كلية الحاسبات والذكاء الاصطناعي
FACULTY OF COMPUTERS & ARTIFICIAL INTELLIGENCE

Intrusion Prevention System

A senior project submitted in partial fulfillment of the requirements for the degree of Bachelor of Computers and Artificial Intelligence.

"Information Security & Digital Forensics" Program,

Project Team

1- **Salah Foad Mohamed**
2- **Mai Medhat Qandil**
3- **Shahd Hassan Farag**
4- **Mahmoud Sarwat Elkhouly**
5- **Ahmed Megahed Shaker**

Under Supervision of

Dr.Mohamed Taha

Benha, June 2025

Dedication

This project is dedicated to my beloved family, whose unwavering support and encouragement have been my greatest strength.

To my professors and mentors, whose guidance and wisdom have shaped my academic journey.

To my friends, for their constant motivation and belief in my abilities.

To my colleagues, for the shared experiences and teamwork that made this journey memorable.

To all those who inspire me to strive for excellence and never give up.

And finally, to the pursuit of knowledge and innovation, which drives us to make a difference.

Acknowledgment

I would like to express my sincere gratitude to **Dr. Mohamed Taha** for his invaluable guidance, continuous support, and encouragement throughout the development of this project. His insights and expertise have significantly contributed to the quality and direction of this work.

I am also deeply thankful to my supervisor, **Eng. Ahmed Tawfik**, for his dedicated mentorship and constructive feedback during all stages of the project. His support has been instrumental in shaping the final outcome.

My appreciation extends to the faculty members and professors at **Benha University** for providing the academic foundation and resources that made this project possible.

Special thanks to my family for their unconditional love, patience, and motivation. Their belief in me has been a constant source of strength and inspiration.

Finally, I would like to thank everyone who contributed, directly or indirectly, to the successful completion of this project. Your support has been deeply appreciated.

Declaration

We hereby declare that the project titled **"Network Intrusion Prevention System"** is our original work and has been conducted independently under the valued guidance and support of **Dr. Mohamed Taha**, and under the supervision of **Eng. Ahmed Tawfik**.

We affirm that this work has not been submitted previously, in whole or in part, for any degree or academic qualification at this or any other institution. All sources and references used have been properly cited and acknowledged in accordance with academic integrity guidelines.

We take full responsibility for the content presented and confirm that the information and findings are based on our own research and analysis.

Signed: _____

Date: 16th of June 2025

Abstract

With the rapid growth of digital networks, cybersecurity threats have become a major concern. This project, titled "Network Intrusion Prevention System" (NIPS), focuses on detecting and preventing unauthorized access and malicious activities in network environments. The primary objective is to develop an effective intrusion prevention system that enhances network security by identifying and mitigating potential threats in real-time.

To achieve this, the system utilizes IPS (Suricata), firewall (OPNsense), Dos attack using Metasploit, Python, Linux, and VMware to analyze network traffic and detect suspicious patterns. The results demonstrate that our approach improves threat detection accuracy while minimizing false positives, ensuring robust security for networks.

This study contributes to the field of network security by providing an efficient and proactive solution to mitigate cyber threats. Future enhancements may include integrating artificial intelligence for adaptive threat detection and expanding the system’s capabilities for large-scale enterprise networks.

Keywords: Intrusion Prevention, Network Security, Cyber Threats, Anomaly Detection, Intrusion Detection Systems (IDS), Firewall.

Table of Contents

LIST OF ABBREVIATIONS 7

LIST OF FIGURES 8

1. INTRODUCTION 9

1.1. Problem Definition 10
1.2. Key Issues Addressed by NIPS 10
1.3. Problem Statement 10
1.4. Importance Of network Intrusion Prevention System 11
1.5. How Do We Solve This Problem? 12
1.6. Market Solutions 12
1.7. Implementation And Technologies Used 14
1.7.1. System Architecture and Implementation 14
1.7.2. Conclusion 14
1.8. Project Objectives 15
1.9. Advantages Of Using Our Software 15

2. ANALYSIS & DESIGN 16

2.1. User And System Requirements 16
2.1.1. System Requirements 16
2.1.2. User Requirements 17
2.2. Functional Requirements 18
2.3. Non-Functional Requirements 19
2.4. Block diagram 20
2.5. Context diagram 22
2.6. Model design 25
2.7. Prediction design 28
2.8. Algorithm design 31
2.9. Use Case Diagram 33
2.10. Activity Diagram 35
2.11. Sequence Diagram 38
2.12. Class Diagram 39
2.13. DFD (Data Flow Diagram) 43
2.14. ERD (Entity Relationship Diagram) 48

3. Deliverable and Evaluation 50

3.1. IPS 50
3.2. Suricata IPS 50
3.3. Simulating a Dos Attack Using Metasploit 51
3.4. Simulating a DDoS Attack Using Hping3 51
3.5. OPNsense Firewall 53

4. PRACTICAL IMPLEMENTATION 54

4.1. Virtual Environment Setup 54
4.2. Routing Techniques 55
4.3. Practical Connection of VMs 57
4.4. Project Execution and Simulation Steps 58
4.5. Project Screenshots and Visuals 61
4.6. Practical Challenges and System Performance 64
4.7. Python Code Documentation 64

5. Conclusion and References 69

5.1. Conclusion 69
5.2. References 71

List Of Abbreviations

IPS	Intrusion Prevention System
IDS	Intrusion Detection System
NIP	Network Intrusion Prevention System
DoS	Denial of Service
DDoS	Distributed Denial of Service
OISF	Open Information Security Foundation
DPI	Deep Packet Inspection
NGFW	Next-Generation Firewall
SOC	Security Operations Center
SIEM	Security Information and Event Management
GDPR	General Data Protection Regulation
HIPAA	Health Insurance Portability and Accountability Act
PCI-DSS	Payment Card Industry Data Security Standard

List of Figures

Figure 2-1: Block Diagram .....

Figure 2-2: Context Diagram.....

Figure 2-3: Model Design .....

Figure 2-4: Prediction Design.....

Figure 2-5: Algorithm Design We Use.....

Figure 2-6: Use Case Diagram .....

Figure 2-7: Activity Diagram .....

Figure 2-8: Sequence Diagram .....

Figure 2-9: Class Diagram.....

Figure 2-10: DFD (Data Flow Diagram).....

Figure 2-11: ERD (Entity Relationship Diagram) ...

Chapter One

1.INTRODUCTION

In today's digital world, cyber threats are evolving at an alarming rate, posing serious risks to organizations and individuals. With the increasing reliance on computer networks for communication, data storage, and business operations, the need for robust cybersecurity measures has become more critical than ever. Hackers and malicious actors constantly attempt to exploit network vulnerabilities, launch denial-of-service (DoS) attacks, and gain unauthorized access to sensitive information. Traditional security measures, such as firewalls and antivirus software, are often insufficient in dealing with sophisticated and evolving cyber threats.

To address these challenges, we have developed a Network Intrusion Prevention System (NIPS) that actively monitors and protects network infrastructure from cyber-attacks. Our system is designed to detect, analyze, and prevent malicious activities in real-time, providing an additional layer of security beyond traditional defense mechanisms. By leveraging advanced Intrusion Prevention System (IPS) technologies, firewalls, and penetration testing tools, our solution aims to enhance network security, minimize vulnerabilities, and prevent unauthorized access before damage occurs.

The project incorporates Suricata, an open-source intrusion prevention system, to monitor network traffic and detect suspicious behavior using deep packet inspection. Additionally, FortiGate firewall is integrated to provide advanced security features such as traffic filtering, threat intelligence, and malware prevention. To ensure the effectiveness of our system, we conducted extensive penetration testing using Metasploit and simulated DoS attacks, allowing us to evaluate and improve the system's response to real-world cyber threats.

Our implementation is built on a Cisco Server with a Linux-based environment, ensuring high performance, security, and scalability. To manage and visualize network security events, we developed a web-based dashboard using PHP, HTML, CSS, and JavaScript, allowing administrators to monitor real-time alerts, analyze detected threats, and configure security settings. The entire system is tested and deployed within a virtualized environment using VMware, providing a controlled and secure platform for experimentation and evaluation.

By combining intrusion prevention technologies, firewalls, penetration testing, and real-time monitoring, our Network Intrusion Prevention System (NIPS) provides a comprehensive cybersecurity solution.

This project demonstrates how modern cybersecurity techniques can be used to proactively defend against cyber threats, ensure data integrity, and protect network infrastructure from potential attacks.

1.1. Problem Definition

With the rise of cyber threats, organizations face increasing risks of network intrusions, data breaches, and denial-of-service (DoS) attacks. Traditional security measures like firewalls and antivirus software are not always sufficient to detect and prevent sophisticated cyber threats in real time. Attackers exploit network vulnerabilities, leading to data loss, unauthorized access, and service disruptions.

To address this issue, there is a need for a Network Intrusion Prevention System (NIPS) that can monitor, detect, and block malicious activities before they cause harm. Our project aims to develop an efficient IPS solution using Suricata and FortiGate firewall, capable of real-time threat detection, deep packet inspection, and proactive defense against cyber-attacks.

1.2. Key Issues Addressed by NIPS:

- Unauthorized Access: Detects and prevents attackers from gaining unauthorized entry into the network.
- Malware and Exploits: Identifies malicious payloads in network traffic and prevents them from reaching internal systems.
- Zero-Day Attacks: Uses behavior-based detection and anomaly analysis to identify new and unknown threats.
- DDoS Attacks: Identifies and mitigates distributed denial-of-service attacks that can overwhelm network resources.
- Policy Enforcement: Ensures compliance with security policies by blocking unauthorized applications or connections.

1.3. Problem Statement:

The increasing sophistication of cyber threats necessitates an advanced security solution that goes beyond traditional detection methods. A Network Intrusion Prevention System (NIPS) must be capable of real-time traffic analysis, threat identification, and automatic mitigation while minimizing false positives and maintaining network performance. The challenge lies in developing a highly accurate, scalable, and adaptive NIPS that effectively prevents cyber threats without disrupting legitimate network activities.

1.4 Importance of Network Intrusion Prevention System (NIPS) in cybersecurity

A Network Intrusion Prevention System (NIPS) plays a critical role in modern cybersecurity by proactively detecting and preventing malicious activities before they compromise a network. As cyber threats become more sophisticated, organizations need advanced security mechanisms to protect sensitive data, maintain system integrity, and ensure network availability.

Key Importance of NIPS in Cybersecurity:

- Real-Time Threat Prevention:

  Unlike traditional security measures that only detect threats, NIPS actively blocks malicious activities in real time, preventing potential breaches before they cause harm.

- Protection Against Advanced Attacks:

  NIPS helps defend against zero-day attacks, malware, exploits, and DDoS attacks that can bypass traditional firewalls and antivirus solutions.

- Minimization of Data Breaches:

  By blocking unauthorized access and malicious traffic, NIPS prevents data theft, financial loss, and reputational damage caused by cyber intrusions.

- Reduced False Positives:

  Advanced NIPS solutions use machine learning and behavioral analysis to distinguish between legitimate and malicious traffic, reducing unnecessary alerts and improving accuracy.

- Automated Incident Response:

  NIPS automates threat detection and response, reducing the workload on cybersecurity teams and allowing faster mitigation of attacks.

- Compliance with Security Standards:

  Many regulatory frameworks (e.g., GDPR, HIPAA, PCI-DSS) require organizations to implement intrusion prevention measures to protect sensitive information.

- Network Performance Optimization:

  By filtering out harmful traffic, NIPS helps maintain network stability and performance, ensuring smooth business operations without disruption.

1.5. How Do We Solve This Problem?

To solve the problem of cyber-attacks on the network, a Network Intrusion Prevention System (NIPS) detects and prevents malicious activity by continuously monitoring network traffic. It uses techniques like deep packet inspection (DPI) and anomaly detection to identify suspicious patterns and unauthorized access. The system is regularly updated with new attack signatures, and in case of an intrusion, it takes immediate action to block the threat. Post-attack analysis helps strengthen the system for future attack.

1.6. Market Solution

Market solutions for cyber-attacks on networks typically involve a variety of Network Intrusion Prevention Systems (NIPS) and security appliances that combine multiple technologies. Some of the most common solutions include:

- **Suricata:**

  Suricata is a high-performance Network IDS, IPS, and Network Security Monitoring engine. It is open source and owned (as well as developed) by a community-run non-profit foundation, the Open Information Security Foundation (OISF). Suricata inspects all traffic on a link for malicious activity and can extensively log all flows seen on the wire, producing high-level situational awareness and detailed application layer transaction records. It needs specific rules (holding instructions) to tell it not only how to inspect the traffic it looks at but also what to look for. Suricata was designed to perform at very high speeds on commodity and purpose-built hardware.

❖ **The most important Suricata features and capabilities are:**

- Deep packet inspection Packet capture logging Intrusion Detection Intrusion
- Prevention IDPS Hybrid Mode Network Security Monitoring Anomaly Detection
- Lua scripting (You can use the Lua programming language to write custom
- scripts that will be executed when a particular rule or signature triggers an alert)
- Rust (Allows Suricata to fail in a safe mode) GeolP Multitenancy File Extraction
- (from protocols like SMTP, HTTP, etc.) Full IPv6 and IPv4 support IP Reputation
- JSON logging output Advanced protocol inspection Multi-threading

- **Snort:**

Snort is an open-source Intrusion Detection System (IDS) and Intrusion Prevention System (IPS) developed by Martin Roesch in 1998 and later acquired by Cisco. It is widely used in network security to detect and prevent malicious activities by analyzing network traffic in real time.

Snort operates by inspecting network packets using a set of predefined rules to detect attacks. It works in three primary modes:

Sniffer Mode – Captures and displays real-time network traffic.

Packet Logger Mode – Logs network traffic for later analysis.

Network Intrusion Detection/Prevention Mode – Actively analyzes traffic and takes action based on rule-based detection

- **OPNsense Firewall**

OPNsense is an open-source, FreeBSD-based firewall and routing platform. It offers a wide range of features, making it a popular choice for network security among both home users and enterprises. OPNsense is designed with a strong focus on security, performance, and user-friendliness.

❖ **Some key features of OPNsense include:**

- Firewall and NAT: Advanced filtering rules and network address translation for secure traffic control.

- VPN Support: Supports multiple VPN types like IPsec, OpenVPN, and WireGuard for secure remote access.

- Intrusion Detection & Prevention System (IDS/IPS): Uses Suricata to detect and block suspicious traffic.

- Web Interface: A clean and responsive web-based GUI for easy configuration and monitoring.

- Traffic Shaping: Helps prioritize bandwidth for critical applications.

- Reporting and Monitoring: Includes real-time graphs, logs, and statistics.

1.7. Implementing And Technologies Used

In our Network Intrusion Prevention System (NIPS) project, we developed a robust security solution to detect and prevent cyber threats in real time. Our implementation involved several key components, including hardware, software, and security tools to ensure effective protection against network intrusions. Below is a detailed explanation of our approach and the technologies we used.

1.7.1. System Architecture And Implementation

Our system is designed to monitor network traffic, detect potential threats, and take proactive measures to prevent attacks. The main components of our implementation include:

- ***Virtualization with VMware***

We utilized VMware to create a virtual environment for testing and simulating various network attacks. This allowed us to safely deploy and test our Intrusion Prevention System (IPS) in a controlled setting.

- ***Operating System: Linux***

The entire system runs on Linux, known for its security, stability, and performance in network environments. We used Linux to configure and manage various security tools, ensuring seamless integration with our IPS.

- ***Intrusion Prevention System (IPS) - Suricata***

To detect and prevent malicious network activities, we implemented Suricata, an open-source Intrusion Prevention System (IPS). Suricata provides high-performance network traffic analysis and deep packet inspection, enabling us to detect cyber threats such as malware, unauthorized access, and exploits.

- ***Simulating Attacks: Metasploit, DoS and DDOS Attacks***

To test the effectiveness of our IPS and firewall, we conducted penetration testing using:

Metasploit, a powerful penetration testing tool, to simulate real-world cyberattacks.
Denial-of-Service (DoS) attack simulations, to evaluate how well our system can prevent and mitigate such threats.
Distributed Denial Of Service attack simulations, DDoS makes a server crash by sending too much traffic.

1.7.2. Conclusion

By integrating Suricata IPS, FortiGate firewall, and penetration testing tools like Metasploit, we developed a highly secure Network Intrusion Prevention System (NIPS). The system successfully detects and prevents malicious activities, providing a real-time security solution against cyber threats. Our web-based dashboard allows administrators to monitor network security, analyze alerts, and configure security settings, making our project a comprehensive and practical cybersecurity solution.

1.8. Project Objectives

Enhance Network Security: Detect and prevent cyber threats.
Real-time Detection: Monitor network traffic for suspicious activity.
Automated Mitigation: Block malicious traffic automatically.
Scalability: Handle various network sizes without performance loss.
Customization: Tailor detection rules for specific needs.
Logging and Reporting: Provide detailed attack logs and reports.
Compliance: Meet industry security standards.
Integration: Work with other security tools.
Cost-effective: Provide an affordable solution for different organizations.

1.9. Advantages Of Using Our Software

Our software provides real-time threat detection and automatic blocking, ensuring robust network security. It’s scalable, customizable, and integrates easily with existing tools while being cost-effective and low maintenance

Chapter Two

2. ANALYSIS & DESIGN

2.1. User and System Requirements

2.1.1. System Requirements

Operating System Compatibility: The system should be compatible with Windows operating System (Win10 and above).

Python Version & Dependencies: The system must be developed and run with Python 3.12 or later. Dependencies must be installed

Networking Capability: The system should be able to capture and analyse network packets from a selected interface

Detection Engine: The system must have a rule engine capable of interpreting and Applying rules for intrusion detection and prevention

Alerting Mechanism: An alerting mechanism should be in place to notify users of Detected threats based on specified rules.

Modularity & Scalability: The system architecture should be modular, allowing easy integration of new features and scalability for larger networks. (OOP concepts in python programming)

Logging & Reporting: The system should maintain detailed logs of network activities and provide reporting capabilities for analysis.

User Authentication: A secure user authentication system should be implemented

to control access to the system.

Performance: The system should require the least resources possible without

affecting network performance significantly or affecting the host CPU resources.

2.1.2. User Requirements

Easy Installation & Setup – The software should have a simple installation process with clear instructions.

User-Friendly Interface – A dashboard with intuitive navigation for monitoring and managing security events.

Real-Time Alerts – Instant notifications for detected threats to allow quick response.

Customizable Security Rules – Users should be able to modify detection rules based on their needs.

Log & Report Access – Detailed logs and reports should be easily accessible for analysis.

Multi-User Support – Different user roles (Admin, Analyst) with controlled access levels.

Integration with Existing Security Systems – Compatibility with firewalls, SIEMs, and other security tools.

Regular Updates & Maintenance – The system should be easy to update and maintain with minimal downtime.

2.2. Functional Requirements

1. Traffic Monitoring & Analysis
   – Continuously monitor and analyze network traffic.

2. Signature-Based Detection
   – Detect threats based on known attack signatures.

3. Anomaly-Based Detection
   – Identify suspicious activities through behavioral analysis.

4. Real-Time Threat Prevention
   – Take immediate action to prevent detected threats.

5. Packet Filtering
   – Filter suspicious packets before they reach the internal network.

6. Deep Packet Inspection (DPI)
   – Inspect packet contents to detect hidden threats.

7. Logging & Reporting
   – Record all suspicious activities and generate analytical reports.

8. Incident Alerting
   – Send instant alerts upon detecting an attack or unusual activity.

9. Integration with SIEM
   – Support integration with Security Information and Event Management (SIEM)
Systems.

10. User Access Control
   – Manage user permissions for system configuration.

10. Automatic Updates
   – Update the threat database automatically.

11. Performance Optimization
   – Ensure minimal impact on network performance during inspection.

12. False Positive Reduction
   – Reduce false alarms by improving detection algorithms.

13. Custom Rule Creation

– Allow users to define custom detection rules.

14. Multi-Layer Defense

– Implement multiple layers of security to protect the network from various attack vectors.

2.3. Non-Functional Requirements

1. Scalability
   – The system must handle increasing network traffic without performance degradation.

2. Performance Efficiency
   – The system should operate quickly and efficiently without affecting network performance.

3. Reliability
   – The system should function consistently with minimal failures.

4. Availability
   – The system must ensure continuous operation with minimal downtime.

5. Security
   – The system itself must be protected against attacks and unauthorized access.

6. Usability
   – The system should have a user-friendly interface for security teams.

7. Maintainability
   – The system should be easy to update and maintain.

8. Compatibility
   – The system must work seamlessly with various operating systems and protocols.

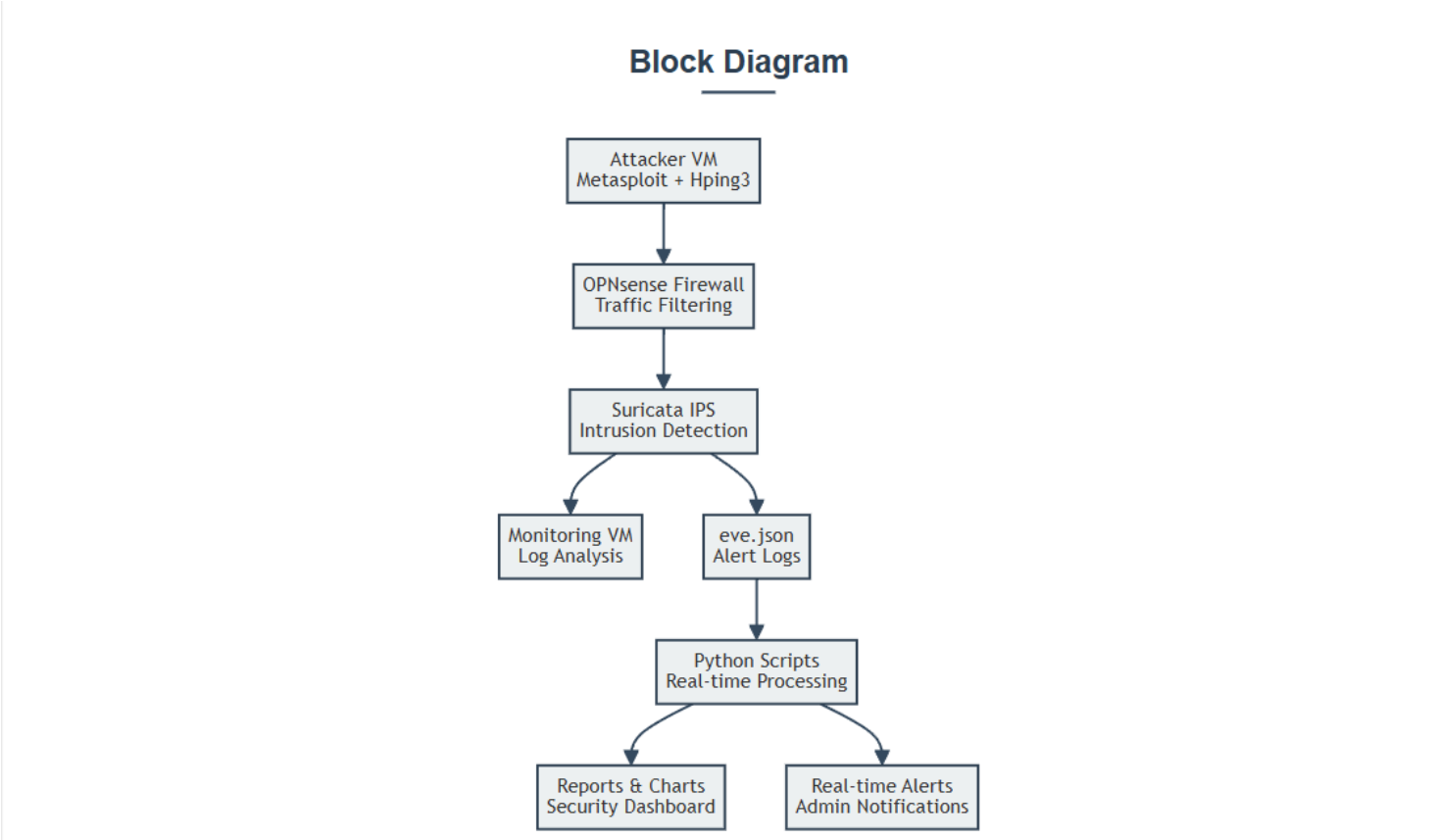
9. Configurability
   – The system should be flexible and customizable to fit different network needs.

10. Fault Tolerance
    – The system should continue operating even if some components fail.

11. Logging & Auditing
    – The system must maintain detailed logs of all security activities for analysis and review.

12. Disaster Recovery
    – The system must include recovery plans to quickly restore functionality in case of failure.

2.4. Block Diagram



Block Diagram Overview: This diagram shows the main components of our NIPS system. Traffic flows from the attacker through the firewall to the IPS, which logs events that are processed by Python scripts for real-time monitoring and reporting.

This block diagram comprehensively illustrates the architecture and data flow of our **Network Intrusion Prevention System (NIPS)**, showcasing how each component seamlessly integrates to form a cohesive and robust cybersecurity solution. The diagram provides a detailed overview of the systematic approach we employed for network monitoring, intrusion detection, and threat mitigation within our virtualized lab environment.

Component Overview:

1. Attacker VM: Threat Simulation Environment

The Attacker VM serves as a controlled environment for simulating cyber threats. It is equipped with:

- **Metasploit Framework**: Used to generate sophisticated attack vectors such as denial-of-service (DoS) attacks, buffer overflow exploits, and reconnaissance scans. It leverages auxiliary modules to simulate real-world penetration scenarios.
- **Hping3 Tool**: Utilized to perform customizable DDoS attacks. Hping3 provides granular control over packet characteristics such as size, frequency, and protocol type, enabling simulation of various flood-based attacks.

2. OPNsense Firewall: Traffic Filtering and Protection

The OPNsense Firewall acts as the first defensive layer in the system. It provides:

- Stateful packet inspection.
- Rule-based traffic filtering.
- Deep packet inspection across multiple OSI layers.
- Session tracking and real-time traffic filtering to block suspicious and unauthorized network packets.

3. Suricata IPS: Intrusion Prevention and Detection

Suricata is the core intelligence component of the system. It offers:

- Real-time intrusion detection and prevention using signature-based analysis.
- Protocol anomaly detection and behavioral monitoring.
- Multi-threaded packet processing for high-performance inspection.
- Continuous log generation in eve.json for detailed event tracking and analysis.

4. Python Scripts: Log Analysis and Reporting

Our custom-developed Python scripts perform:

- Real-time log parsing from Suricata's eve.json file.
- Automated detection of security alerts and trends.
- Correlation and visualization of attack patterns.
- Automated generation of security reports and executive summaries for the Admin user.

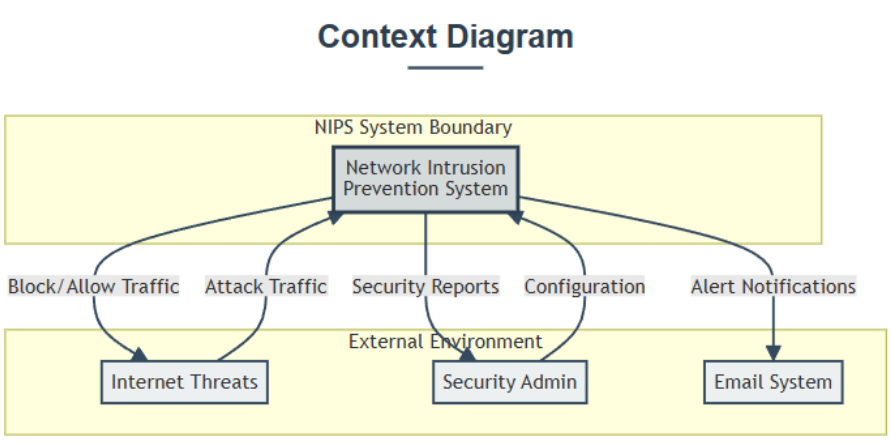
Data Flow Architecture:

1. **Traffic Generation**:
   The attacker initiates traffic (including DoS and DDoS attacks) towards the firewall.
2. **Firewall Filtering**:
   OPNsense inspects and filters incoming traffic, forwarding only allowed packets to the IPS.
3. **Intrusion Detection**:
   Suricata analyzes the filtered traffic in real-time, generating alerts for any detected suspicious activities.
4. **Log Management**:
   Suricata continuously logs security events in eve.json in JSON format.
5. **Log Analysis and Reporting**:
   Python scripts read the logs, process security events, and generate alerts and visual reports.
6. **Administrator Access**:
   The Admin receives real-time alerts and detailed security reports for monitoring and decision-making.

System Outputs:

- **Real-Time Alerts**: Instant notifications for detected security incidents.
- **Detailed Security Reports**: Trend analysis, top attackers, attack types, and comprehensive logs presented in readable formats and visual dashboards.

2.5. Context Diagram



Context Diagram: Shows the NIPS system as a single entity and its interactions with external entities including threats from the internet, security administrators, and notification systems.

The Context Diagram presents a high-level overview of our Network Intrusion Prevention System (NIPS), clearly illustrating its interactions with external entities and defining the system boundaries that separate internal processes from the outside world. This diagram is critical for understanding how

stakeholders, attackers, and supporting systems communicate with NIPS, and for accurately determining the system's scope.

External Entity Analysis:

1. Attacker VM: Threat Source

The Attacker VM represents simulated external threats using:

- **Metasploit Framework** for penetration testing and exploitation.
- **Hping3** for DDoS traffic generation.

These tools simulate:

- Denial-of-service attacks.
- Distributed denial-of-service (DDoS) scenarios.
- Exploit delivery and network scanning.

2. Security Administrator: System Controller

The Security Administrator is the human operator responsible for:

- Configuring the NIPS.
- Monitoring security events.
- Receiving automated security reports and alerts.
- Performing incident response activities.

3. Monitoring VM: Log Processing Unit

This VM continuously reads and processes Suricata logs using:

- Custom Python scripts for:
  - Real-time log analysis.
  - Alert correlation.
  - Security report generation.

4. Email / Notification System: Alert Delivery

The notification system ensures:

- Real-time alerts are delivered to the Security Administrator via email or other channels.
- Rapid dissemination of critical security incidents to enable fast response.

System Boundary Definition:

The **NIPS System Boundary** clearly defines what is internal to our solution:

- OPNsense Firewall.
- Suricata IPS.
- Python log analysis scripts.
- Local log storage (eve.json).

External entities such as the Attacker VM, Security Administrator, Monitoring VM, and Notification System are positioned outside this boundary.

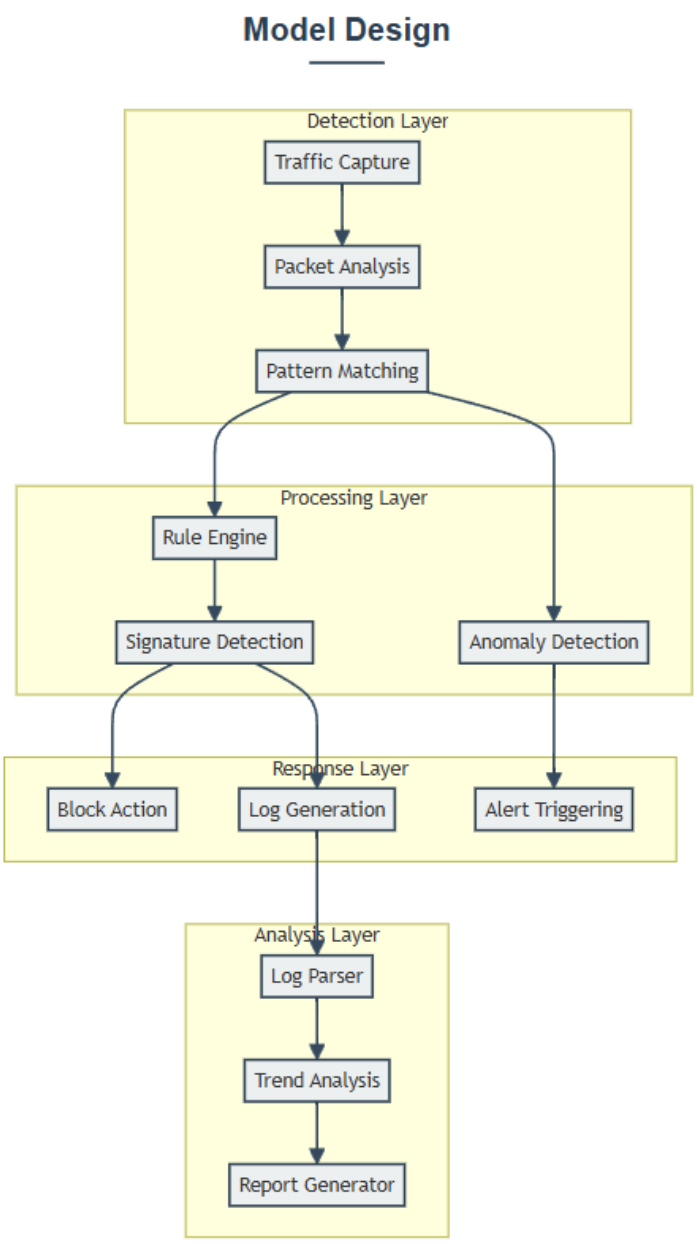
Data Flow Analysis:

1. **Attack Traffic**:
    - Continuous network traffic from the Attacker VM to the Firewall and IPS.
    - Includes both legitimate and malicious traffic requiring real-time inspection.
2. **Alert Notifications**:
    - Generated by Python scripts based on Suricata alerts.
    - Sent to the Security Administrator via the Email/Notification system.
3. **Security Reports**:
    - Generated automatically by the Python scripts.
    - Provide detailed summaries, top attack trends, and system logs.
4. **Configuration Data**:
    - Sent from the Security Administrator to the NIPS.
    - Includes firewall rules, IPS signatures, alert thresholds, and reporting parameters.

Key Features Highlighted:

- **Real-Time Traffic Monitoring**.
- **Automated Alerting** to the Security Administrator.
- **Detailed Security Reports** for strategic analysis.
- **Bidirectional Configuration Control** between the Admin and NIPS.
- **Support for Multiple Notification Channels** to ensure incident delivery.

2.6. Model Design



Model Design: Four-layer architecture showing how our NIPS processes network traffic from initial detection through final reporting and analysis.

- The Model Design diagram illustrates the multi-layered architecture of our Network Intrusion Prevention System (NIPS), highlighting how each processing layer contributes to detecting, analyzing, and mitigating network threats. The system is divided into four main functional layers that work together to ensure robust network security.

Detection Layer:

- Traffic Capture:
  This component captures incoming and outgoing network packets efficiently with minimal packet loss, even under high traffic loads. Packet capturing supports various network interfaces and enables packet mirroring for forensic analysis. It ensures accurate timestamping and handles both complete headers and configurable payloads while complying with privacy policies.

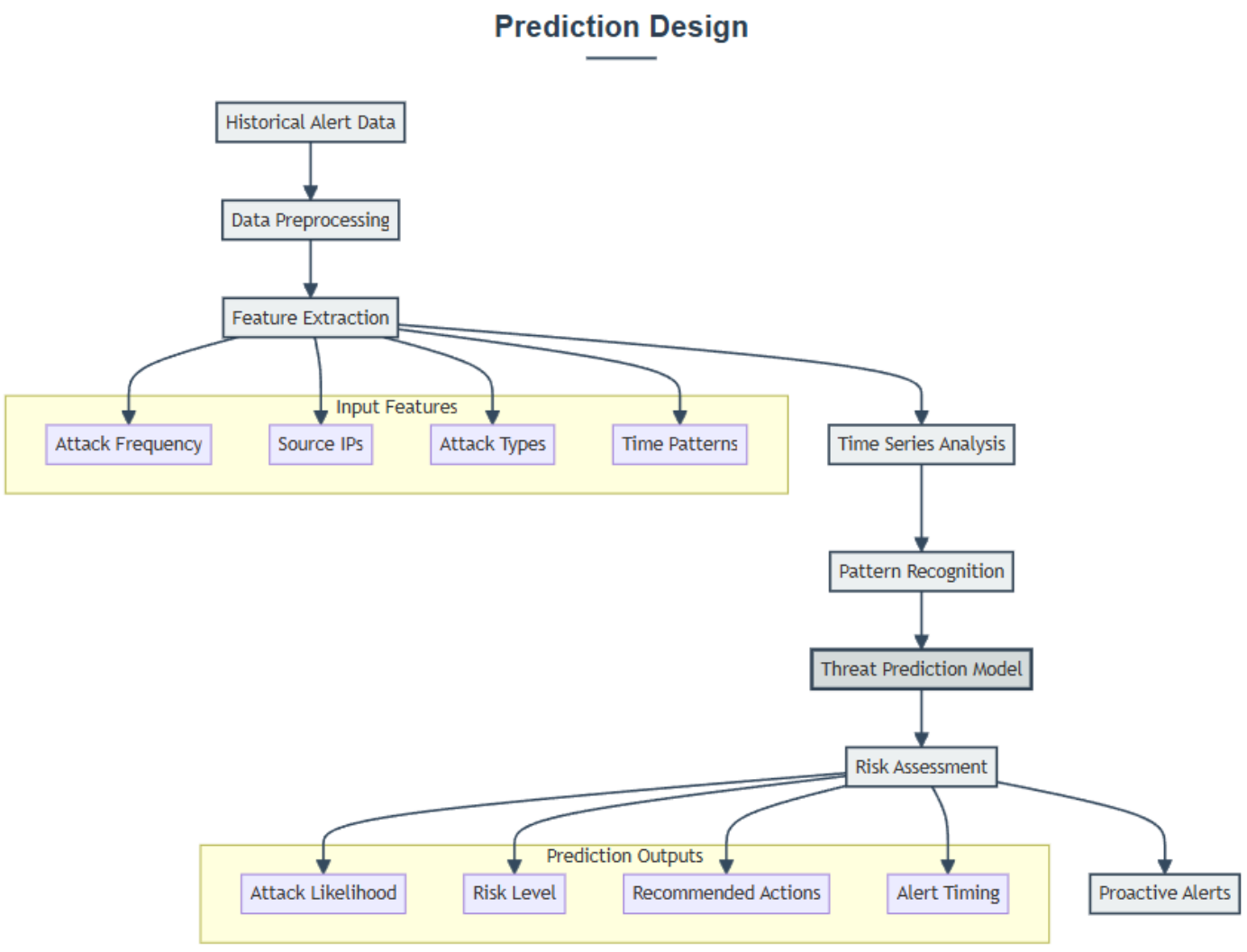
- Packet Analysis:
  The captured traffic undergoes preliminary inspection, including protocol identification, packet validation, header parsing, and fragmentation handling. This layer supports detection of malformed packets, traffic classification, and extraction of critical metadata.
- Pattern Matching:
  The core detection engine compares network traffic against known threat signatures and behavioral patterns using string matching, regular expressions, and statistical analysis. It maintains a dynamic database of malware signatures, exploit patterns, and suspicious communication behaviors.
- ---

  Processing Layer:
- Rule Engine:
  This component applies complex security rules using conditional logic, Boolean operators, threshold settings, and time-based correlations. Rules can be customized to match network structure and risk profiles.
- Signature Detection:
  Suricata uses up-to-date threat signatures to detect malware, known exploits, reconnaissance activity, and other malicious behaviors.
- Anomaly Detection:
  Advanced threat detection methods using statistical analysis and behavioral baselines identify unknown or zero-day attacks that traditional signature-based methods may miss.
- ---

  Response Layer:
- Block Action:
  The system can automatically block malicious traffic using TCP resets, packet drops, or IP blocking. It can also coordinate with upstream firewalls for broader network protection.
- Log Generation:
  All security events are recorded with detailed packet data, detection results, and system responses. Logs are stored in structured formats, supporting reliable retrieval and regulatory compliance.
- Alert Triggering:
  The system sends real-time alerts through prioritized notification channels (such as email or dashboards) to ensure rapid incident response.

- Analysis Layer:
- Log Parser:
  Python scripts continuously read Suricata's logs (eve.json), normalize data, and extract meaningful insights.
- Trend Analysis:
  Analyzes historical security data to identify repeated patterns, track attack sources, and detect changes in attack frequency.
- Report Generator:
  Automatically produces detailed security reports, including executive summaries, technical logs, and compliance documents, using customizable templates

2.7. Prediction design



Prediction Design: Machine learning-based prediction system that analyzes historical attack patterns to forecast potential security threats and enable proactive defense measures.

The Prediction Design introduces a forward-thinking, machine learning-based security layer that enhances the NIPS by forecasting potential future threats based on historical data patterns.

Detailed Workflow:

1. Historical Data Collection:
   - Aggregates historical security event logs from Suricata and traffic archives.
   - Builds a comprehensive dataset including source IPs, attack frequencies, and traffic characteristics.
2. Data Preprocessing:
   - Cleans raw data to remove inconsistencies and duplicates.
   - Normalizes data for consistency across different log sources.
   - Handles missing values and outlier detection.
3. Feature Extraction:
   - Identifies predictive features such as:
     - Source IP behavior over time.
     - Commonly attacked ports and services.
     - Time-of-day attack frequencies.
     - Packet size distributions.
4. Pattern Recognition:
   - Applies time series analysis to detect repetitive attack patterns.
   - Uses statistical modeling to uncover high-risk periods and sources.
   - Implements clustering techniques to group similar attacks.

5. Machine Learning Model:

- Employs classification algorithms (e.g., Random Forest, SVM) to forecast likely attack vectors.
- Trains on labeled historical data to distinguish benign from malicious traffic patterns.

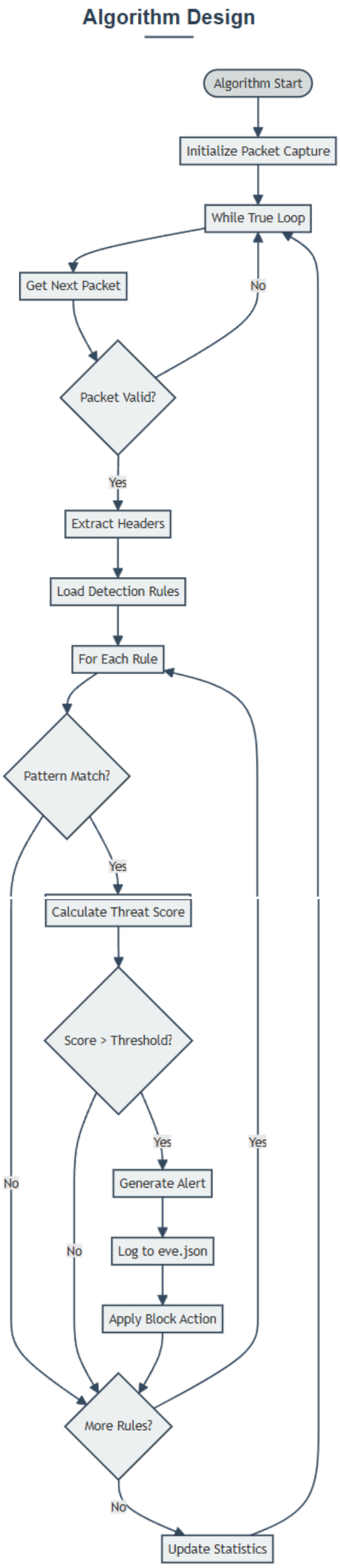
6. Predictive Alerts:

- Generates proactive security warnings for expected attacks.
- Suggests dynamic adjustments to detection thresholds and firewall rules based on predicted threats.

7. Continuous Learning:

- Continuously updates the predictive model with new incoming data.
- Improve forecast accuracy over time through adaptive learning.

2.8. Algorithm design



Algorithm Design: Flowchart showing the core detection algorithm that processes each network packet through rule matching and threat scoring to determine appropriate actions.

The **Algorithm Design** presents the core threat detection logic implemented within the NIPS. This flowchart describes a systematic, multi-layered decision-making process that ensures efficient packet evaluation, accurate threat identification, and appropriate automated responses.

Detailed Algorithm Workflow:

1. Packet Capture:

- Real-time capture of network packets from monitored interfaces.
- Supports deep packet inspection and multi-threaded processing.

2. Packet Validation:

- Ensures packet headers and protocol structures are intact.
- Drops corrupted, malformed, or incomplete packets.
- Validates packet source and destination addresses.

3. Traffic Classification:

- Identifies packet type:
  - Normal traffic.
  - Suspicious traffic.
  - Potentially malicious traffic.

4. Signature Matching:

- Compares packet against known threat signatures.
- Supports exact pattern matching, regex-based analysis, and composite signatures.

5. Anomaly Detection:

- Detects deviations from established baseline behaviors.
- Evaluates traffic volume, timing, and source reputation.

6. Threat Scoring:

- Assigns a severity score based on:
  - Threat type.
  - Matching rule priority.
  - Impact assessment.

7. Response Decision:

- If the threat score exceeds the configured threshold:
  - Block traffic (drop packet, TCP reset, IP ban).
  - Trigger alerts to administrators.
  - Log detailed event data.

- If the traffic is safe:
    - Allow communication to proceed.

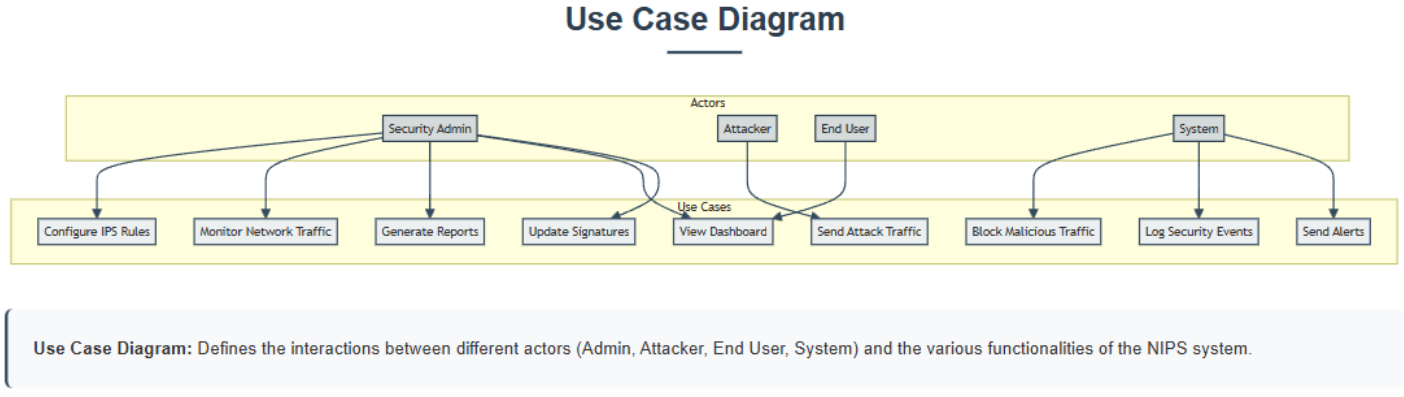
8. Logging:

- Records all security decisions, packet details, and response actions.
- Uses Suricata's eve.json structured format for real-time processing.

9. Notification:

- Sends real-time alerts to security personnel via multiple channels.
- Supports alert prioritization, escalation, and acknowledgment tracking.

2.9. Use Case Diagram



Use Case Diagram: Defines the interactions between different actors (Admin, Attacker, End User, System) and the various functionalities of the NIPS system.

The Use Case Diagram defines all functional interactions and system capabilities within our Network Intrusion Prevention System (NIPS), highlighting the roles of different users and automated system processes.

Actors:

1. Security Administrator:

The primary system operator responsible for:

- Configuring IPS rules.
- Monitoring network traffic.
- Managing incident responses.
- Generating security reports.
- Updating threat signatures.

2. Attacker:

Represents external or internal threat actors attempting to:

- Exploit vulnerabilities.
- Launch denial-of-service or distributed denial-of-service (DoS/DDoS) attacks.
- Perform network reconnaissance and exploit delivery.

3. End User:

Authorized personnel such as security analysts or network operators who:

- View security dashboards.
- Access system-generated reports.
- Monitor network status without administrative privileges.

4. System (Automated Processes):

Internal system components that:

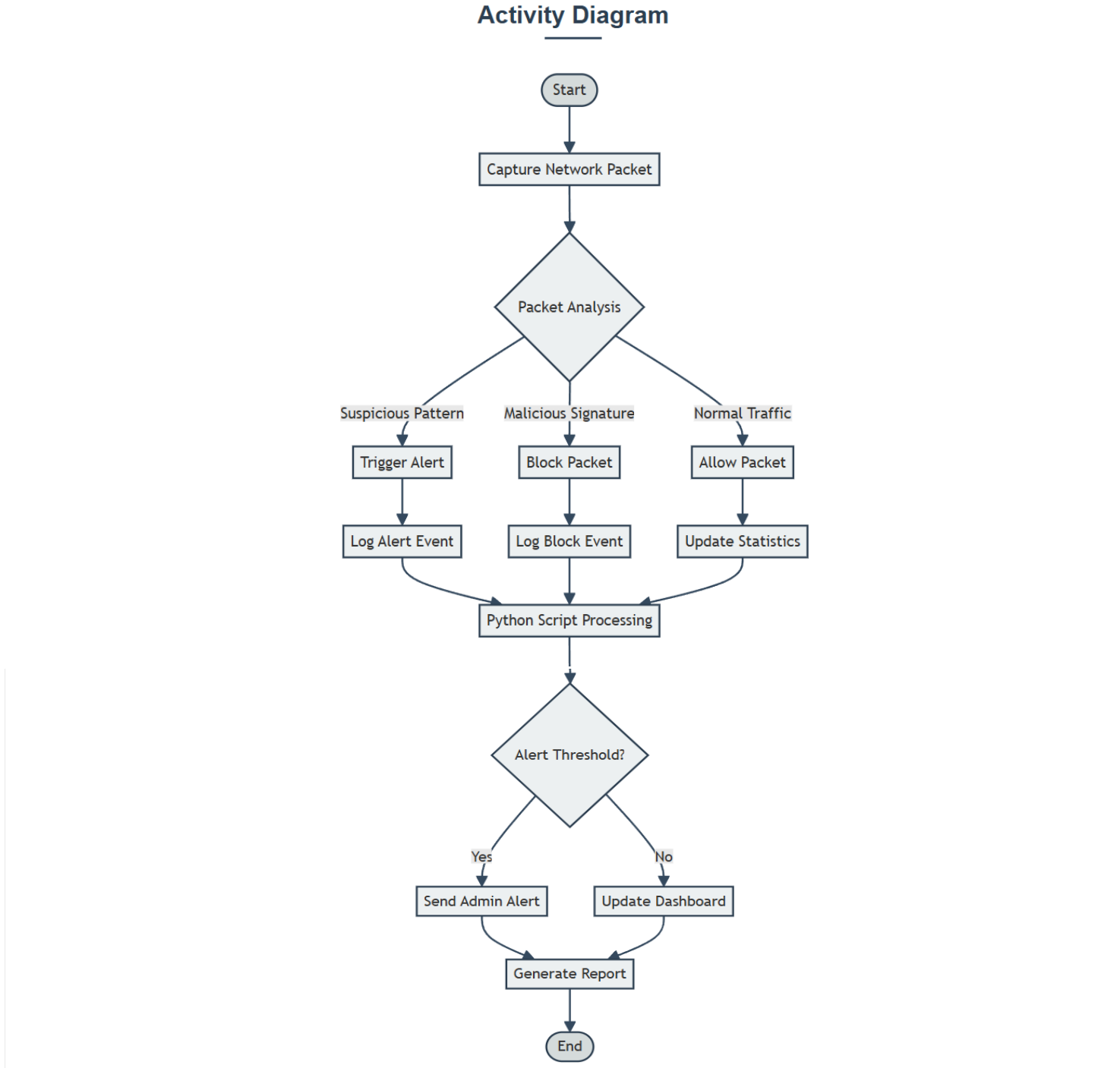
- Perform log analysis.
- Trigger alerts.
- Automatically block malicious traffic.
- Generate scheduled security reports.

Key Use Cases:

- Configure IPS Rules:
  Allows the administrator to define and manage detection policies, thresholds, and system behavior.
- Monitor Network Traffic:
  Provides real-time traffic visibility and alert monitoring through dashboards and detailed logs.
- Generate Reports:
  Automatically creates detailed security reports, compliance documents, and trend analyses for various stakeholders.
- Send Attack Traffic:
  Simulated attacker behavior that tests the system's detection and blocking capabilities.
- Block Malicious Traffic:
  Automated system response to identified threats, including IP blocking and traffic dropping.
- Log Security Events:
  All security incidents and system activities are logged in detail for auditing and forensic purposes.
- Send Alerts:
  Real-time alerts are automatically sent to the administrator and relevant personnel to support immediate action.
- View Dashboard:
  Security dashboards provide a visual representation of active threats, system health, and log summaries.
- Update Signatures:
  Administrators regularly update detection signatures to keep the system aligned with the latest threat intelligence.

- System Integration:
- The use cases are interconnected, demonstrating:
- How updated signatures influence detection accuracy.
- How attack traffic triggers alerts and system responses.
- How logs feed into reporting and dashboard displays.
- How automated responses minimize administrator workload

2.10. Activity Diagram



Activity Diagram: Shows the workflow of packet processing from capture to final reporting, including decision points for blocking, alerting, and reporting.

- The Activity Diagram presents the detailed workflow of our Network Intrusion Prevention System (NIPS), illustrating how network traffic is systematically captured, analyzed, and processed from initial packet arrival through final reporting and administrator notification. This workflow ensures real-time threat detection, automated decision-making, and optimized system performance.

- Workflow Stages:
- 1. Initial Packet Capture:
- Continuous packet capturing from multiple network interfaces using high-performance mechanisms.
- Handles high traffic volumes while preserving packet integrity and ensuring accurate timestamping.
- Implements advanced buffer management and load balancing across processing threads.

- 2. Packet Analysis Decision Point:
- Verifies packet validity, structure, and protocol compliance.
- Invalid or malformed packets are logged for further forensic investigation.

- 3. Traffic Classification:
- Analyzes packet attributes including:
- Source and destination IP addresses.
- Port numbers and protocols.
- Packet sizes and timing patterns.
- Traffic is classified as normal, suspicious, or malicious.

- 4. Normal Traffic Handling:
- Legitimate traffic is forwarded with minimal delay.
- Normal traffic is logged to establish network baselines and support future anomaly detection.

- 5. Suspicious Pattern Detection:
- Suspicious traffic triggers deeper analysis:
- Behavioral monitoring.
- Payload inspection.
- Threat intelligence correlation.
- Generates preliminary alerts while continuing inspection.

- 6. Malicious Signature Matching:
- Immediate threat detection through signature-based matching.
- Detects malware, DDoS patterns, known exploits, and C2 communications.
- Triggers automatic blocking and high-priority alerts.

- 7. Logging and Forensic Records:
- All events are logged in eve.json with detailed packet data, detection logic, and system responses.
- Logs are optimized for integrity, storage efficiency, and rapid retrieval.

- 8. Python Script Real-Time Processing:
- Custom Python scripts continuously monitor Suricata logs.
- Perform:
- Real-time correlation.
- Trend analysis.
- Automated security report generation.

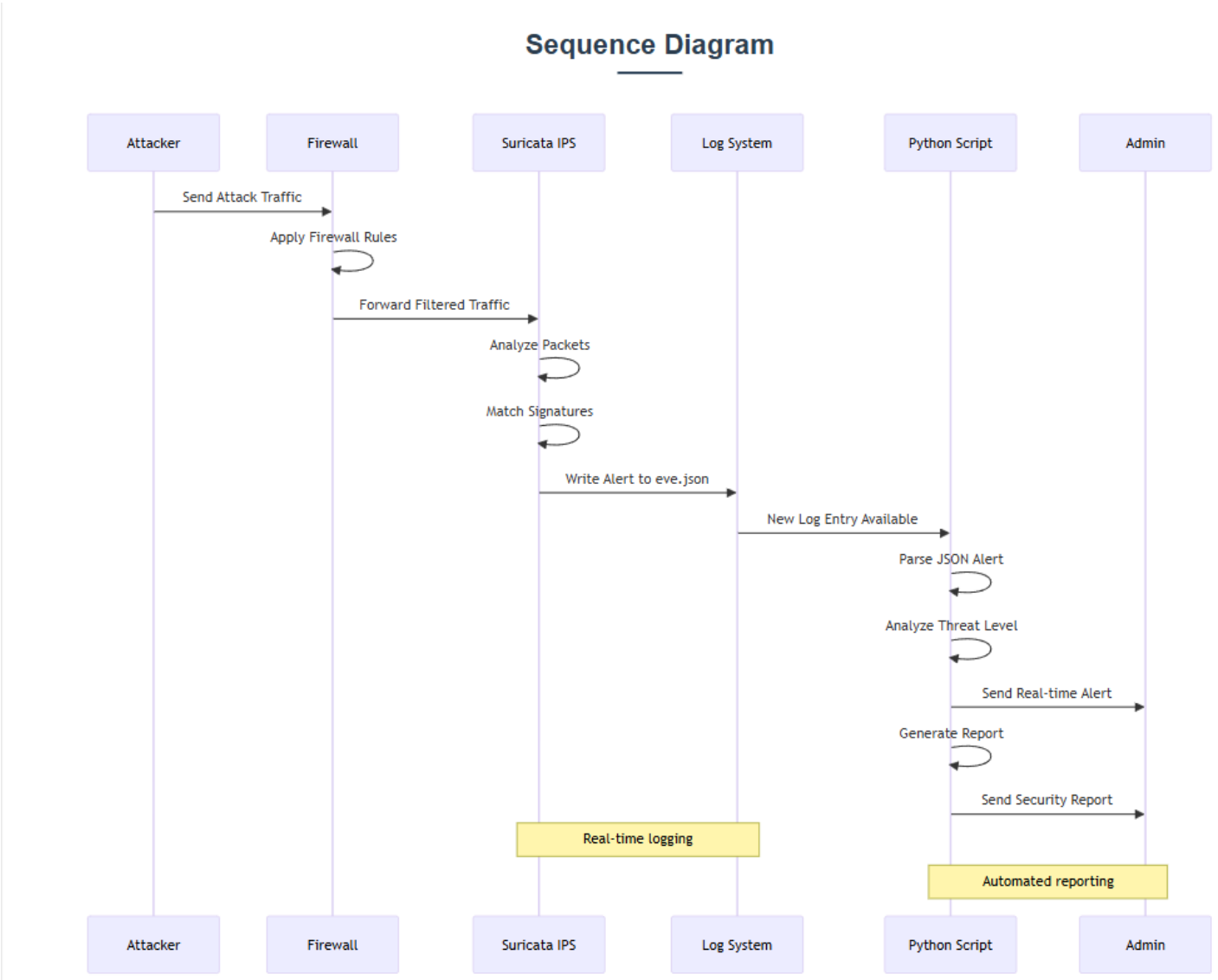
- 9. Alert Threshold Management:
- Dynamic alert thresholds balance detection sensitivity with operational efficiency.
- Prioritizes critical alerts while reducing false positives.

- 10. Administrative Notification:
- Immediate alerts are sent to security administrators via email, SMS, and dashboard updates.
- Supports escalation procedures and acknowledgment tracking.

- 11. Dashboard and Reporting:
- Real-time dashboards display current security status.
- Automated periodic reports summarize trends, incidents, and system performance.

- 12. Workflow Optimization:
- Implements parallel processing, caching, and priority queues.
- Maintains low latency for legitimate traffic while enabling thorough threat analysis.

2.11. Sequence Diagram



Sequence Diagram: Timeline showing the interaction between system components when processing an attack, from initial traffic to final admin notification.

The Sequence Diagram demonstrates the chronological flow of communication between system components during a complete attack detection and response cycle. It highlights how the Network Intrusion Prevention System (NIPS) components interact in real-time to ensure rapid threat detection, response, and reporting.

Workflow Stages:

1. Attacker to Firewall Communication:

The attacker VM sends malicious traffic (e.g., DDoS packets, exploits, reconnaissance scans) toward the firewall.

2. Firewall Processing:

OPNsense Firewall:

Filters traffic based on access control rules.

Blocks known bad IPs and malformed traffic.

Forwards filtered packets to the Intrusion Prevention System (IPS).

3. IPS Deep Packet Inspection:

Suricata IPS:

Performs multi-layer inspection.

Detects protocol anomalies, signature-based threats, and application-level attacks.

Maintains stateful session tracking for complex attacks.

4. Logging System:

Suricata logs security events in real-time to eve.json.

Logs include metadata, packet details, detection results, and suggested actions.

5. Python Script Real-Time Processing:

Python scripts monitor eve.json using real-time file tracking.

Perform:

Log parsing.

Event correlation.

Alert prioritization.

6. Intelligent Alert Analysis:

Correlates multiple events to detect coordinated attack campaigns.

Assigns threat severity and tracks attack progression.

7. Administrative Notification:

Sends real-time alerts via:

Email (detailed incident information).

SMS (critical priority).

Dashboard updates (real-time security status).

8. Reporting Generation:

Automatically generates:

Executive summaries.

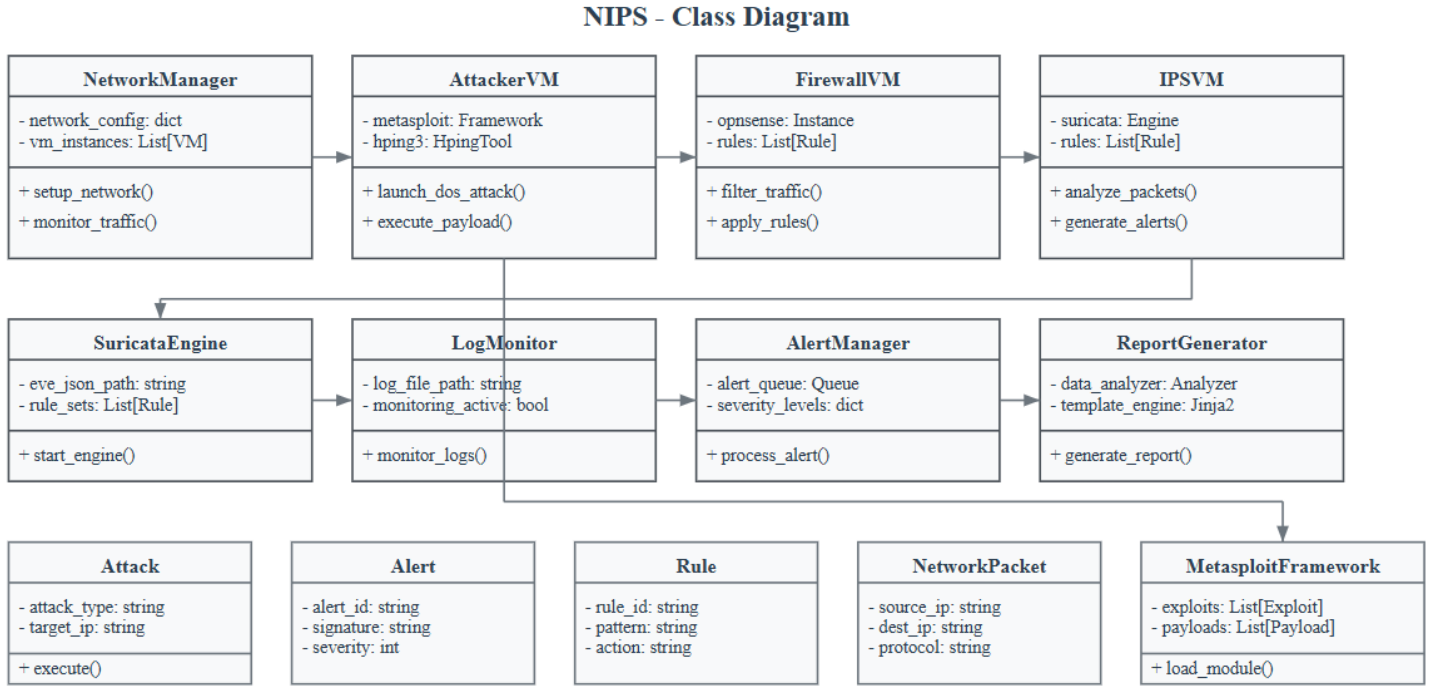
Technical incident reports.

Daily operational reports.

9. System Feedback Loop:

Admin actions (e.g., new rules, blocked IPs) feed back into the firewall and IPS configurations to continuously improve detection and response.

2.12. Class Diagram



The Class Diagram provides a detailed object-oriented architectural design for the Python-based Network Intrusion Prevention System (NIPS). It demonstrates the modular, maintainable, and scalable structure that supports efficient log processing, event correlation, alert management, real-time monitoring, and automated reporting.

The design uses separation of concerns, where each class handles a specific task in the security pipeline, and employs advanced software engineering design patterns such as Observer, Factory, Singleton, and Strategy to ensure system reliability and future extensibility.

Core Classes and Responsibilities:

1. NetworkPacket Class

- **Purpose: Represents individual packets captured by the NIPS system.**
- **Attributes:**
  - **source_ip**
  - **destination_ip**
  - **source_port**
  - **destination_port**
  - **protocol**
  - **payload**
  - **timestamp**
- **Methods:**
  - **validate_packet(): Ensures packet structure and integrity.**
  - **inspect_payload(): Performs deep packet inspection.**
  - **extract_signature(): Identifies key threat indicators.**
  - **reassemble(): Handles fragmented packet reassembly.**
- **Notes:**
  - **Supports both IPv4 and IPv6 traffic.**
  - **Enables protocol-specific validation (e.g., TCP, UDP, ICMP).**

---

2. SuricataAlert Class

- **Purpose: Represents security events detected by Suricata.**
- **Attributes:**
  - **alert_id**
  - **signature**
  - **severity**
  - **timestamp**
  - **source_ip**
  - **destination_ip**
  - **protocol**
- **Methods:**
  - **validate_alert(): Ensures completeness and correctness of log entries.**
  - **correlate_alerts(): Matches related alerts across sessions.**
  - **escalate_alert(): Triggers high-priority notifications.**

- **Notes:**
    - **Supports integration with external threat intelligence.**

---

3. LogParser Class

- **Purpose: Handles log file reading and parsing.**
- **Attributes:**
    - **log_path**
    - **format_type (JSON, CSV, etc.)**
- **Methods:**
    - **read_log(): Reads new log entries in real-time.**
    - **parse_entry(): Extracts and normalizes security event data.**
    - **batch_process(): Processes large log files efficiently.**
- **Notes:**
    - **Supports file monitoring (inotify) and stream processing.**

---

4. ReportGenerator Class

- **Purpose: Generates visual and text-based security reports.**
- **Attributes:**
    - **report_id**
    - **generated_date**
    - **format_type (HTML, PDF, CSV)**
- **Methods:**
    - **create_summary(): Produces high-level executive reports.**
    - **generate_charts(): Visualizes trends and attack frequencies.**
    - **export_report(): Saves reports to designated paths.**
- **Notes:**
    - **Supports customizable report templates.**

---

5. AlertManager Class

- **Purpose: Manages multi-channel alert distribution.**
- **Attributes:**
    - **alert_queue**
    - **contact_list (email, SMS recipients)**
- **Methods:**
    - **send_email(): Sends detailed alert messages.**
    - **send_sms(): Sends high-priority SMS notifications.**
    - **escalate_alert(): Applies escalation rules.**
    - **track_acknowledgment(): Monitors alert responses.**

- **Notes:**
  - **Supports multi-layer escalation and alert grouping.**

---

6. TrafficMonitor Class

- **Purpose: Continuously monitors live network traffic.**
- **Attributes:**
  - **interface**
  - **monitoring_mode**
- **Methods:**
  - **capture_packets(): Captures traffic in real-time.**
  - **filter_suspicious(): Filters based on configured rules.**
- **Notes:**
  - **Integrates with Suricata or raw packet capture tools.**

---

7. ThresholdManager Class

- **Purpose: Dynamically manages alert thresholds and system sensitivity.**
- **Attributes:**
  - **current_threshold**
  - **adjustment_policy**
- **Methods:**
  - **update_thresholds(): Adjusts thresholds based on system load and false positive rates.**
  - **apply_dynamic_sensitivity(): Balances security accuracy and performance.**
- **Notes:**
  - **Supports automated and manual adjustments.**

---

8. AdminConfig Class

- **Purpose: Stores system-wide configuration settings.**
- **Attributes:**
  - **config_parameters**
  - **last_updated**
- **Methods:**
  - **load_config(): Loads system settings at startup.**
  - **update_config(): Allows dynamic reconfiguration.**
- **Notes:**
  - **Manages notification channels, log paths, and rule update schedules.**

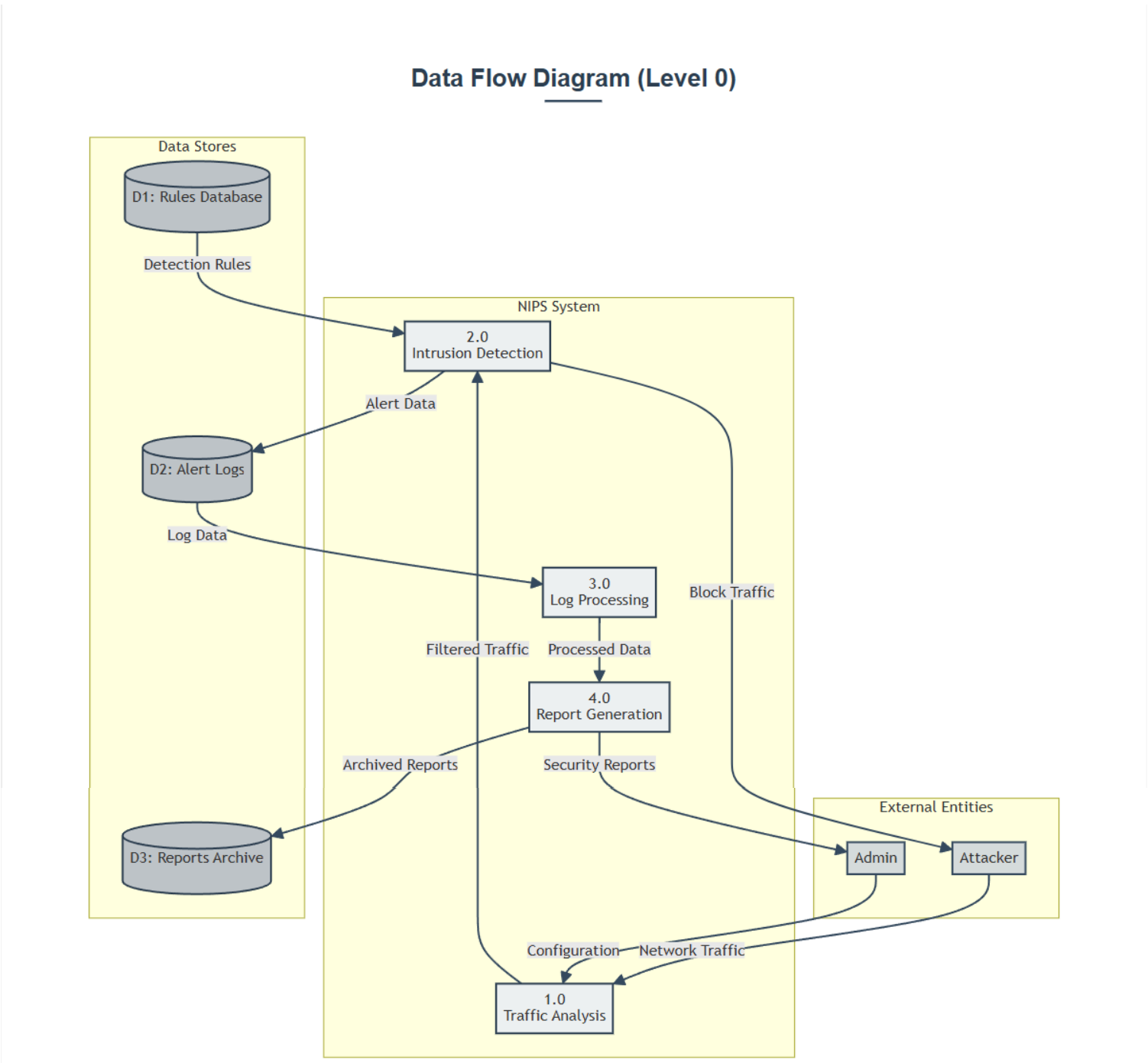
Design Patterns Applied:

- **Observer Pattern: Used for real-time log monitoring and alert triggering.**
- **Factory Pattern: Used for report generation across different output formats.**
- **Singleton Pattern: Used in LogParser to ensure a single active log reading instance.**
- **Strategy Pattern: Used for dynamic threshold management and response actions.**

Class Relationships:

- **Aggregation:**
  - **AlertManager aggregates multiple SuricataAlert instances.**
  - **ReportGenerator aggregates alerts and network traffic summaries.**
- **Association:**
  - **LogParser is associated with SuricataAlert to parse event logs.**
  - **NetworkPacket is associated with TrafficMonitor for real-time capture.**
- **Composition:**
  - **AdminConfig is composed within all classes to provide centralized configuration access.**

2.13. DFD (Data Flow Diagram)



Data Flow Diagram: Shows how data flows through the system processes, from traffic analysis to report generation, including data storage points.

The Level 0 Data Flow Diagram (DFD) provides a comprehensive and high-level visualization of the complete data architecture and transformation processes within our Network Intrusion Prevention System (NIPS). This diagram effectively illustrates the systematic flow of data, starting from initial traffic generation to final security reporting, while also showing how external entities, internal processes, and data storage components interact to maintain a secure network environment.

External Entities - In-Depth Analysis

1. Attacker: Threat Source Characterization

The Attacker entity represents a broad spectrum of potential threat actors that the NIPS must continuously monitor and defend against. These include:

- Advanced Persistent Threats (APTs): Well-funded, highly skilled groups targeting sensitive infrastructure through multi-phase attacks.
- Organized Cybercrime Groups: Entities focusing on financial gain through data theft, ransomware deployment, and fraudulent activities.
- Nation-State Actors: Entities driven by espionage and cyberwarfare strategies aiming to destabilize or surveil critical assets.
- Script Kiddies: Amateur hackers leveraging publicly available tools to conduct opportunistic attacks with limited sophistication.
- Insider Threats: Authorized users misusing access to compromise system integrity.

The attack traffic generated by these entities varies from:

- Network Layer Attacks: Exploiting routing protocols, TCP/IP stack weaknesses, or network services.
- Application Layer Attacks: Targeting web applications, databases, and APIs.
- Social Engineering Attacks: Exploiting human vulnerabilities through phishing, baiting, or impersonation.
- Hybrid and Multi-Vector Attacks: Combining several techniques to maximize impact and evade defenses.

---

2. Administrator: Multi-Role Stakeholder

The Administrator entity represents multiple organizational roles critical to maintaining and leveraging the NIPS system:

- SOC Analysts: Responsible for continuous monitoring, threat triage, and first-line incident response.
- Network Administrators: Focused on managing firewall configurations, IPS tuning, and ensuring optimal network performance.

- Compliance Officers: Ensure that the system adheres to regulatory standards and prepares evidence for audits.
- Executive Decision Makers: Require strategic security insights, risk assessments, and high-level security summaries to inform business decisions.
- Incident Response Teams: Collaborate to contain, eradicate, and recover from cyber incidents.

Each role interacts with the NIPS through different dashboards, alert formats, and reporting levels of detail to support operational, tactical, and strategic decision-making.

Process Decomposition and Functional Workflow

1. Process 1.0 – Traffic Analysis:

The Traffic Analysis process is responsible for continuous real-time monitoring of all network communications passing through the system. It includes:

- Protocol inspection and validation.
- Bandwidth utilization analysis.
- Communication pattern profiling (source, destination, timing, frequency).
- Deep packet inspection to identify application-specific behaviors.
- Behavioral profiling to detect unusual traffic trends.

The process supports:

- Encrypted Traffic Analysis: Identifying suspicious encrypted channels.
- Baseline Establishment: Differentiating normal network traffic from anomalies.
- Capacity Planning: Providing network performance statistics to guide infrastructure expansion.

---

2. Process 2.0 – Intrusion Detection:

This is the core security engine of the NIPS that employs:

- Signature-Based Detection: Matches traffic patterns against known threat signatures.
- Anomaly-Based Detection: Identifies statistically significant deviations from established traffic baselines.
- Heuristic Detection: Applies expert-defined rules to identify suspicious behaviors.
- Machine Learning Models: Used to detect evolving, sophisticated, or unknown attack patterns.

The process correlates:

- Multiple attack phases.
- Different attack vectors across sessions.
- Events linked to broader coordinated attacks.

It assigns risk scores, detection priorities, and severity classifications to guide immediate security actions.

---

3. Process 3.0 – Log Processing:

The Log Processing function transforms large volumes of raw Suricata logs (eve.json) into structured, actionable data by:

- Normalizing log formats from diverse sources.
- Validating log completeness and accuracy.
- Correlating events to detect multi-stage attacks.
- Aggregating security events for efficient analysis and reporting.

The system supports:

- Real-Time Processing: Instant analysis of incoming logs.
- Batch Processing: Deep-dive historical analysis for trend evaluation.
- Data Retention Management: Ensuring logs comply with regulatory storage timelines.

Advanced processing features include indexing, caching, and duplicate suppression to optimize speed and accuracy.

---

4. Process 4.0 – Report Generation:

The Report Generation process produces:

- Operational Reports: Detailed real-time incident summaries for SOC teams.
- Strategic Reports: Trend analysis and attack summaries for executives.
- Compliance Reports: Formal documentation for legal, regulatory, and audit requirements.
- Forensic Reports: Detailed logs and evidence trails for incident investigation and potential legal proceedings.

The system supports:

- Automated report scheduling.
- Multi-format outputs (HTML, PDF, CSV, JSON).
- Customizable templates for different user roles.
- Interactive dashboards and drill-down reporting for deep analysis.

Data Store Architecture – Detailed Components

D1: Rules Database

The Rules Database is a critical component that stores:

- Detection signatures for malware, exploits, DDoS patterns, and C2 communications.
- Security policies and thresholds.
- System configuration parameters.

The database:

- Supports version tracking.
- Integrates external threat intelligence feeds.
- Optimizes rules for performance and minimal false positives.

D2: Alert Logs

The Alert Logs repository maintains:

- Detailed records of all detected security events.
- Metadata including event sources, timestamps, and response actions.
- Indexed storage for rapid querying.
- Integrity protection to ensure log authenticity.
- Automated backup and recovery mechanisms.

D3: Reports Archive

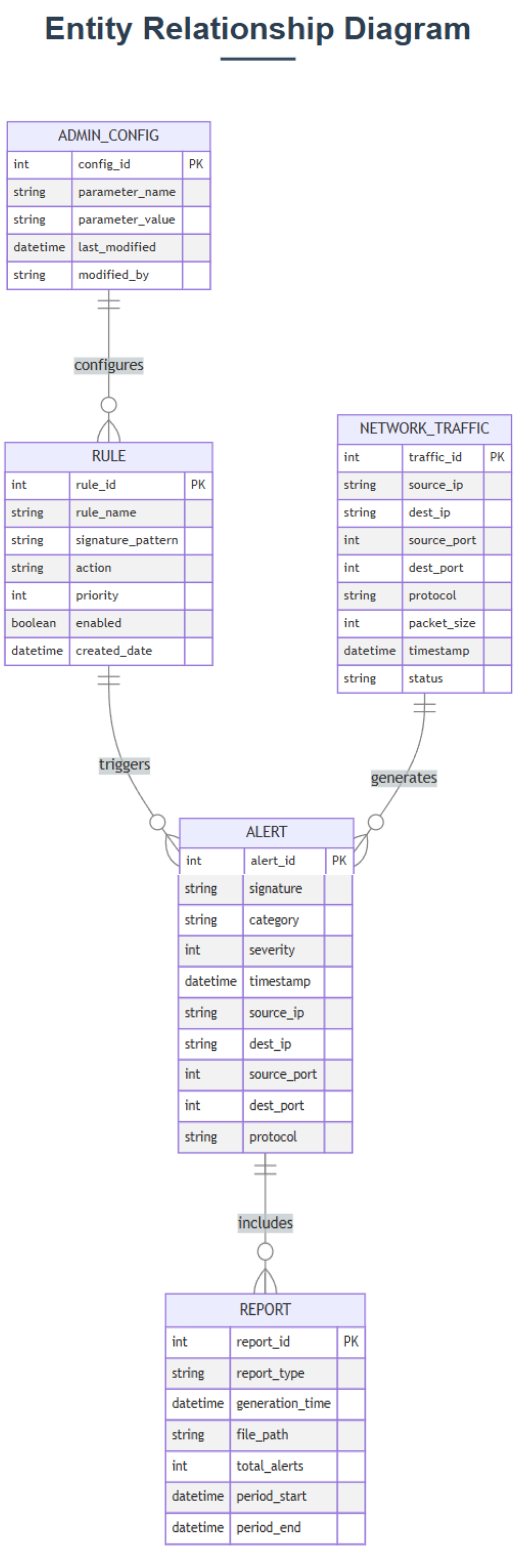
The Reports Archive securely stores:

- Generated reports across all time periods.
- Compliance-related documentation.
- Forensic summaries used in security investigations.
- Data for strategic trend analysis and executive reporting.

The archive supports:

- Efficient retrieval for audits.
- Historical comparisons to detect evolving threats.
- Multi-level access controls to protect sensitive data.

2.14. ERD (Entity Relationship Diagram)



Entity Relationship Diagram: Database schema showing the relationships between alerts, rules, reports, network traffic, and configuration data in our NIPS system.

The Entity Relationship Diagram (ERD) comprehensively illustrates the relational database structure designed to support efficient storage, management, and retrieval of security events and system configurations within the Network Intrusion Prevention System (NIPS).

This robust database schema is optimized for scalability, fast querying, and data integrity, enabling detailed forensic analysis, operational reporting, and long-term security trend evaluation.

Core Entities and Attributes:

1. ALERT Table:
   - Purpose: Stores detailed records of all security events detected by the NIPS.
   - Attributes:
     - alert_id (Primary Key): Unique identifier for each alert.
     - timestamp: Precise detection time for each security event.
     - signature: Description of the matched threat signature.
     - severity: Assigned threat level (e.g., low, medium, high).
     - source_ip and destination_ip: Captured source and target IP addresses.
     - source_port and destination_port: Network ports involved in the communication.
     - protocol: Protocol type (TCP, UDP, ICMP, etc.).
     - traffic_id: Foreign key linking to the NETWORK_TRAFFIC table.
     - response_action: Action taken (block, alert, allow, etc.).
   - Features:
     - Supports fast querying via indexes on IPs and timestamps.
     - Maintains an audit trail for forensic investigations.

---

2. RULE Table:
   - Purpose: Stores detection rules, policies, and security configurations.
   - Attributes:
     - rule_id (Primary Key): Unique identifier for each rule.
     - signature: Pattern or behavior the rule is designed to detect.
     - protocol: Network protocol the rule applies to.
     - priority: Rule priority (high, medium, low).
     - creation_date and last_updated: Tracks rule history.
     - enabled: Indicates whether the rule is active or disabled.
   - Features:
     - Supports rule versioning.
     - Integrates with external signature sources for updates.

---

3. REPORT Table:
   - Purpose: Stores all system-generated reports.
   - Attributes:
     - report_id (Primary Key): Unique report identifier.
     - generated_date: Date and time of report creation.
     - report_type: Classification (executive, operational, forensic, compliance).
     - summary: High-level overview of the report contents.
     - file_path: Location of the saved report file.
   - Features:
     - Enables historical tracking and retrieval for compliance purposes.
     - Supports multi-format reporting (PDF, HTML, CSV).

---

4. NETWORK_TRAFFIC Table:

- Purpose: Captures and stores raw traffic metadata for each network session.
- Attributes:
  - traffic_id (Primary Key): Unique session identifier.
  - source_ip and destination_ip: Full session path details.
  - source_port and destination_port: Ports used in communication.
  - protocol: Session protocol.
  - packet_count: Number of packets within the session.

- Features:
  - Supports session reconstruction.
  - Links directly to ALERT records for traceability.

---

5. ADMIN_CONFIG Table:

- Purpose: Stores system configuration settings managed by the administrator.
- Attributes:
  - config_id (Primary Key): Unique configuration entry.
  - parameter: Configuration key (e.g., alert threshold, reporting interval).
  - value: Parameter value.
  - last_updated: Timestamp of the last configuration change.
- Features:
  - Supports dynamic system tuning.
  - Provides a historical log of configuration adjustments.

Key Relationships:

- RULE to ALERT:
  One rule may trigger many alerts (one-to-many relationship).
- ALERT to NETWORK_TRAFFIC:
  Each alert is directly tied to specific network traffic for forensic traceability.
- ALERT to REPORT:
  Each report can include multiple alerts; reports aggregate related security events.
- ADMIN_CONFIG:
  Independently controls system settings affecting all processes.

Chapter Three

3.Deliverable and Evaluation

3.1. Intrusion Prevention System (IPS)

An **Intrusion Prevention System (IPS)** is a security mechanism that actively monitors network traffic to detect and prevent malicious activities. Unlike an Intrusion Detection System (IDS), which only alerts on potential threats, an IPS takes proactive measures such as dropping malicious packets, blocking traffic from suspicious IPs, and modifying firewall rules to stop threats in real time.

➢ Key functions of an IPS include:

Traffic analysis: Continuously scans incoming and outgoing packets for suspicious behavior or known attack signatures.

Threat detection: Uses signature-based, anomaly-based, or policy-based detection to identify threats.

Automatic prevention: Takes immediate action against threats, such as blocking IP addresses or terminating sessions.

Logging and alerting: Keeps records of all threats and actions taken for audit and analysis.

❖ IPS is crucial in a network security setup as it acts as a gatekeeper, preventing attacks like port scanning, brute force attempts, buffer overflow exploits, and more.

3.2. Suricata IPS

Suricata is a powerful, open-source, high-performance Network Threat Detection Engine. It can act as an IDS, IPS, and even a network security monitoring tool. In the context of your project, Suricata is used in **IPS mode** to actively block network attacks detected through traffic inspection.

➢ **Key features of Suricata as an IPS:**

Multi-threading support: Suricata can efficiently use multi-core systems for high-speed packet processing.

Deep packet inspection (DPI): Suricata examines the entire packet (including payload) to detect complex threats.

Signature-based detection: Compatible with Snort/VRT rules, enabling use of a wide database of known attack signatures.

Protocol identification: Automatically identifies protocols like HTTP, SSL, FTP, and others to enhance detection accuracy.

Integration with OPNsense: In your project, Suricata is integrated with OPNsense, a FreeBSD-based firewall and routing platform, enabling seamless blocking of malicious traffic.

❖ Using Suricata in IPS mode allows real-time threat prevention and detailed alerting/logging through the OPNsense interface.

3.3. Simulating a DoS Attack Using Metasploit

A **Denial of Service (DoS)** attack is designed to overwhelm a system or service with excessive traffic or requests, rendering it unavailable to legitimate users. To test the resilience of your IPS and firewall configuration, a simulated DoS attack can be performed using tools like **Metasploit**.

➢ **Steps typically involved:**

Target identification: You configure a virtual machine to act as the target (usually with a web server or open port).

Launching the attack: Using Metasploit modules like auxiliary/dos/tcp/synflood, you simulate a flood of TCP requests.

Observation: The firewall and Suricata IPS are expected to detect abnormal traffic patterns and block the attack.

Result analysis: Logs and dashboards in OPNsense and Suricata are reviewed to verify if the attack was successfully mitigated.

❖ This helps in validating your IPS's ability to handle single-source volumetric attacks.

3.4. Simulating a DDoS Attack Using Hping3

A Distributed Denial of Service (DDoS) attack is a coordinated cyber-attack designed to flood a target system or network with an overwhelming volume of traffic, effectively disrupting normal service availability. Unlike a simple DoS attack, a DDoS originates from multiple attack sources, often using botnets, which makes mitigation significantly more complex.

In our project, we simulate DDoS attacks using Hping3, a powerful and flexible packet crafting tool, in combination with Metasploit auxiliary modules to create realistic multi-source attack scenarios in a controlled lab environment.

Simulation Setup

1. Attack Sources: Multiple VMs and Parallel Sessions

We configure multiple attacker virtual machines (VMs) in our VMware environment to emulate a distributed attack network.

Hping3 is deployed on each attacker VM to generate synchronized high-volume traffic targeting the victim machine.

We also utilize parallel Metasploit auxiliary sessions for additional flood patterns, ensuring that the attack load closely resembles a real-world DDoS.

2. Hping3: The Core Attack Tool

Hping3 is extensively used for generating custom traffic floods with precise control over:

Packet rate (packets per second)

Packet size

Source IP spoofing

TCP flags (SYN, ACK, FIN, etc.)

ICMP, UDP, and raw IP packet generation

Sample Command for TCP SYN Flood using Hping3:

bash

CopyEdit

hping3 -c 100000 -d 120 -S -w 64 -p 80 --flood --rand-source <Target-IP>

-c 100000: Send 100,000 packets.

-d 120: Packet size 120 bytes.

-S: TCP SYN flag.

-w 64: TCP window size.

-p 80: Target port 80 (HTTP service).

--flood: Send packets as fast as possible.

--rand-source: Randomize source IPs to mimic a distributed attack.

This command overwhelms the target with SYN packets from random spoofed IP addresses, simulating a DDoS scenario.

3. Metasploit Integration

Metasploit auxiliary modules are used to simulate additional attack types such as:

auxiliary/dos/http/slowloris: Sends slow, incomplete HTTP requests to exhaust server resources.

auxiliary/dos/udp/udp_flood: Generates UDP floods targeting critical services.

Metasploit complements Hping3 by providing layer 7 (application layer) flooding techniques, while Hping3 handles layer 3/4 (network/transport layers).

4. High-Volume Traffic Generation Goals

The simulated DDoS traffic is intended to:

Consume CPU resources by overloading connection-handling processes.

Exhaust memory allocation via persistent connection requests.

Flood available bandwidth with large amounts of data.

This multi-layered attack forces the target system to degrade or become unavailable, closely replicating real-world attack consequences.

3.5. OPNsense Firewall

The OPNsense Firewall was implemented on a **dedicated VM** to serve as the first line of defense, applying traffic filtering, packet inspection, and session tracking.

Key Features:

- Stateful packet inspection to monitor connection states.
- Policy-based filtering to block known attack vectors.
- Integrated with Suricata for dynamic rule enforcement.
- Customizable network zones and interface segregation.

Evaluation:

- Successfully blocked malicious traffic flagged by the IPS.
- Maintained session integrity and connection tracking for legitimate users.
- Demonstrated high compatibility with Suricata for automatic defense coordination.
- Effectively logged blocked sessions and provided real-time policy enforcement.

Chapter Four

4. PRACTICAL IMPLEMENTATION

4.1. Virtual Environment Setup

The practical implementation of our Network Intrusion Prevention System (NIPS) was carried out within a **fully virtualized environment** using **VMware Workstation.** This setup provided a controlled, isolated network where attacks and defenses could be safely simulated and analyzed without impacting external systems.

4.1.1. VMware Configuration

We used **VMware Workstation Pro** to deploy and manage our virtual machines. This tool allowed us to easily configure network adapters, assign IP addresses, and create isolated test environments.

VMware Setup:

- **Number of Virtual Machines (VMs):** 3
- **Virtual Machines Created:**
    1. **Attacker VM (Kali Linux)**
    2. **Firewall VM (OPNsense)**
    3. **IPS VM (Suricata -Kali)**

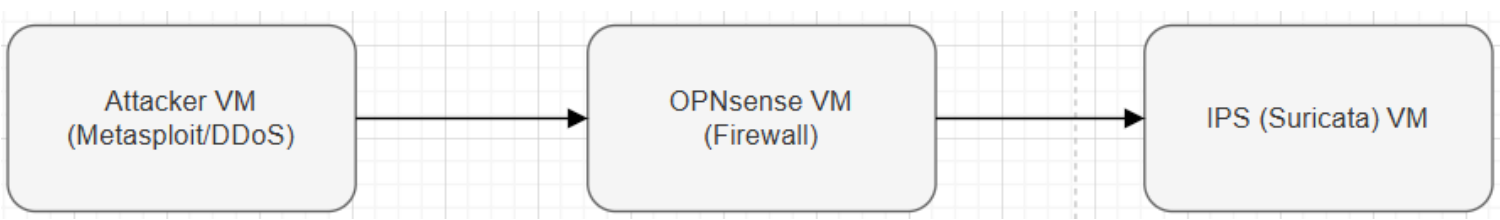
VMware Network Configuration:

- **Network Type:** Host-Only / Bridged (depending on the isolation needed)
- **Each VM was connected to the same virtual network** to allow controlled traffic flow.

4.1.2. Network Topology

The virtual network was structured to mirror a typical real-world network with an attacker, a firewall, an intrusion prevention system, and a victim.

Network Diagram:



- **Traffic Flow:**
    - Attacker VM → Firewall VM (OPNsense) → IPS VM (Suricata)
- **Connection Path:**
    - All attacker traffic is first filtered by the firewall, which then forwards allowed traffic to the IPS for deep packet inspection.

- **Management Access:**
  - Each VM can be accessed separately from the host for configuration and monitoring.

Key Design:

- **Segregated VMs to simulate real network segments.**
- **VMs are connected via virtual network interfaces, enabling full control over traffic direction and attack simulation.**

4.1.3. VM Roles and IP Assignments

Each virtual machine was assigned a specific role in the project with a unique static IP to ensure clear traffic flow and easy packet tracing.

VM Role	Operating System	Tool(s) Used	IP Address
Attacker	Kali Linux	Hping3, Metasploit	192.168.100.10
Firewall	OPNsense	Stateful Packet Inspection	192.168.100.1
Intrusion Prevention	Kali Linux	Suricata IPS, Python Scripts	192.168.100.10

VM Roles:

- **Attacker VM:** Launches DoS and DDoS attacks using Hping3 and Metasploit modules.
- **Firewall VM (OPNsense):** Acts as the first line of defense to filter and block malicious traffic.
- **IPS VM (Suricata):** Performs deep packet inspection, generates alerts, and logs traffic using custom Python scripts.

4.2. Routing Techniques

In our virtual lab environment, we implemented **static routing** to control the traffic flow between the virtual machines (VMs). Static routing was chosen to provide a **simple, predictable, and fully manageable routing structure** suitable for the controlled simulation of attacks and defense mechanisms within VMware.

4.2.1. Static Routing Configuration

What is Static Routing?

Static routing is a network routing method where **routes are manually configured** by the administrator instead of being dynamically discovered by routing protocols. In our project, static routes were set to **direct traffic through the OPNsense Firewall and the Suricata IPS** in a fixed path.

How We Configured Static Routes:

Each VM was manually configured with routing tables that define **the next-hop gateway** to reach other machines in the network.

Example Static Route Command:

On the Attacker VM:

bash

CopyEdit

sudo route add -net 192.168.100.0 netmask 255.255.255.0 gw 192.168.100.1

- 192.168.100.0: Destination network
- 255.255.255.0: Subnet mask
- 192.168.100.1: Gateway (Firewall VM)

On the Firewall VM (OPNsense):

- Routes configured via OPNsense Web GUI to **forward traffic to the IPS VM.**

On the IPS VM:

bash

CopyEdit

sudo route add default gw 192.168.100.1

- This ensures the IPS forwards packets back through the firewall after inspection.

4.2.2. Traffic Flow Explanation

Traffic Path:

1. **Attacker VM → Firewall VM:**
   - The attacker sends traffic to the target via the default gateway (Firewall).
   - The firewall filters and forwards legitimate traffic to the IPS.
2. **Firewall VM → IPS VM:**
   - The firewall inspects basic packet headers and forwards allowed traffic to the IPS.
3. **IPS VM (Suricata):**
   - The IPS performs deep packet inspection.
   - If malicious traffic is detected, it is logged and can be blocked.
   - Traffic is forwarded based on static routes.
4. **Return Path:**
   - Response traffic follows the same static route in reverse.

4.2.3. Advantages and Limitations

Advantages of Static Routing:

- **Full Control:** Traffic flow is fully controlled by the administrator.
- **Predictability:** No automatic route changes, which is ideal for a lab environment.

- **Simplicity:** Easy to configure and maintain for small networks.
- **Low Overhead:** No extra CPU or memory usage for dynamic routing protocols.

Limitations of Static Routing:

- **No Automatic Failover:** If a VM goes down, the routes will not automatically adjust.
- **Manual Updates Required:** Any network changes require manual reconfiguration.
- **Scalability Issues:** Not practical for large or complex networks with many paths.
- **No Load Balancing:** Cannot distribute traffic dynamically across multiple routes.

4.3. Practical Connection of VMs

In this project, the virtual machines (VMs) were configured to **simulate a real-world network environment** using VMware. Each VM played a specific role in the traffic flow and attack-defense cycle.

4.3.1. VM-to-VM Network Configuration

The VMs were connected to each other using **custom VMware virtual networks** to allow seamless communication without exposing the lab to external systems. We ensured the attacker, firewall, and IPS VMs were on the same virtual network segment for direct packet exchange.

Configuration Summary:

- All VMs are connected to the same **custom Host-Only network** to allow isolated communication.
- The **attacker VM sends traffic to the firewall VM,** which inspects it and forwards allowed traffic to the IPS VM.
- **Return traffic** flows back to the attacker through the same path.

4.3.2. NIC Settings (NAT, Host-Only, Bridged)

VMware Network Adapter Settings:

VM Role	NIC Mode	Purpose
Attacker VM	Host-Only	Controlled attack traffic, isolated from the internet.
Firewall VM	Host-Only	Middle node for filtering and forwarding traffic.
IPS VM	Host-Only	Deep packet inspection and logging.

NIC Modes Explanation:

- **Host-Only:**
  VMs can communicate with each other and the host machine but are completely isolated from external networks. This was used to ensure a **secure lab environment.**

- **Bridged (Optional):**
  Can be used if you want the VMs to be part of the same physical LAN. (Not used in our project for security reasons.)
- **NAT (Optional):**
  Can provide VMs with internet access via the host. (Used if updates were needed but disconnected during attack simulations.)

4.3.3. Connection Flow between Attacker, IPS, Firewall, and Victim

Traffic Flow:

1. **Attacker VM → Firewall VM (OPNsense):**
   All attack traffic is sent to the firewall, which applies initial filtering rules.
2. **Firewall VM → IPS VM (Suricata):**
   Cleaned traffic is forwarded to the IPS for deep packet inspection and logging.
3. **IPS VM:**
   Suricata inspects packets, logs alerts in eve.json, and either allows or blocks the traffic based on detection results.
4. **Return Path:**
   Response traffic follows the same path in reverse to maintain proper session states.

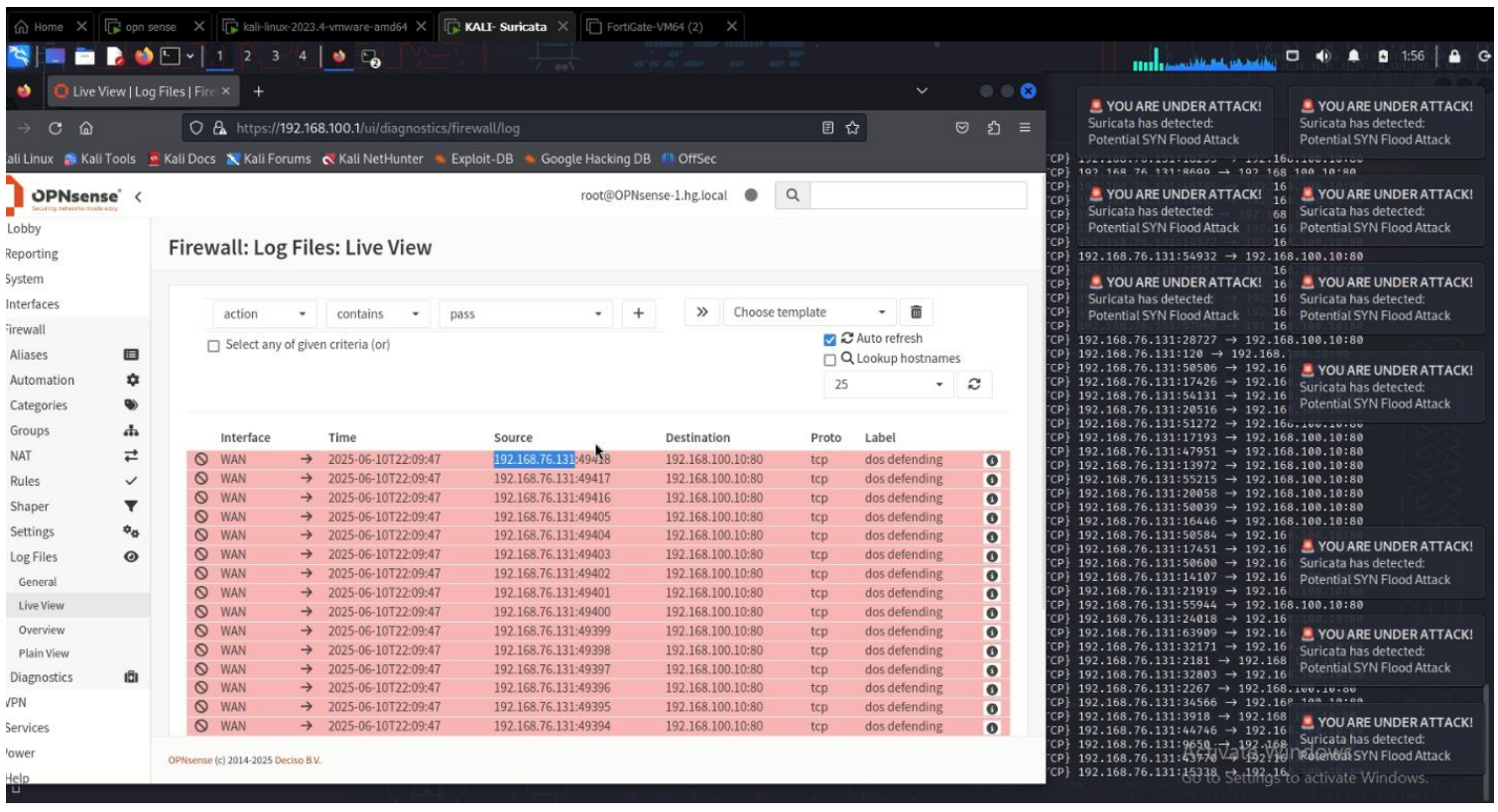
4.4. Project Execution and Simulation Steps

This section describes the step-by-step procedure followed to **deploy, attack, and monitor** the system in the virtual lab.

4.4.1. Launching the IPS and Firewall

Firewall Deployment (OPNsense):

- OPNsense was booted first and configured via the web interface.
- Firewall rules were applied to control which ports and IP addresses were allowed.
- Logging was enabled for all packets passing through.



IPS Deployment (Suricata):

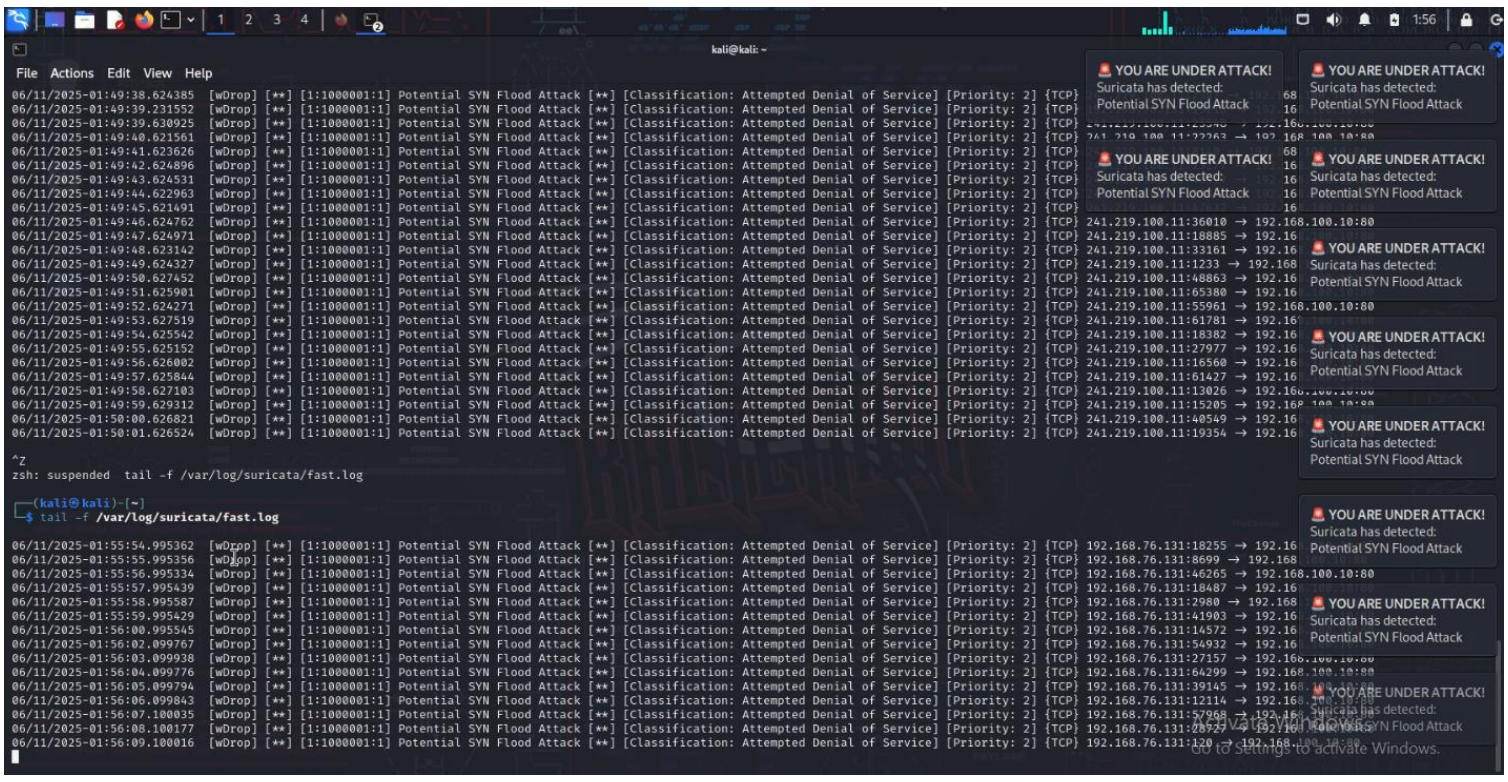
- Suricata was launched on the kali VM with the following command:

bash

CopyEdit

sudo suricata -c /etc/suricata/suricata.yaml -i eth0

- Suricata started monitoring the eth0 network interface in real-time.
- Logs were continuously written to /var/log/suricata/eve.json.



4.4.2. Performing DoS and DDoS Attacks

DoS Attack:

- The attacker VM used **Metasploit auxiliary modules** to launch single-source denial-of-service attacks.
- Example Metasploit Command:

bash

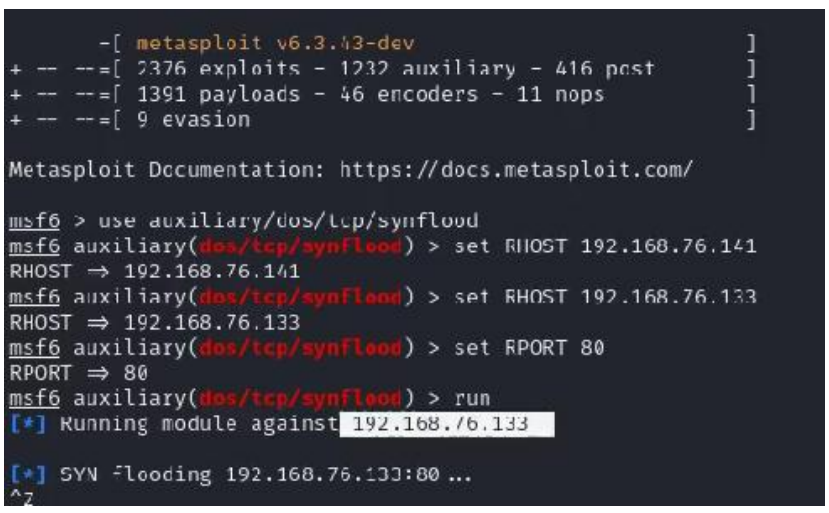
CopyEdit

use auxiliary/dos/http/slowloris

set RHOST <Firewall-IP>

set RPORT 80

run



DDoS Attack:

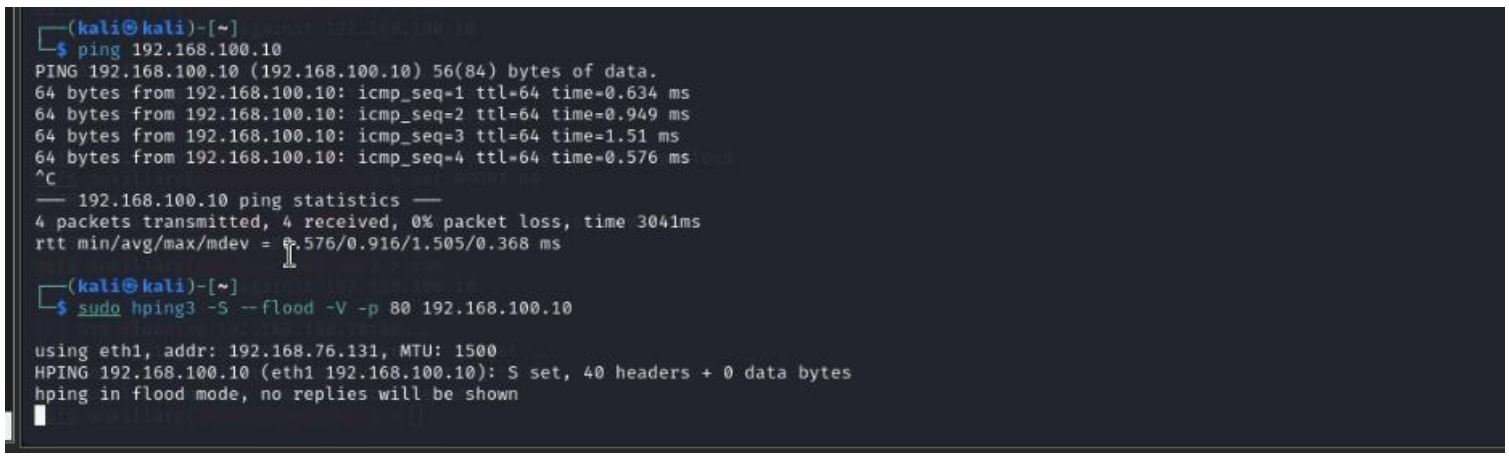
- The attacker VM used **Hping3** to perform TCP SYN floods.
- Example Hping3 Command:

bash

CopyEdit

hping3 -c 100000 -d 120 -S -w 64 -p 80 --flood --rand-source <Firewall-IP>

- Multiple terminals were launched simultaneously to **simulate multi-source DDoS.**



4.4.3. Monitoring and Logging in Real-Time

IPS Monitoring:

- Suricata detected abnormal traffic rates and logged alerts in eve.json.
- Custom Python scripts monitored the log file in real-time and generated:
  - **Desktop notifications**
  - **Live attack summaries**
  - **Automated HTML reports**

Firewall Logging:

- OPNsense displayed blocked packets and forwarded alerts from Suricata.

Real-Time Dashboard:

- Python-based visual dashboards were used to display:
  - Number of alerts per minute
  - Top attacking IP addresses
  - Most frequent attack signatures

4.5. Project Screenshots and Visuals

This section presents visual documentation of the practical implementation, configurations, attack simulations, and real-time monitoring results from our project. These visuals are essential to demonstrate the **real-world setup, practical execution, and system behavior during attacks.**

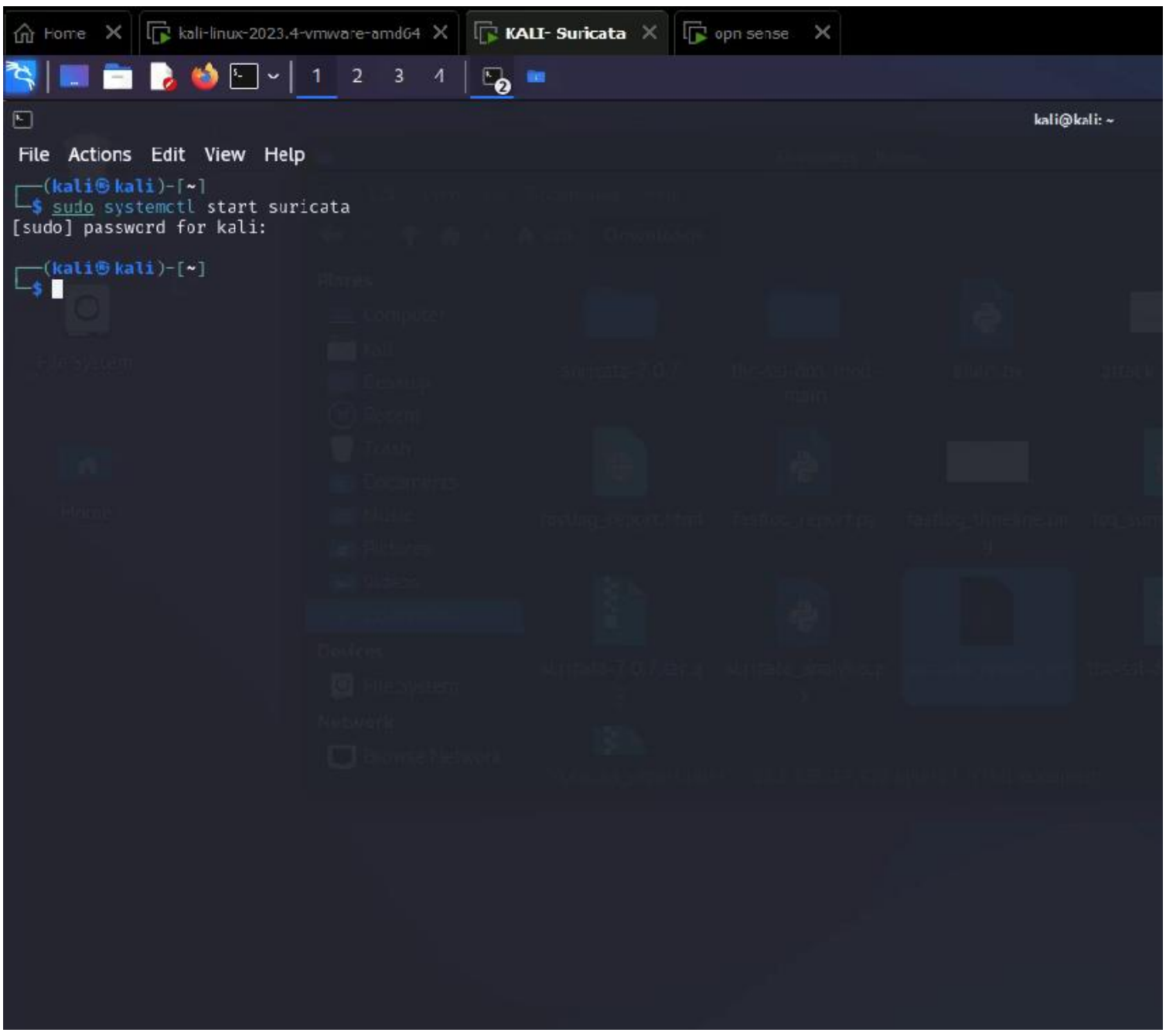
4.5.1. Full Network Topology

- **Attacker VM → Firewall VM → IPS VM → Host (Victim/Network)**
- IP addresses and routing paths are clearly indicated.
- Static routes direct traffic sequentially through each security layer.

4.5.2. Configuration Screenshots

Screenshots were taken from the following key configuration steps:

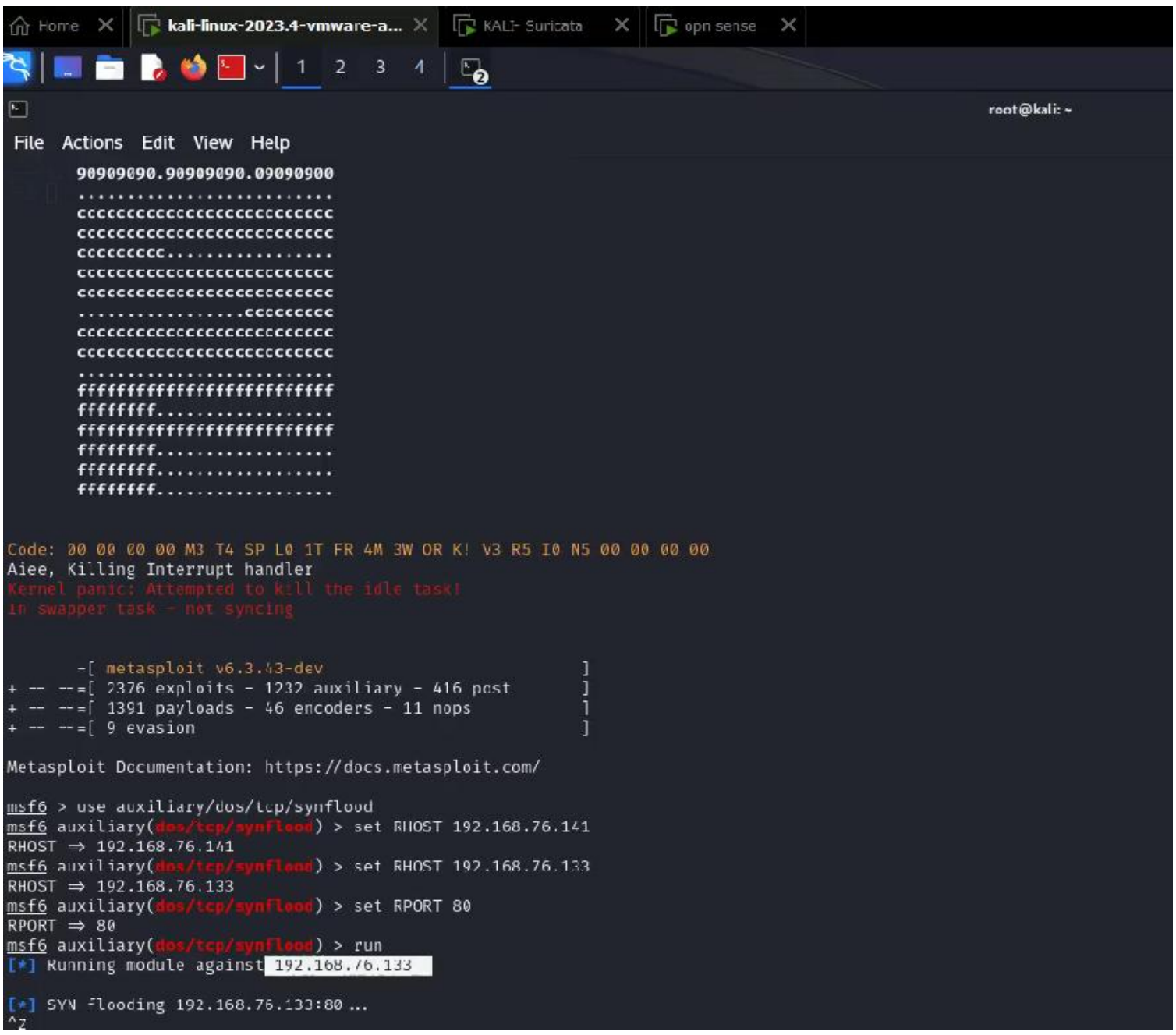
- OPNsense firewall interface settings and rule configurations.
- VMware NIC adapter settings for each VM.
- Suricata terminal showing live monitoring and startup logs.
- Static routing tables from all VMs.



4.5.3. Attack Simulation Screenshots

Visual evidence of simulated attacks includes:

- Metasploit running DoS attack modules.
- Hping3 DDoS flood commands in active sessions.
- Multiple parallel terminal windows simulating distributed attacks.



4.5.4. Real-Time Alert and Dashboard Visuals

Screenshots from real-time monitoring and alert systems:

- Python desktop notifications triggered by Suricata alerts.
- Logs captured in eve.json showing detected attack signatures.
- Automatically generated HTML reports summarizing attack types and sources.
- Real-time Python dashboards visualizing:
  - Number of attacks per minute.
  - Top attacking IP addresses.
  - Most frequent attack types.

Live View | Log Files | Fire × Suricata [wDrop] Report ×

file:///home/kali/Downloads/fastlog_report.html

Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec

Suricata [wDrop] Attack Report

Total Dropped Attacks: 329

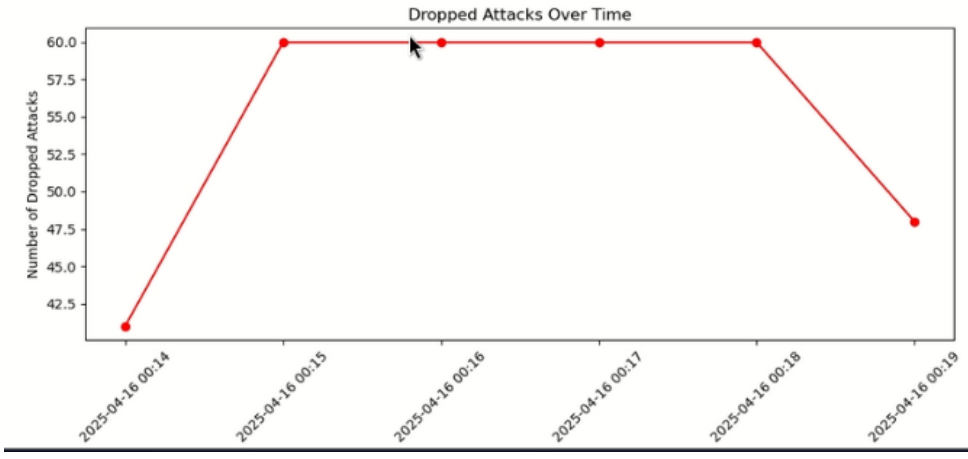
Top Attack Sources

- 228.115.182.43 - 329 attacks

Top Signatures

- [1:1000001:1] Potential SYN Flood Attack - 329 alerts

Attack Timeline



Suricata Attack Report ×

file:///home/kali/Downloads/suricata_report.html

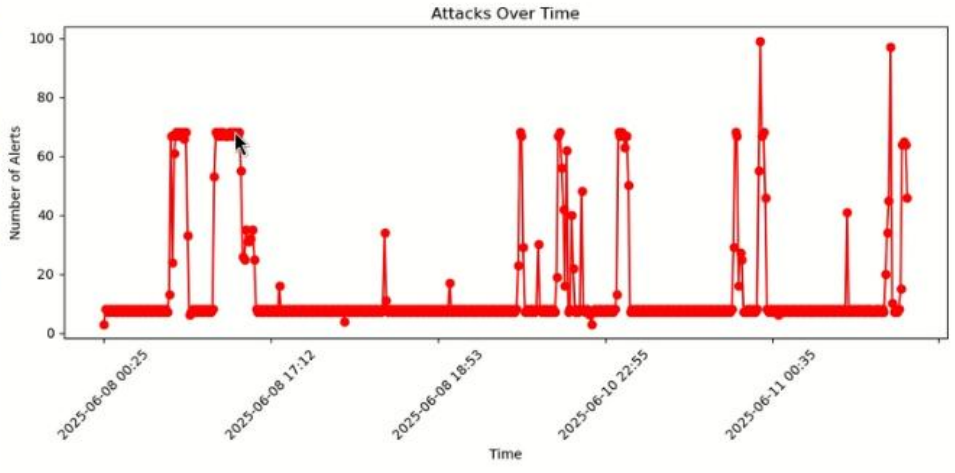
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec

- 192.168.76.131 - 327 alerts

Top Alert Types

- Potential SYN Flood Attack - 3658 alerts

Attack Timeline



Raw Alert Samples


```
[
    {
        "timestamp": "2025-06-11T01:59:40.243650-0400",
        "flow_id": 416024081595197,
```


Snort Log Summary ×

file:///home/kali/Downloads/log_summary.html

Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec

Intrusion Detection Alert

- **Timestamp:** 04/16/2025-00:20:11.753795
- **Action:** Attack has been dropped
- **Message:** Potential SYN Flood Attack [**]
- **Classification:** Attempted Denial of Service
- **Priority:** 2
- **Protocol:** TCP
- **Source:** 228.115.182.43:18243
- **Destination:** 192.168.76.133:80
- **Signature ID:** 1000001 (Rev 1)

4.6. Practical Challenges and System Performance

During project execution, we encountered several **practical challenges** that impacted system stability and simulation accuracy. These challenges provided valuable lessons about system limitations and optimization requirements.

4.6.1. Network Delay Issues

- High packet generation rates from Hping3 and Metasploit sometimes caused noticeable network delays.
- Delays occurred due to resource sharing between VMs on the same physical host.
- Static routing added additional hops that introduced minimal but measurable latency.
- Solution: Reduced attack packet rate during some simulations to maintain system responsiveness.

4.6.2. VM Resource Management

- Limited CPU and RAM availability on the host machine constrained VM performance.
- Suricata's multi-threaded packet inspection was resource-intensive, especially under high-load DDoS simulations.
- Running all VMs simultaneously sometimes caused VMware to lag or freeze.
- Solution: VM resources (RAM, CPU cores) were optimized and unnecessary background processes were closed to stabilize performance.

4.6.3. Packet Loss and Simulation Stability

- Under very high-volume traffic (during flood attacks), packet loss was observed within VMware's virtual switches.
- Some Suricata logs missed packets when the traffic exceeded processing capacity.
- OPNsense occasionally delayed log updates due to high queue sizes during attack peaks.
- Solution: Adjusted Hping3 flood rates to keep traffic within Suricata's processing limits while still simulating realistic attack loads.

4.7 Python Code Documentation

This section provides a detailed description of the custom Python scripts developed and used in this project. Each script plays a key role in real-time monitoring, alerting, and reporting within the Network Intrusion Prevention System (NIPS).

The Python scripts are used for three core purposes:

- Real-time log monitoring and alert notification.
- Log parsing and security report generation.
- Flash log file processing and summarization.

The Python source code files and their functions are explained below.

4.7.1 Real-Time Log Monitoring and Alert Notification Script

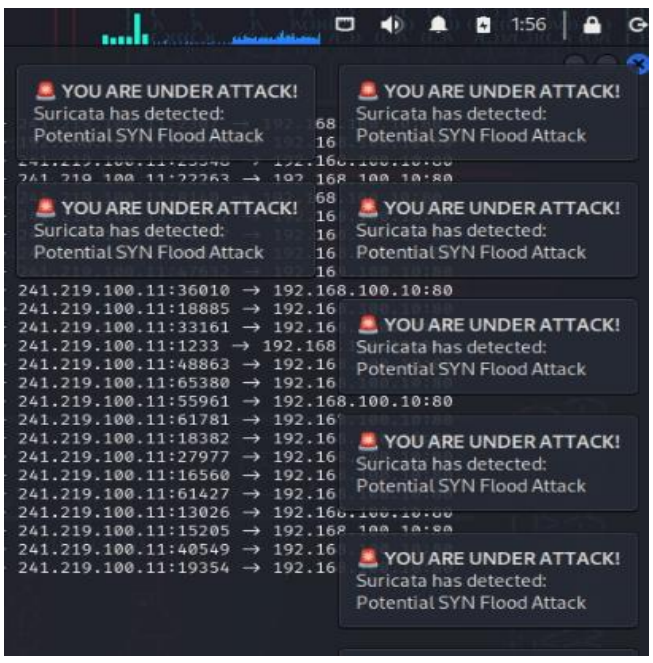
Purpose:

This Python script continuously monitors the eve.json log file generated by Suricata IPS. It reads log entries as they are written and instantly triggers desktop notifications and terminal alerts when a security event is detected.

Functionality:

- Monitors the log file in real-time.
- Detects security alerts based on event type.
- Sends system desktop notifications with the alert signature.
- Prints warning messages in the terminal.

Screenshot:



allert.py

```python
import json
import time
from plyer import notification  # Desktop notifications

# Suricata log path (default)
eve_log_path = "/var/log/suricata/eve.json"

def send_desktop_notification(alert_msg):
    notification.notify(
        title='🚨 YOU ARE UNDER ATTACK!',
        message=f'Suricata has detected:\n{alert_msg}',
        timeout=10  # The notification will stay for 10 seconds
    )

# Monitor the eve.json log
print("🚀 Monitoring Suricata logs...")

with open(eve_log_path, "r") as log_file:
    log_file.seek(0, 2)  # Move to end of file

    while True:
        line = log_file.readline()
        if not line:
            time.sleep(0.2)
            continue

        try:
            log = json.loads(line)
            if log.get("event_type") == "alert":
                alert_msg = log["alert"]["signature"]
                print("🚨 YOU ARE UNDER ATTACK!")
                print(f"⚠️  Alert: {alert_msg}")
                send_desktop_notification(alert_msg)
        except json.JSONDecodeError:
            continue
```

4.7.2 Log Parsing and Report Generation Script

Purpose:

This Python script reads large Suricata log files and automatically generates:

- Summarized security reports in HTML format.
- Graphs displaying attack frequencies and trends over time.
- Lists of top attacking IPs and most frequent attack types.

Functionality:

- Parses the eve.json log file.
- Groups attacks by type, source IP, and timestamp.
- Generates a visual HTML report with graphs and summaries.
- Helps visualize attack patterns for security analysis.

suricata_analysis.py

```python
import json
import pandas as pd
import matplotlib.pyplot as plt
from collections import deque
from datetime import datetime
from jinja2 import Template


data = deque(maxlen=10000)

with open("/var/log/suricata/eve.json", "r") as file:
    for line in file:
        if '"alert"' in line:
            try:
                data.append(json.loads(line))
            except json.JSONDecodeError:
                continue  # تجاهل السطر التالف

# ✅ Step 2: تحويل البيانات إلى DataFrame
alerts = pd.json_normalize(data)

alerts['timestamp'] = pd.to_datetime(alerts['timestamp'])
alerts['time_only'] = alerts['timestamp'].dt.strftime('%Y-%m-%d %H:%M')
alerts['src'] = alerts['src_ip']
alerts['dest'] = alerts['dest_ip']
alerts['signature'] = alerts['alert.signature']

# ✅ Step 3: رسم عدد التنبيهات على مدار الوقت
attacks_per_minute = alerts.groupby('time_only').size()
plt.figure(figsize=(10,5))
attacks_per_minute.plot(kind='line', marker='o', color='red')
plt.title('Attacks Over Time')
plt.xlabel('Time')
plt.ylabel('Number of Alerts')
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig('attack_chart.png')
plt.close()

# ✅ Step 4: توليد تقرير HTML
top_attackers = alerts['src'].value_counts().head(5)
top_signatures = alerts['signature'].value_counts().head(5)

html_template = """
<html>
<head><title>Suricata Attack Report</title></head>
<body style="font-family: Arial;">
    <h1>📊 Suricata Attack Report</h1>
    <p><b>Total Alerts:</b> {{ total_alerts }}</p>

    <h2>🔝 Top Attackers</h2>
    <ul>
    {% for ip, count in top_attackers.items() %}
        <li>{{ ip }} - {{ count }} alerts</li>
    {% endfor %}
    </ul>

    <h2>⚠️ Top Alert Types</h2>
    <ul>
    {% for sig, count in top_signatures.items() %}
        <li>{{ sig }} - {{ count }} alerts</li>
    {% endfor %}
    </ul>

    <h2>🕒 Attack Timeline</h2>
    <img src="attack_chart.png" width="800"/>

    <h2>📄 Raw Alert Samples</h2>
    <pre>{{ samples }}</pre>
</body>
</html>
"""

template = Template(html_template)
report = template.render(
    total_alerts=len(alerts),
    top_attackers=top_attackers.to_dict(),
    top_signatures=top_signatures.to_dict(),
    samples=json.dumps(list(data)[-3:], indent=4)
)

with open("suricata_report.html", "w") as f:
    f.write(report)

print("✅ Report generated: suricata_report.html")
print("✅ Attack chart saved: attack_chart.png")

```

4.7.3 Flash Log Parser Script

Purpose:

This Python script uses regular expressions to parse formatted log files or flash logs. It extracts key security event data and generates a structured, user-friendly HTML report.

Functionality:

- Reads and parses custom-formatted log files.
- Extracts:
  - Timestamp
  - Source and destination IPs
  - Ports
  - Protocols
  - Alert messages
- Outputs the result in an organized HTML file.

fastlog_report.py

```
import re

log = "04/16/2025-00:20:11.753795  [wDrop] [**] [1:1000001:1] Potential SYN Flood Attack [**] [Classification: Attempted Denial of Service] [Priority: 2] {TCP} 228.115.182.43:18243 -> 192.168.76.133:80"

# Regex pattern to parse the log
pattern = r"""
(?P<timestamp>\d+/\d+/\d+-\d+:\d+:\d+\.\d+)\s+
\[(?P<action>\w+)\].*?
\[(?P<gid>\d+):(?P<sid>\d+):(?P<rev>\d+)\]\s+
(?P<message>.*?)\s+
\[Classification:\s(?P<classification>.*?)\]\s+
\[Priority:\s(?P<priority>\d+)\]\s+
\{(?P<protocol>\w+)\}\s+
(?P<src_ip>[\d.]+):(?P<src_port>\d+)\s+->\s+
(?P<dst_ip>[\d.]+):(?P<dst_port>\d+)
"""

match = re.search(pattern, log, re.VERBOSE)
if match:
    data = match.groupdict()

    # Friendly action message
    action_message = "Attack has been dropped" if data['action'] == "wDrop" else data['action']

    # Create HTML summary
    html_summary = f"""
    <html>
    <head><title>Snort Log Summary</title></head>
    <body>
        <h2>🚨 Intrusion Detection Alert</h2>
        <ul>
            <li><strong>Timestamp:</strong> {data['timestamp']}</li>
            <li><strong>Action:</strong> {action_message}</li>
            <li><strong>Message:</strong> {data['message']}</li>
            <li><strong>Classification:</strong> {data['classification']}</li>
            <li><strong>Priority:</strong> {data['priority']}</li>
            <li><strong>Protocol:</strong> {data['protocol']}</li>
            <li><strong>Source:</strong> {data['src_ip']}:{data['src_port']}</li>
            <li><strong>Destination:</strong> {data['dst_ip']}:{data['dst_port']}</li>
            <li><strong>Signature ID:</strong> {data['sid']} (Rev {data['rev']})</li>
        </ul>
    </body>
    </html>
    """
```

Chapter Five

5.Conclusion and References

5.1. Conclusion

1. Summary of the Project

Restate the goal of your project: "To build and test a network intrusion prevention system using OPNsense firewall, Suricata IPS, and simulate network attacks (DoS and DDoS) using Metasploit."

Emphasize that the system was designed to detect and block **real-time threats in a virtual lab environment.**

Highlight the tools used (VirtualBox, OPNsense, Suricata, Kali Linux, Metasploit, etc.)

2. Key Accomplishments

Successfully integrated OPNsense firewall **with** Suricata IPS**, enabling real-time traffic inspection and blocking.**
Simulated realistic network threats **using Metasploit, giving hands-on experience with how modern attacks work.**
Achieved detection and prevention **of both DoS and DDoS attacks, as seen in the Suricata logs and OPNsense alerts.**
Built a virtual testing environment**, showing how security systems can be deployed and evaluated without affecting real networks.**

3. Challenges Faced

❖ **Expand on some of the problems you faced and how you solved them:**

Configuration issues**: Getting OPNsense and Suricata to run smoothly together, fine-tuning rules, and avoiding false positives.**
Network setup**: Bridging VMs correctly, configuring firewall rules, and routing traffic through the IPS.**
Performance**: Ensuring Suricata didn't slow down the network too much, especially under DDoS simulation.**
Learning curve**: Understanding Metasploit modules, configuring attacks, and interpreting log files.**

4. Skills and Knowledge Gained

Deepened understanding of network protocols and how traffic inspection works.
Hands-on experience with penetration testing tools (Metasploit) and defensive tools (IPS/Firewall).
Learned about signature-based vs anomaly-based detection, rule writing, and traffic filtering.
Gained practical skills in cybersecurity operations**,** network administration**, and** incident response**.**

5. Real-World Relevance

Emphasize how this project mirrors real-world security setups **used in companies, ISPs, and government systems.**
Talk about how IPS systems **like Suricata are used in** Security Operation Centers (SOCs)**.**
Mention the growing importance of network security **due to increasing cyber threats and data breaches.**

6. Future Enhancements

Add anomaly-based detection**: Use tools like Zeek/Bro for behavioral analysis.**
Real-time alerts**: Integrate email or SMS alerts using third-party plugins.**
Dashboard improvements**: Set up a centralized SIEM (Security Information and Event Management) like** Wazuh or ELK stack**.**
Test more attack types**: SQL injections, MITM (Man-In-The-Middle), phishing simulations.**
Use machine learning**: Explore ML-based intrusion detection for more adaptive defense.**
Deploy on real hardware**: Migrate the virtual testbed to a physical network lab for advanced testing.**

7. Final Thoughts

This project demonstrates how **open-source tools** can provide powerful cybersecurity defense.
The hands-on nature of the work made abstract security concepts real and understandable.
Acknowledging that **cybersecurity is a continuous process**, and tools must be kept up to date.
The project lays a strong foundation for further learning in **ethical hacking**, **penetration testing**, and **defensive security**.

5.2. References

- OPNsense Documentation: https://docs.opnsense.org/
- Suricata Documentation: https://docs.suricata.io/en/latest/
- Metasploit Framework: https://docs.metasploit.com/
- Kali Linux: https://www.kali.org/docs/
- VirtualBox: https://www.virtualbox.org/wiki/Documentation