; detection_filter: track by_src, count 5, seconds 60; sid:1000003; rev:1;)
```



Detect simple HTTP C2 beacon pattern (suspicious User-Agent or fixed URI)

```
alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"NIDS: Suspicious HTTP beacon (possible C2)"; http.user_agent; content:"BadAgent/1.0"; nocase; sid:1000004; rev:1;)
```



The project highlights how foundational configuration and custom rule development play integral roles in building effective intrusion detection capabilities. Challenges encountered during setup, particularly configuration hiccups, underlined the importance of precise environment preparation. The detection successes with scan types affirm the viability of open-source NIDS solutions for network security monitoring in a lab or enterprise context.

6. Attack Commands

Reconnaissance / Scanning

- `nmap -sS -p 1-65535 -T4 192.168.84.85`

- `nmap -sF 192.168.84.85`

- `nmap -sX 192.168.84.85`

Brute-force / Authentication Testing

- `hydra -l admin -P /usr/share/wordlists/rockyou.txt ssh:// 192.168.84.85`

C2 Beacon Simulation

- A simple python script to create periodic HTTP GET requests to a controlled endpoint.

6. Results of Simulated Attacks

- Each simulated attack produced Suricata alerts as expected. Alert records were stored in `eve.json` (JSON format) and also visible in `fast.log` for quick review.
- Alerts that are capured include timestamp, source/destination IPs, alert message, SID, and relevant payload snippets.

7. Conclusion

Our team successfully established a functional NIDS deployment capable of detecting basic network reconnaissance activities through attentive configuration and rule creation. The lab demonstrates a practical workflow for developing, testing, and tuning custom NIDS rules using Suricata/Snort. The ruleset reliably detected the simulated reconnaissance scans, brute-force attempts, and simple C2-like beaconing in the controlled environment.