الجمهورية الجزائرية الديمقراطية الشعبية
People's Democratic Republic of Algeria
وزارة التعليم العالي والبحث العلمي
Ministry of Higher Education and Scientific Research
المدرسة الوطنية العليا للتكنولوجيات المتقدمة
National Higher School of Advanced Technologies

Department of Electrical Engineering and Industrial Computing

Final Year Project to Obtain the Diploma of Engineering

– Field –

Telecommunication

– Specialty –

Telecommunications Systems and Networking

– Subject –

Hybrid AI-SIEM Framework: Leveraging CNN-LSTM Models and Wazuh For Attack Identification and Response

Realized by

AIT MIMOUNE Yasmine

REBAHI Khadidja

Presented publicly, the 22/06 /2025

Members of The Jury :

Name	Establishment	Grade	Quality
Mrs. BOUTERFAS Malika	ENSTA	MCA	President
Mr. BEGHAMI Sami	ENSTA	MAA	Examinator
Mr. CHERIFI Tarek	ENSTA	MCA	Examinator
Mrs. LAKHDARI Kheira	ENSTA	MCB	Supervisor
Mr. TIZIRINE Nasraddine	OTA (Djezzy)	CS Administrator	External Supervisor

2024 / 2025

Dedication

To my parents and closest ones — for always being there.
To the rare ones I met on the way— we may part here, but the story we shared, will always shape the way we carry each other forward.

Yasmine

For my family, and the friends I crossed path with, Hope I have made you proud and I hope this project could reflect what we have carried and learned through these years.

Khadidja

Acknowledgments

We would like to sincerely thank our academic supervisor, **Dr. LAKHDARI Khiera**, Assistant Professor in the GEII department at ENSTA, for his support, insightful feedback, and continuous guidance throughout this project.

Our deep gratitude also goes to **Mr. TIZIRINE Nassradine**, our industrial supervisor at Djezzy, whose professional mentorship, technical insight, and practical advices were instrumental in shaping the direction and quality of this work.

We extend our appreciation to the members of the jury **Mrs. BOUTERFAS Malika, Mr. Beghami Sami** and **Mr. CHERIFI Tarek** for their time, valuable comments, and thoughtful evaluation of our thesis.

We also acknowledge the teaching staff at the National Higher School of Advanced Technology, as well as the professional instructors involved in our training, whose expertise and dedication made this academic journey possible.

الملخّص

تُعد أنظمة إدارة معلومات وأحداث الأمان (SIEM) جزءًا حيويًا من الأمن السيبراني الحديث، إلا أن اعتمادها على قواعد كشف ثابتة يحدّ من فعاليتها في مواجهة التهديدات المتطورة. يعرض هذا البحث إطارًا هجينيًا يجمع بين الذكاء الاصطناعي ونظام Wazuh مفتوح المصدر لتعزيز الكشف عن الهجمات والاستجابة لها. تم دمج نموذج عصبي عميق يعتمد على بنية CNN-LSTM لاكتشاف الهجمات المعقدة مثل هجمات حجب الخدمة (DDoS) والأنماط الشاذة، مع تعزيز Wazuh بأدوات متقدمة مثل YARA لكشف البرمجيات الخبيثة، وتقنيات النماذج اللغوية الكبيرة (LLMS) ، وقواعد مخصصة، بالإضافة إلى دمج Suricata كنظام كشف التسلل الشبكي. وقد تم اختبار الإطار المقترح ضد عدة أنواع من الهجمات الحرجة مثل هجمات DDoS وهجمات القوة الغاشمة، وهجمات الويب، والبرمجيات الخبيثة، وهجمات التصيّد. والنتيجة هي نظام مرن وفعّال لا يكتفي بالكشف بل يقدّم استجابات ذكية ودقيقة مع تقليل التنبيهات الكاذبة. يثبت هذا البحث كيف يمكن للذكاء الهجين أن يحوّل أنظمة SIEM التقليدية إلى بيئات أمان أكثر ذكاءً واستجابة.

الكلمات المفتاحية: الأمن السيبراني، نظام SIEM ، Wazuh ، **الذكاء الاصطناعي، كشف التهديدات، هجمات** DDoS ، **البرمجيات الخبيثة، كشف التسلل**

Abstract

Security Information and Event Management (SIEM) systems are critical for modern cybersecurity, but their reliance on static rule-based detection limits their effectiveness against evolving threats. This thesis presents a Hybrid AI-SIEM Framework that combines the strengths of machine learning models and the open-source Wazuh SIEM to improve attack identification and response. We integrate a CNN-LSTM architecture for detecting complex attacks like DDoS and anomalies, and augment Wazuh with targeted solutions for various threats, including malware detection using YARA, LLMs technology, Wazuh rules, and Suricata-based network intrusion detection. The framework was tested against several critical attack types, including DDoS, brute-force login attempts, web application exploits, malware infections, and suspicious activities. The result is a layered, adaptable system that not only detects but actively responds to security incidents with improved accuracy and reduced false positives. This research demonstrates how hybrid intelligence can transform static SIEMs into smarter, more responsive security ecosystems.

Keywords: Cybersecurity, SIEM, Wazuh, AI, Threat Detection, DDoS, Malware, Intrusion Detection

Résumé

Les systèmes de gestion des informations et des événements de sécurité (SIEM) sont essentiels pour la cybersécurité moderne, mais leur dépendance aux règles statiques limite leur efficacité face à des menaces en constante évolution. Ce mémoire présente un cadre hybride IA-SIEM qui combine les atouts des modèles d'apprentissage automatique avec la solution open-source Wazuh afin d'améliorer l'identification et la réponse aux attaques. Nous intégrons une architecture CNN-LSTM pour détecter des attaques complexes comme les attaques DDoS et les anomalies, et nous renforçons Wazuh avec des outils spécifiques tels que YARA pour la détection des malwares, la technologie des grands modèles de langage (LLMs), des règles personnalisées, ainsi que Suricata comme système de détection d'intrusions réseau. Le cadre proposé a été testé sur plusieurs types d'attaques critiques, notamment les attaques DDoS, les tentatives de force brute, les attaques Web, les infections par malware et les campagnes de phishing. Le résultat est un système modulaire et réactif, capable non seulement de détecter les incidents de sécurité mais aussi d'y répondre avec une meilleure précision et moins de faux positifs. Ce travail démontre comment l'intelligence hybride peut transformer les SIEM classiques en écosystèmes de sécurité plus intelligents et adaptatifs.

Mots-clés: Cybersécurité, SIEM, Wazuh, Intelligence Artificielle, Attaques, DDoS, Malware, Détection d'intrusions

Contents

Introduction **6**

1 Cybersecurity **7**

1.1 Introduction . . . 7
1.2 Cybersecurity Threats and Attacks . . . 7
1.2.1 Denial of Service (DoS) Attacks . . . 7
1.2.2 Distributed Denial of Service (DDoS) Attacks . . . 7
1.2.3 Brute Force Attacks . . . 8
1.2.4 Man in the middle (MitM) Attacks . . . 8
1.2.5 Malware . . . 8
1.2.6 Web Attacks . . . 8
1.3 Intrusion detection systems . . . 8
1.3.1 Definition . . . 8
1.3.2 Types of IDS . . . 9
1.3.3 IDS Architecture . . . 10
1.3.4 IDS Detection Techniques . . . 10
1.4 Security Information and Event Management (SIEM) . . . 10
1.4.1 Definition . . . 10
1.4.2 SIEM Functional Processes and Workflow . . . 11
1.4.3 SIEM Solution Platforms . . . 12
1.4.3.1 Licensed Solutions . . . 12
1.4.3.2 Open Source Solutions . . . 12
1.4.3.3 Comparative Analysis of SIEM Solutions . . . 12
1.5 Wazuh SIEM . . . 13
1.5.1 Wazuh Architecture and Components . . . 13
1.5.1.1 Wazuh Manager (Server) . . . 13
1.5.1.2 Wazuh Indexer . . . 14
1.5.1.3 Wazuh Dashboard . . . 14
1.5.1.4 Wazuh Agents . . . 15
1.5.2 Communication between Wazuh Components . . . 16
1.6 The Case Of AI Integration . . . 16
1.7 Conclusion . . . 16

2 Deep Learning-based Intrusion Detection **17**
2.1 Introduction . . . 17
2.2 Convolutional Neural Networks (CNNs) . . . 17
2.2.1 Convolution Layer . . . 18
2.2.2 Pooling Layer . . . 18
2.2.3 Activation Function . . . 18
2.2.4 Fully Connected Layer . . . 19
2.3 Recurrent Neural Networks (RNNs) . . . 19
2.3.1 RNNs Structure . . . 19
2.3.2 Long Short Term Memory (LSTM) . . . 20
2.3.3 LSTM Structure . . . 20
2.4 Hybrid CNN-LSTM Model . . . 22
2.5 Conclusion . . . 22

3 Methodology **23**
3.1 Introduction . . . 23
3.2 Project Overview : The Hybrid AI-SIEM Framework . . . 23
3.2.1 Data Collection Layer : . . . 24
3.2.2 Integration Layer : . . . 25
3.2.3 Correlation Layer : . . . 25
3.2.4 Decision and Response Layer : . . . 25
3.3 Experimental Environment and Design Rationale . . . 26
3.3.1 Docker and Docker Compose . . . 26
3.3.2 Network-infrastructure Simulation . . . 26
3.4 AI-Based Detection Module (CNN-LSTM) . . . 28
3.4.1 Dataset . . . 28
3.4.1.1 Dataset Cleaning . . . 29
3.4.1.2 Handling Class Imbalance . . . 29
3.4.1.3 Feature Selection . . . 30
3.4.1.4 Data Scaling and Reshaping . . . 31
3.4.2 Overview of Model Architecture . . . 31
3.5 Training and Evaluation of CNN-LSTM Model . . . 33
3.6 Conclusion . . . 35

4 Implementation and Results **36**
4.1 Introduction . . . 36
4.2 Wazuh Implementation . . . 36
4.2.1 Docker Compose Configuration . . . 38
4.2.2 Wazuh Agent Deployment . . . 38
4.3 Integration Layer Implementation . . . 39
4.3.1 Integration of the AI Model into the Wazuh SIEM Framework . . . 39
4.3.1.1 Deployment of the AI Model as an API Service . . . 40

4.3.1.2 Real-Time Traffic Process and Feature Collection . . . 41
4.3.1.3 Method-1 : Direct Ingestion into Wazuh Indexer via HTTP . . . 42
4.3.1.4 Method-2 : File-Based Integration . . . 42
4.3.2 Test Environment and Attack Simulation for Both Methods . . . 43
4.3.2.1 Attacks Simulation . . . 43
4.3.2.2 Results of the Wazuh-IA Integration . . . 45
4.3.2.3 Conclusion . . . 47
4.3.3 Integrations with External Detection Tools . . . 47
4.3.3.1 Network IDS Integration (Suricata) . . . 47
4.3.3.2 Detecting Malware and Suspicious Binaries with YARA Integration . . . 49
4.3.3.3 Leveraging LLMs for Alert Enrichment (YARA + ChatGPT API) . . . 53
4.3.4 Core Security Monitoring Use Cases . . . 56
4.3.4.1 Blocking Malicious Actors and Brute-force Attacks . . . 56
4.3.4.2 Monitoring Docker Events . . . 59
4.3.4.3 Detecting Web Application Attacks (SQL Injection Shellshock) . . . 61
4.3.4.4 Monitoring Execution of Malicious Commands . . . 64
4.3.4.5 Disabling a Linux User Account with Active Response . . . 66
4.3.4.6 Detecting Suspicious Binaries . . . 68
4.4 Conclusion . . . 69

Conclusion **70**

Future Work and Perspectives **71**

A Software Deployments **72**
A.1 Wazuh Deployment Steps . . . 72
A.2 Wazuh Agent Installation . . . 73
A.3 Java CICFlowMeter Setup . . . 73

List of Figures

1.1 Classification of IDS [29]. . . . 9
1.2 Types of IDS [31] 9
1.3 SIEM Functional Process 11
1.4 Wazuh Server Components [30]. . . . 14
1.5 Wazuh Agent Components[30]. . . . 15

2.1 Simplified CNN Layers 18
2.2 The layout of a basic RNN 20
2.3 LSTM Cell Structure 21
2.4 An example of a Hybrid CNN-LSTM Model [26] 22

3.1 Hybrid AI-SIEM Framework Architecture. . . . 24
3.2 Hybrid AI-SIEM Framework High Level Layers Architecture 25
3.3 Docker-compose architecture. . . . 26
3.4 Simulated Network infrastructure. . . . 27
3.5 Label Distribution Before and After Cleaning the Dataset. . . . 29
3.6 Class Distribution Before SMOTE. . . . 30
3.7 Class Distribution After SMOTE. . . . 30
3.8 Final Selected Features. . . . 31
3.9 Simplified CNN-LSTM model Architecture. . . . 32
3.10 Training Curves. . . . 34
3.11 Confusion Matrix. . . . 35

4.1 Wazuh Workflow 36
4.2 Wazuh Cluster Containers Stutus 37
4.3 Wazuh Dashboard Access 37
4.4 Wazuh Dashboard 37
4.5 Docker Compose Configuration. . . . 38
4.6 Wazuh Agent deployment. . . . 39
4.7 AI-Integration Approaches with Wazuh. . . . 40
4.8 Deployment of the AI Model as an API Service. . . . 41
4.9 Data Real-Time Processing Flow. . . . 42
4.10 Wazuh Configuration to Monitor Json File Alerts. . . . 43
4.11 Custom Rule to Detect DDoS Attacks. . . . 43

4.12 Live Capture of CICFlowMeter during DDOS Attacks Simulation 44
4.13 Data Processing 45
4.14 Low DDoS Attack Detected. 45
4.15 High DDoS Attack Detected. 45
4.16 DDoS Attacks Alerts by The CNN-LSTM Model on The Wazuh Dashboard (File Based Integration Method). 46
4.17 Alert Successfully generated to The Wazuh Indexer. 46
4.18 DDoS Attacks Alerts by The CNN-LSTM Model on The Wazuh Dashboard (Direct Ingestion Method) 47
4.19 Suricata Alert Workflow. 48
4.20 Configuration of Suricata.yaml File. 48
4.21 Configuration of Wazuh Agent to Ingest Suricata Logs 49
4.22 Suricata Alerts Through the Threat Hunting module. 49
4.23 Diagram of YARA Integration Stages within Wazuh. 50
4.24 Loading Malware Detection Rules from Nextron Systems Valhalla Repository 50
4.25 Snippet From The yara.sh File. 51
4.26 Custom Wazuh Rules for File Changes and YARA Scan Results. 51
4.27 Custom Wazuh Decoders for YARA Scan Results Extracting. 52
4.28 Active Response Configuration for Triggering the yara.sh Script. 52
4.29 YARA Alerts Through the Threat Hunting Module. 53
4.30 YARA Alerts Through the Wazuh Discover. 53
4.31 Yara.sh Script Modifications. 54
4.32 Adjustments of The Custom Decoders. 55
4.33 Open-AI API Query. 55
4.34 Real Time Active Response : Deleting Malware Files. 56
4.35 Configuration of Wazuh Agent to Monitor Web Server Logs. 57
4.36 Downloading The AlienVault IP Reputation Database. 57
4.37 Wazuh Rule Configuration for Blacklisted IPs Alerts. 57
4.38 Active Response Configuration to Block Flagged IPs. 57
4.39 Alert triggered from blacklisted IP (Rule ID 100100) 58
4.40 Active Response Configuration to Block Detected IPs. 58
4.41 SSH Brute-Force Attack Simulation on Ubuntu Using Hydra. 59
4.42 Attacker Blocked for 60 seconds. 59
4.43 Alert for SSH brute-force detection And Active response 59
4.44 Enabling Docker listener. 60
4.45 Agent Configuration Via Syslog. 60
4.46 Docker Listener Capturing Real-Time Docker Events. 61
4.47 Alerts Triggered from Monitored Docker Events. 61
4.48 Wazuh Agent Configuration of to Monitor Apache access Logs. 62
4.49 SQL injection attempt Simulation. 62
4.50 Alert triggered by SQL injection attempt Through the Threat Hunting Module. 63

4.51 Alert triggered by SQL injection attempt (Rule ID 31103). 63
4.52 Alert Triggered by Shellshock Attack Attempt. 64
4.53 Alert Triggered by Shellshock Attack Attempt on Threat Hunting Module. 64
4.54 Enabling Process Monitoring. 65
4.55 Custom Rule to Detect Suspicious Command Execution. 65
4.56 Lookup File For Suspicious Programs. 65
4.57 Configuration of CDB Lists on the wazuh server's Rulest Section. 66
4.58 Alert Triggered by Execution of Suspicious Command. 66
4.59 Rule Appears if There Three Failed Logins. 67
4.60 Active Response Configuration to Disable Account. 67
4.61 Account Locked Output. 68
4.62 Alert Triggered by Rule-ID 120100 with Active Response Disabling User Account. 68
4.63 Rootcheck Block 68
4.64 Malware Test Simulation 69
4.65 Rootcheck Alert 69

List of Tables

1.1 Comparison between SIEM solutions: Deployment and Supported Operating Systems [13] 13
1.2 Wazuh Indices for Storing Events. 14

3.1 Dataset Characteristics Summary 28
3.2 Architecture Layers of the CNN–LSTM Model 32
3.3 Training Configuration 33
3.4 Classification Performance Metrics 34

List of Acronyms

- **AI** — Artificial Intelligence
- **SIEM** — Security Information and Event Management
- **ML** — Machine Learning
- **DL** — Deep Learning
- **RNN** — Recurrent Neural Network
- **CNN** — Convolutional Neural Network
- **DDoS** — Distributed Denial of Service
- **DoS** — Denial of Service
- **IDS** — Intrusion Detection System
- **NIDS** — Network Intrusion Detection System
- **JSON** — JavaScript Object Notation
- **LLM** — Large Language Model
- **LSTM** — Long Short-Term Memory
- **MitM** — Man in the Middle
- **SMOTE** — Synthetic Minority Oversampling Technique
- **SQL** — Structured Query Language
- **API** — Application Programming Interface
- **SSH** — Secure Shell
- **CSV** — Comma-Separated Values

Introduction

Organizations these days are up against a rapidly shifting environment of cyber threats from brute-force login attacks to advanced distributed denial-of-service (DDoS) attacks and covert malware. Conventional SIEM tools like Wazuh are good at providing robust capabilities in log analysis, rule-based alerting, and incident tracking—yet are inadequate in addressing new or high-volume attacks that necessitate more dynamic detection methods. Static rules are not able to keep pace with attackers and therefore create blind spots as well as a deluge of false positives.

In the meantime, deep learning algorithms like Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) networks have also demonstrated high potential to identify patterns and abnormalities in massive datasets. Still, these algorithms run independently, isolated from the operational stream of actual SIEM systems. These outputs are infrequently utilized to generate real-time responses or to input to larger detection processes, reducing their real-world usefulness.

The thesis proposes a Hybrid AI-SIEM Framework that incorporates the real-time operational power of Wazuh plus the adaptive intelligence of deep learning. We integrate CNN-LSTM models natively within the Wazuh pipeline, and supplement the system with utilities like Suricata for intrusion detection at the network level and YARA for malware and file signature scanning. The resulting framework is capable of detecting, correlating, and responding to attacks of a wide variety across various vectors.

To confirm this strategy, the framework is validated in simulation against live traffic and real attack situations. By blending AI-driven inspection with traditional rule-based monitoring, detection accuracy is greatly enhanced, while reducing the occurrence of alert fatigue, and enabling quicker, smarter response to incidents. The modularity of the framework also provides the flexibility to extend the framework in the future with support for novel types of attacks or different AI models, thus enabling the framework to evolve according to developing security requirements.

Chapter 1

Cybersecurity

1.1 Introduction

Cybersecurity refers to the strategies, technologies, and processes used to protect digital systems, networks, and data from unauthorized access, misuse, or disruption, it's 'the ability to protect or defend the use of the medium of cyberspace from cyber attacks, according to the National Institute of Standards and Technology (NIST, 2017). This chapter summarizes the most prevalent forms of cyberattacks, along with the systems that identify them and manage them. Special focus is given to intrusion detection systems and security information and event management solutions, particularly to the open-source framework at the center of our proposed architecture.

1.2 Cybersecurity Threats and Attacks

A cyber attack refers to an action designed to target a computer or any element of a computerized information system to change, destroy, or steal data, as well as exploit or harm a network. Cyber attacks have been on the rise, in sync with the digitization of business that has become more and more popular in recent years. Technological evolution has also brought new ways that are continuously developed to perform attacks, reach even harder to penetrate targets, and remain untracked. However, traditional cyber threats remain the source of the most common attacks [1]. Below, the most common types of these attacks, according to international literature, are introduced.

1.2.1 Denial of Service (DoS) Attacks

DOS attacks take place when a single attacker sends a huge amount of traffic to a target such as websites, controllers to overwhelm the resources of a system to the point where the victim is unable to reply to legitimate users requests.

1.2.2 Distributed Denial of Service (DDoS) Attacks

DDOS are initiated by a vast array of malware-infected host machines controlled by the attacker, in which they take control of multiple computers often referred to as a botnet, and use them simultaneously to send

an even larger volume of traffic to the victim machine. This flood of traffic drains the resources of a system , making its services inaccessible to legitimate users. Common types of DDOS include : **SYN Floods , UDP Floods,Goldeneye, Slowsoris, Slowhttptest**

1.2.3 Brute Force Attacks

This type of attacks consist of repeated attempts to gain access to protected information (e.g. passwords, encryption, etc.) until the correct key is found.

1.2.4 Man in the middle (MitM) Attacks

Man in the middle attack occurs when the attacker interferes between the two communication ends, in a way that every message sent from point A to point B reaches the attacker before reaching its destination [1].

1.2.5 Malware

Malware is a generic term describing types of malicious software, Malware infects a computer and changes how it functions, destroys data, or spies on the user or network traffic as it passes through, used by the attacker to compromise the confidentiality, availability and integrity of data. Most common types of malware are: **Trojan , Spyware , Ransomware, Fileless malware , Worms , Rootkits, Bot/Botnet**

1.2.6 Web Attacks

Web attacks Refer to threats that target vulnerabilities in web-based applications. Every time information is entered into a web application, it triggers a command that generates a response. Attackers work within the frameworks of these kinds of requests to their advantage. Some common web attacks include : **SQL injection , Cross-Site Scripting (XSS) , Phishing Attacks**

1.3 Intrusion detection systems

1.3.1 Definition

An IDS is a security tool, either hardware or software, that monitors network traffic and raises alarms when it detects malicious activity, it catches early signs of attacks like Denial of Service (DoS) or Man in the Middle (MitM) [3], enabling quick responses, an IDS helps security teams understand incidents and improve defenses by logging traffic and providing real-time details. IDS can be classified with the perspective of its deployment or detection methods. A classification taxonomy is given in Figure 1.1 and details are provided in the following section.

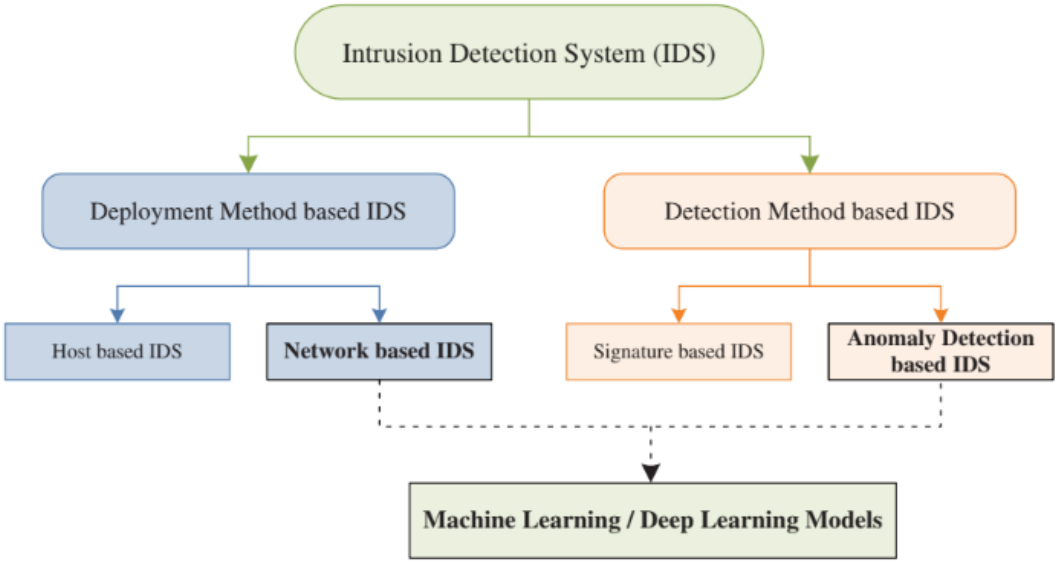


Figure 1.1: Classification of IDS [29].

1.3.2 Types of IDS

There are three main types of IDS Network IDS (NIDS) and Host IDS (HIDS) and hybrid systems. The figure 1.2 demonstrates how the three types of IDS can be implemented to work together on a network infrastructure. **NIDS** are designed to continuously monitor network traffic to detect various threats, including DDoS attacks, unauthorized attempts, and port scanning activities [2]. In contrast, **HIDS** operates directly on individual devices or hosts [3]. They monitor local system logs, file integrity, and system calls to detect suspicious activities. Hybrid systems integrate the strengths of both HIDSs and NIDSs by combining detailed insights from individual hosts with a broad perspective on overall network activity ([4],[5])

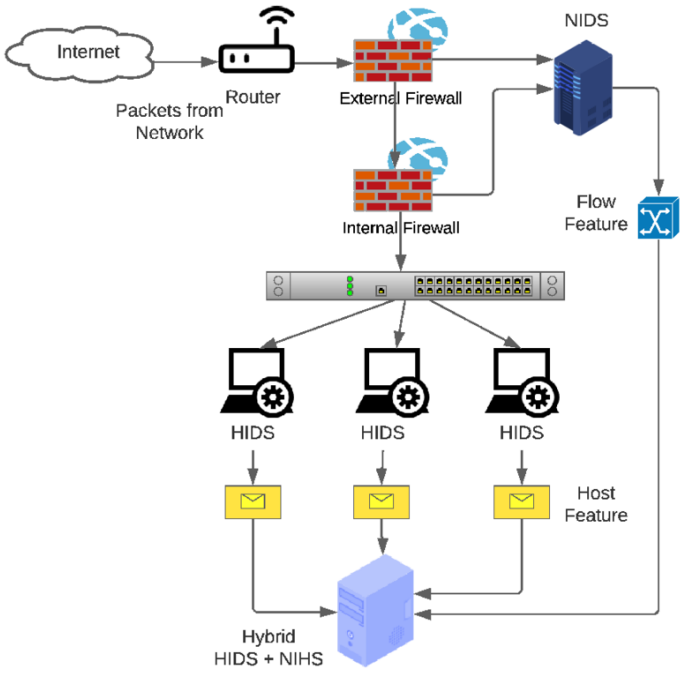


Figure 1.2: Types of IDS [31]

1.3.3 IDS Architecture

IDS architecture can follow centralized, distributed, or hybrid models, each offering different levels of visibility and control [2].

- **Centrelized IDS Architecture** Centralized IDS systems consolidate all monitoring data at a single location for unified analysis and management. But it may encounter scalability issues and become a single point of failure in larger networks [2].
- **Distributed IDS Architecture** Distributed IDS architectures deploy independent monitoring nodes across different network segments; it enables localized threat detection and improves scalability.
- **Hybrid IDS Architecture** Hybrid IDS models combine the strengths of centralized and distributed systems. They distribute detection tasks among various nodes while aggregating data centrally for analysis, offering a balanced solution that enhances scalability and robust threat detection [2].

1.3.4 IDS Detection Techniques

For intrusion detection system , there is three mais techniques Misuse-based, Anomaly-based and the combination of the two approaches as a hybrid technique.

1. **Misuse-Based Detection**: It comprises two subcategories [1], the first one being Signature-Based Detection, relies on a predefined database of attack signatures. The second approach is ML-Based Misuse Detection, ML models learn from historical attack data, understanding the general structure of known attacks and potentially predicting evolving variants.
2. **Anomaly-Based Detection** Developed to address the shortcomings of misuse-based methods, anomaly-based detection builds a model of normal behavior by analyzing network traffic features. Deviations from this model are flagged as anomalies,[3]. This method can be implemented [1] using machine learning, statistical techniques, or finite-state machines.
3. **Hybrid Techniques** Hybrid systems combine misuse-based and anomaly-based methods by using signature matching to detect known threats and anomaly detection to identify novel attacks [6].

1.4 Security Information and Event Management (SIEM)

1.4.1 Definition

Security Information and Event Management or SIEM is a combination of Security Information Management (SIM) and Security Event Management (SEM) to provide a centralized real-time monitoring of cybersecurity incidents. collecting,aggregating, and analyzing event logs from network devices and applications, applying correlation rules to detect suspicious activities and respond to potential threats. SIEM solutions can be deployed as software, appliances, or managed services, playing a crucial role in both threat detection and reporting [7].

1.4.2 SIEM Functional Processes and Workflow

SIEM systems follow a defined process to analyze and respond to security events in real time as shown in figure 1.3.

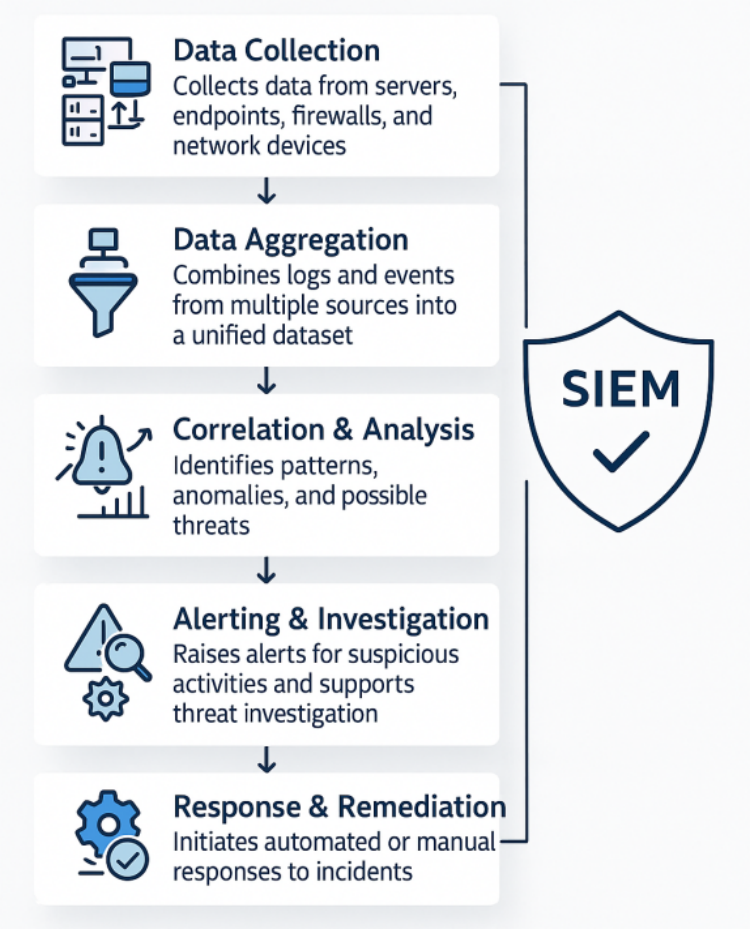


Figure 1.3: SIEM Functional Process

1. **Data Collection.** Gather logs and events from hosts, applications and security devices into a central SIEM platform for unified visibility [8].

2. **Normalization & Aggregation.** Convert diverse log formats into a standard schema and group related events (e.g. logins, malware alerts) via parsers and rules to prepare for analysis [8].

3. **Correlation & Analysis.** Apply rule-based linking of multiple events to uncover multi-step attacks; current Boolean-logic rules are effective for known paths but struggle with novel or zero-day threats [9].

4. **Response & Reporting.** Generate alerts and, optionally, trigger automated actions (e.g. firewall tweaks, scripts, tickets). Most SIEMs require add-ons to enable fully customizable incident response workflows [9].

5. **Storage & Retention.** Archive processed logs—often with manual or script-driven transfers to HDFS, cloud buckets, etc.—but limited retention (6 months) and lack of automated reuse hinder detection of long-running threats [9].

1.4.3 SIEM Solution Platforms

SIEM solutions are generally divided into two categories: licensed and open-source solutions, below is an overview about some widely used SIEM platforms.

1.4.3.1 Licensed Solutions

Below are some of the most widely used licensed SIEM solutions in the market:

- **Splunk**: a paid platform made for big data analysis in sectors like banking, healthcare, and security [10]. It works as the basis of its SIEM solution, allowing for the collecting, storing, and analyzing of security-related data. [11].
- **IBM QRadar** a modular and flexible SIEM solution designed for medium to large businesses. It provides log management, event linking, threat detection, and response to incidents. It can be set up as a single system or in a distributed manner. Supported by X-Force development [12].
- **LogRhythm NextGen** LogRhythm's NextGen SIEM is a security solution that combines different parts like DetectX, AnalytiX, and RespondX, along with monitoring tools for networks and systems like NetworkXDR and SysMon. [12].

1.4.3.2 Open Source Solutions

Examples of popular open-source SIEM solutions include :

- **Wazuh** Wazuh is an open-source security monitoring tool that functions as a host-based intrusion detection system. It provides capabilities such as threat detection, incident monitoring, response, and compliance. Its architecture consists of agents, a server, and the Elastic Stack, which work together to collect, analyze, and correlate security events [12].
- **ELK Stack** Composed of Elasticsearch, Logstash, and Kibana, a powerful open-source solution for log management and analysis. Elasticsearch is a scalable time-series data analytics engine that stores and indexes vast amounts of log data, allowing for efficient searches and real-time analysis [12].
- **Mozdef** Developed by the Mozilla Foundation, Mozdef is an open-source SIEM solution designed to automate security incident handling. It integrates with Elasticsearch and log management modules to enable event aggregation and correlation [12].

1.4.3.3 Comparative Analysis of SIEM Solutions

Important details to consider for system compatibility and ease of integration are shown in Table 1.1, which provides a quick comparison of how each SIEM solution is deployed and which operating systems it supports.

Table 1.1: Comparison between SIEM solutions: Deployment and Supported Operating Systems [13]

SIEM Solution	Deployment	Operating Systems
Wazuh	Deployed on Amazon EC2, or locally in virtual and physical environments.	Amazon Linux 2, CentOS 7, Debian 8, Oracle Linux 6, RHEL 6
ELK Stack	Deployable on virtual environments, physical hardware, private or public cloud.	Windows, RHEL 7, CentOS 7, Oracle Linux 7, Ubuntu 14, SLES/openSUSE 15
Mozdef	SaaS deployment using Docker containers, in the cloud or locally.	CentOS 6, RHEL 6, Ubuntu 14
Splunk	Cloud-based, SaaS, or on-premises deployment.	Windows, Linux, Mac, Solaris
IBM QRadar	Cloud-based, SaaS, or on-premises deployment.	Red Hat, Linux
LogRhythm	Deployable on cloud, SaaS, web, or Windows environments.	Windows, Appliance, Cloud

1.5 Wazuh SIEM

1.5.1 Wazuh Architecture and Components

Wazuh follows a modular architecture that integrates multiple components to enhance security monitoring. It includes agents deployed on endpoints and cloud instances for data collection, a central manager responsible for event correlation and threat detection, and an API that facilitates integration with external security tools. Additionally, the Elastic Stack is utilized for data storage, processing, and visualization, while a customizable ruleset defines security policies [32].

1.5.1.1 Wazuh Manager (Server)

The Wazuh server analyzes agent data using decoders and rules, utilizing threat intelligence to identify indicators of compromise (IOCs). It can analyze hundreds or thousands of agents and scale horizontally when set up as a cluster, managing agents remotely.

- **Wazuh Server Architecture and Components**
  The Wazuh server comprises several components that have different functions, such as enrolling new agents, validating each agent identity, and encrypting the communications between the Wazuh agent and the Wazuh server. The figure 1.4 shows how all Wazuh server components are brought together to form a cohesive SIEM platform, the components includes: Agent Enrollment Service, Agent Connection Service, Analysis Engine, Wazuh RESTful API, Wazuh Cluster Daemon ,Filebeat [30].

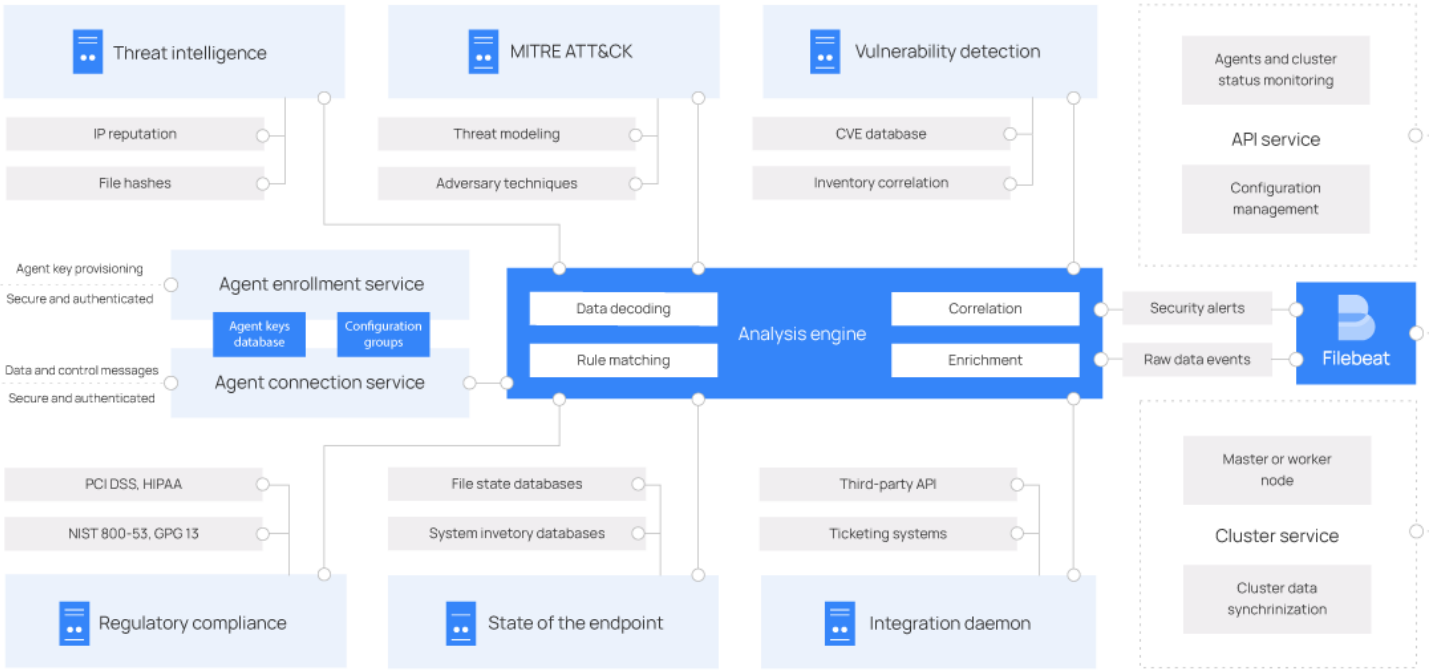


Figure 1.4: Wazuh Server Components [30].

1.5.1.2 Wazuh Indexer

This component is a highly scalable, full-text search and analytics engine. It indexes and stores alerts generated by the Wazuh server, stores data as JSON documents. Wazuh indexers distribute documents across shards, ensuring redundancy and protection against hardware failures. This increases query capacity as nodes are added to a cluster, enhancing overall system performance.
Wazuh uses four different indices to store different event types, these events are outlined in the following table :

Table 1.2: Wazuh Indices for Storing Events.

Index	**Description**
Wazuh-Alerts	Contains alerts generated by the Wazuh manager whenever an event matches a rule with sufficient priority.
Wazuh-Archives	Stores all incoming event data, regardless of whether it triggered an alert. This serves as the full log archive.
Wazuh-Monitoring	Tracks the status of Wazuh agents over time for agent monitoring.
Wazuh-Statistics	Holds performance-related metrics for the Wazuh server. Used to visualize server health and resource usage in the web interface.

1.5.1.3 Wazuh Dashboard

The Wazuh dashboard is the web user interface for data visualization and analysis.It is also used to manage Wazuh configuration and to monitor its status. It includes out-of-the-box dashboards for threat hunting, regulatory compliance detected vulnerable applications, file integrity monitoring data, cloud infrastructure monitoring events, and others.

1.5.1.4 Wazuh Agents

Wazuh agents are installed on endpoints such as laptops, desktops, servers, cloud instances, or virtual machines to provide threat prevention, detection, and response capabilities. They run on operating systems such as Linux, Windows, macOS, Solaris, AIX, and HP-UX. In addition Wazuh platform can monitor agent-less devices such as firewalls, switches, routers, or network IDS, among others.

- **Wazuh Agent Architecture :** Each component in the agent is in charge of its own tasks, including monitoring the file system, reading log messages, collecting inventory data, scanning the system configuration, and looking for malware. Users can manage agent modules via configuration settings, adapting the solution to their particular use cases.
  The diagram below represents the Wazuh agent architecture and its core components, illustrating how its different modules interacts, collect and process the data.

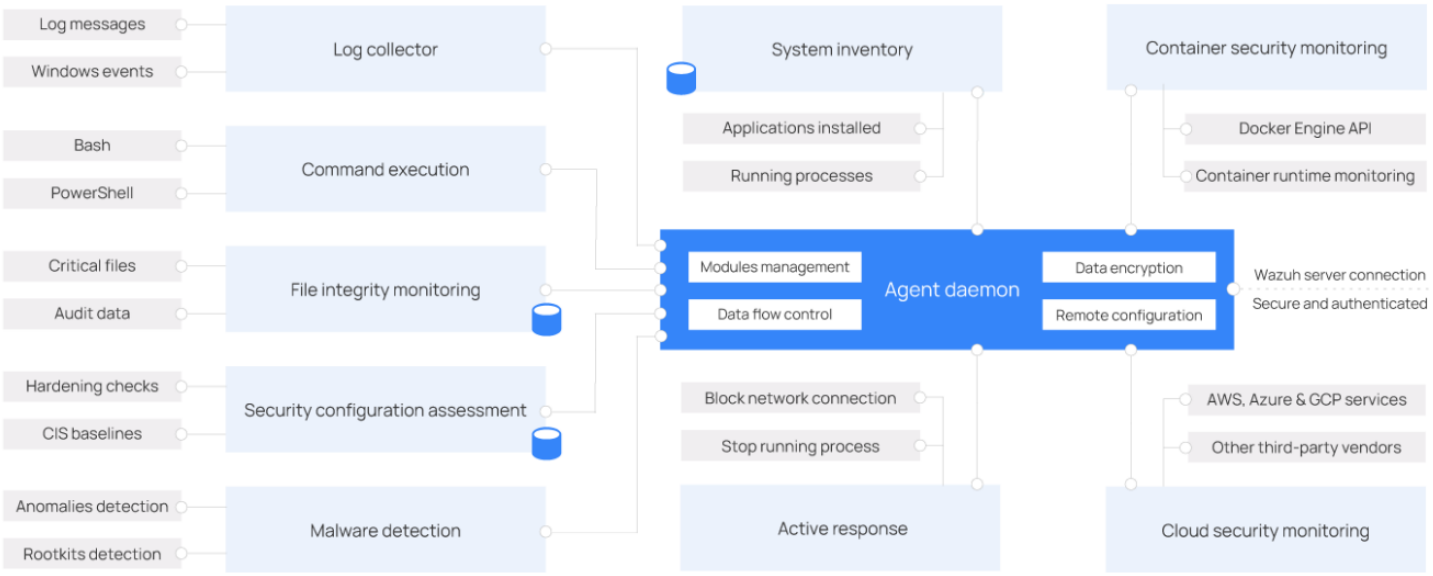


Figure 1.5: Wazuh Agent Components[30].

- **Agent Modules :** include the following points.
  - **Log Collector :** Reads flat log files and Windows events, collecting operating system and application log messages.
  - **File Integrity Monitoring (FIM) :** Monitors the file system, reporting changes in file attributes, permissions, ownership, and content.
  - **Security Configuration Assessment (SCA) :**Provides continuous configuration assessment using out-of-the-box checks based on the Center of Internet Security (CIS) benchmarks.
  - **Malware Detection :** Detects anomalies and the presence of rootkits and looks for hidden processes, files, and ports while monitoring system calls.
  - **Active Response :** Runs automatic actions when threats are detected, triggering responses to block a network connection, stop a running process, or delete a malicious file.
  - **Container Security Monitoring :** Integrated with the Docker Engine API to monitor changes in a containerized environment.

- **Cloud Security Monitoring :** Monitors cloud providers such as Amazon Web Services, Microsoft Azure, or Google GCP.

1.5.2 Communication between Wazuh Components

The Wazuh SIEM consists on several components that need to exchange data to work properly. Below is a description of how these components communicate with each other.

1. **Wazuh Agent - Wazuh Server Communication :** Wazuh agents send events to the server over TCP port 1514. The server decodes each event with its analysis engine, applies rules, and—for events that trigger alerts—writes to 'alerts.json'; non-alerted events can be archived to 'archives.json'.

2. **Wazuh Server - Wazuh Indexer Communication :** The server uses Filebeat to ship alert and event data via TLS to the Wazuh indexer, which by default listens on TCP port 9200. Once indexed, data becomes available for search and visualization.

1.6 The Case Of AI Integration

While Wazuh provides powerful endpoint-centric detection through log analysis and rule evaluation, it lacks native capabilities for handling volumetric or network-layer threats such as DDoS attacks. This limitation stems from its reliance on log-based data rather than direct packet or flow inspection. To overcome this, deep learning models—particularly those designed to analyze flow patterns—can complement Wazuh by dynamically identifying anomalies beyond the reach of static rules. By integrating a DL model trained on network flow data, the system gains the ability to detect complex and high-speed attacks like DDoS in near real time, thus significantly improving overall detection accuracy and scope.

1.7 Conclusion

In this chapter, we presented the fundamentals of cybersecurity showing the importance of protecting digital infrastructures against different cyber threats and attacks. We explored common cyberattacks such as DoS, DDoS, MitM, malware, web-based, and phishing attacks, highlighting their techniques and impacts. To address these challenges, we presented Intrusion Detection Systems (IDS), detailing their types and architecture, also their detection methods. Then we explored SIEMs Security Information and Event Management as a more comprehensive solution for intrusion detection, combining real-time monitoring, data aggregation, and alerting. Special focus on Wazuh SIEM, for its flexibility and efficiency,modular design, and strong integration capabilities , also we introduced the case for integrating IA into Wazuh to overcome limitations in detecting network-level anomalies laying the foundation for its practical use in the following chapters.

Chapter 2

Deep Learning-based Intrusion Detection

2.1 Introduction

Deep Learning is defined as a subset of Machine Learning that is inspired by biological neural networks. It interprets, classifies, and organizes data into different categories, mimicking how the brain processes information [14]. The key characteristic lies in its deep structure, consisting of multiple hidden layers that enable automatic feature extraction from raw data. Which has also made significant contributions to the development of AI-based ids. In this chapter, We will explore deep learning concepts and techniques, focusing on CNNs, RNNs, and LSTM networks. It provides a comprehensive understanding of their functioning and significance in the field.

2.2 Convolutional Neural Networks (CNNs)

A CNN is a type of deep neural network commonly employed for processing structured data like images, inspired by the organization of the biological visual cortex [16]. Its core functionality lies in autonomously acquiring spatial feature hierarchies through adaptive learning mechanisms [17, 18]. Initial layers employ convolutional feature extractors with trainable filters, acting as sliding windows that scan input data. Each CNN layer consists of convolutional kernels that generate distinct feature maps, where neurons in a feature map link to localized regions of the preceding layer.Following successive convolutional and pooling layers, fully connected layers are typically appended to finalize classification tasks [19].
Figure 2.1 provides a simplified overview of the layers in a Convolutional Neural Network (CNN), which will be explained in detail in the following section.

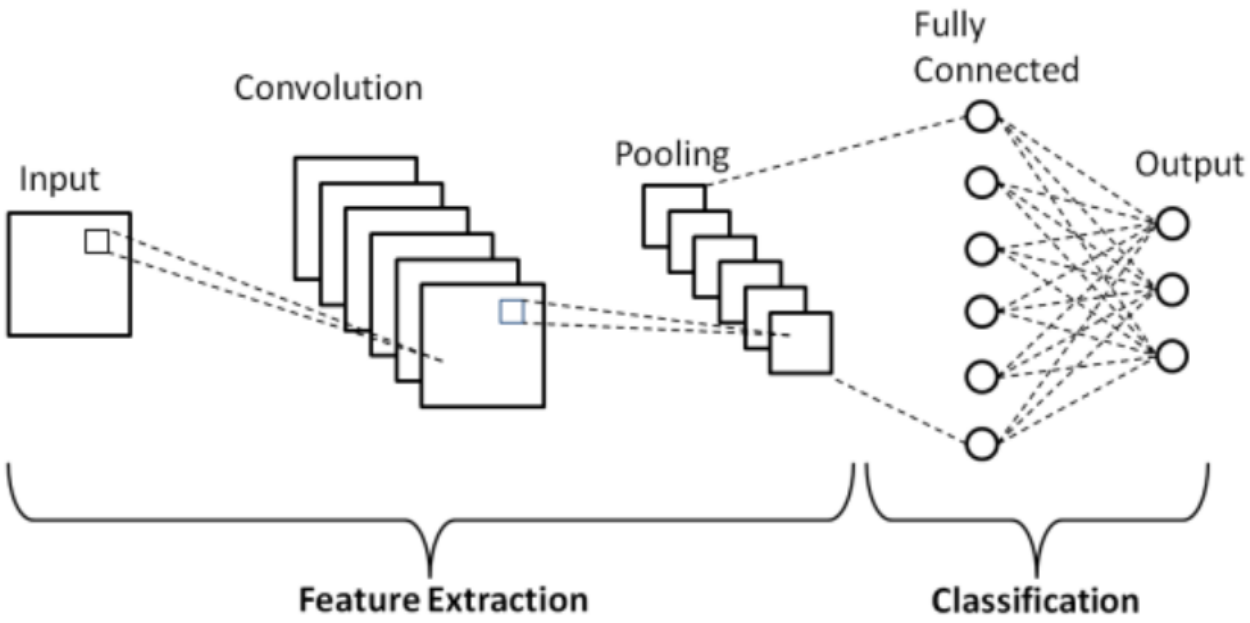


Figure 2.1: Simplified CNN Layers

2.2.1 Convolution Layer

Convolutional layers enable neural networks to analyze features and extract them within localized regions instead of processing complete datasets. They integrate both linear transformations (convolution) and nonlinear activations, such as ReLU or sigmoid functions [17]. Filters (kernels) applied during convolution preserve spatial relationships between pixels through learned feature representations. The outputs of these operations—termed feature maps—capture hierarchical patterns. During training, kernel values (weights) adapt dynamically to optimize feature learning by adjusting connection strengths between neurons. The spatial dimensions of feature maps are governed by three parameters: depth (number of filters), stride (pixel increments for kernel sliding), and zero-padding (border augmentation with zeros)[19].

2.2.2 Pooling Layer

The pooling layer's primary function is to downsample feature maps produced by preceding convolutional layers. This process reduces the dimensionality of large feature maps while preserving critical features. Similar to convolutional operations, pooling employs predefined kernel sizes and stride values[20].The two most prevalent methods are Max-Pooling and Average Pooling. [18, 21]. By reducing feature map dimensions, pooling enhances spatial invariance to distortions, minimizes learnable parameters [17], and combats overfitting by focusing on dominant patterns. [19].

2.2.3 Activation Function

Activation functions map inputs to outputs in neural networks by computing weighted summations of inputs and biases. They determine whether a neuron activates ("fires") for a given input, enabling non-linear

decision-making. In CNNs, non-linear activation layers follow all learnable layers (e.g., convolutional, fully connected), allowing the network to model complex patterns. Critically, these functions must be differentiable to support error backpropagation during training. Below are common activation functions and their properties:

- **Sigmoid :** Squashes real-number inputs into a range of [0, 1] via an S-shaped curve (Eq. 2.1)

$$f(x)_{\text{sigm}} = \frac{1}{1+e^{-x}} \tag{2.1}$$

- **Tanh :** Similar to sigmoid but outputs values between [-1, 1] (Eq. 2.2).

$$f(x)_{\tanh} = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \tag{2.2}$$

- **ReLU (Rectified Linear Unit) :**The most common activation function used in CNN context .It outputs positive inputs directly and zeros negatives (Eq. 2.3). While computationally efficient, ReLU risks "Dying ReLU," where neurons permanently deactivate due to large negative gradients during training.

$$f(x)_{\text{ReLU}} = \max(0,x) \tag{2.3}$$

2.2.4 Fully Connected Layer

The FC layer, typically positioned at the end of a CNN, connects every neuron to all neurons in the preceding layer. This "dense" connectivity mimics traditional multilayer perceptrons (MLPs)[20], serving as the network's classifier. Inputs to the FC layer are flattened feature maps converted into 1D vectors that encode high-level patterns extracted earlier in the architecture.
Each connection between neurons in the FC layer employs learnable weights, enabling the network to map features to class probabilities. For classification tasks, the final FC layer often feeds into a SoftMax classifier (or alternatives like sigmoid for binary tasks), where n neurons correspond to n output classes.[17, 19, 21].

2.3 Recurrent Neural Networks (RNNs)

Unlike feed-forward networks, RNNs introduce cyclic connections where neuron outputs loop back as inputs to themselves or other neurons. This architecture allows RNNs to model sequential data (e.g., time series, text) by maintaining an internal "memory" of previous inputs [22]. In cybersecurity applications like intrusion detection systems (IDSs), RNNs excel at identifying temporal patterns (e.g., correlations between attack behaviors over time), complementing CNNs, which focus on spatial features[19].

2.3.1 RNNs Structure

RNNs are designed to process sequential data based on memorizing the previous inputs through recurrent connections. As shown in Figure 2.2 that describes the layout of a basic RNN, the RNN structure is based on three layers [23]:

- **The Input Layer :** At each time step the input layer receives the input vector x(t). Inputs can be categorized as one-hot encoded or word embeddings, depending on the task.
- **The Recurrent (Hidden) Layer :** This is the core of the RNN structure. At each time step t, the hidden state h(t) is computed not only from the current input x(t), but also from the previous hidden state h(t-1). This recurrence creates what is called a feedback loop that captures temporal dependencies within the data.
- **The Output Layer :** This layer produces the final output for each time step, depending on the task (e.g., classification or sequence prediction).

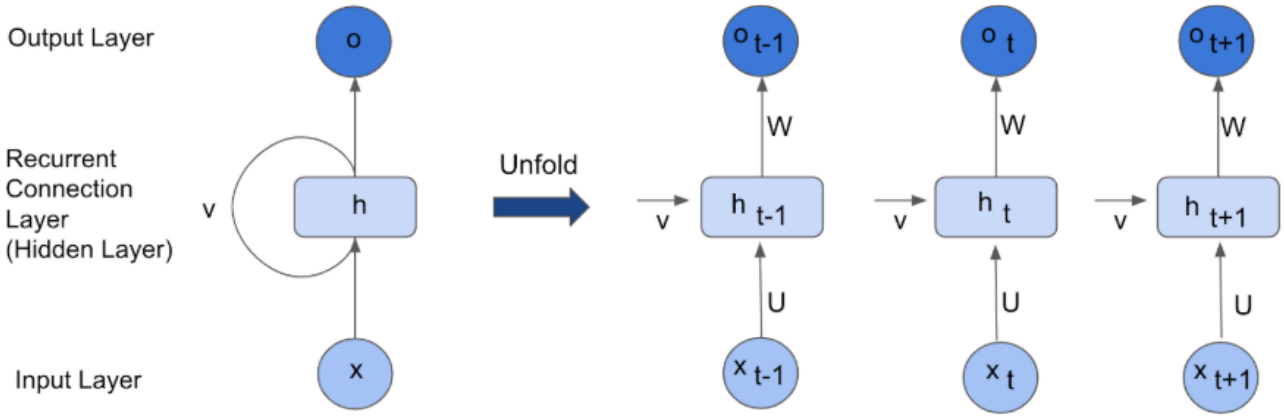


Figure 2.2: The layout of a basic RNN

2.3.2 Long Short Term Memory (LSTM)

Long Short-Term Memory (LSTM) is a variant of Recurrent Neural Networks (RNN). The key difference is that unlike traditional RNNs, which struggle with vanishing gradients over long sequences [24]. LSTM are designed to address the issue of long-term dependency retention by integrating gating mechanisms to regulate the information flow. This architecture enables LSTMs to maintain relevant past information over extended time steps, making them highly effective for sequential data tasks such as natural language processing and time-series prediction [25, 15].

2.3.3 LSTM Structure

The memory cell - as shown in figure 2.3 - represents the state of the LSTM. An LSTM architecture consists of three key gates: the input gate, forget gate, and output gate.

- **The Input Gate :** determines how new information updates the cell state.
- **The Forget Gate :** controls the extent to which past information is retained or discarded.

- **The Output Gate :**regulates how the internal state influences the final output.
  The output gate is defined by:

$$O(t) = \sigma\left(\mathbf{b} + \mathbf{U} \cdot \mathbf{X}(t) + \mathbf{W} \cdot \mathbf{h}(t-1)\right) \tag{2.4}$$

  where:

  - $O(t)$: The output gate activation at time step t,
  - σ: The sigmoid activation function,
  - $\mathbf{b}$: The bias vector for the output gate (learned during training),
  - $\mathbf{U}$: The weight matrix applied to the current input $\mathbf{X}(t)$ for the output gate,
  - $\mathbf{X}(t)$: The input vector at time step t,
  - $\mathbf{W}$: The weight matrix applied to the previous hidden state $\mathbf{h}(t-1)$,
  - $\mathbf{h}(t-1)$: The previous hidden state.

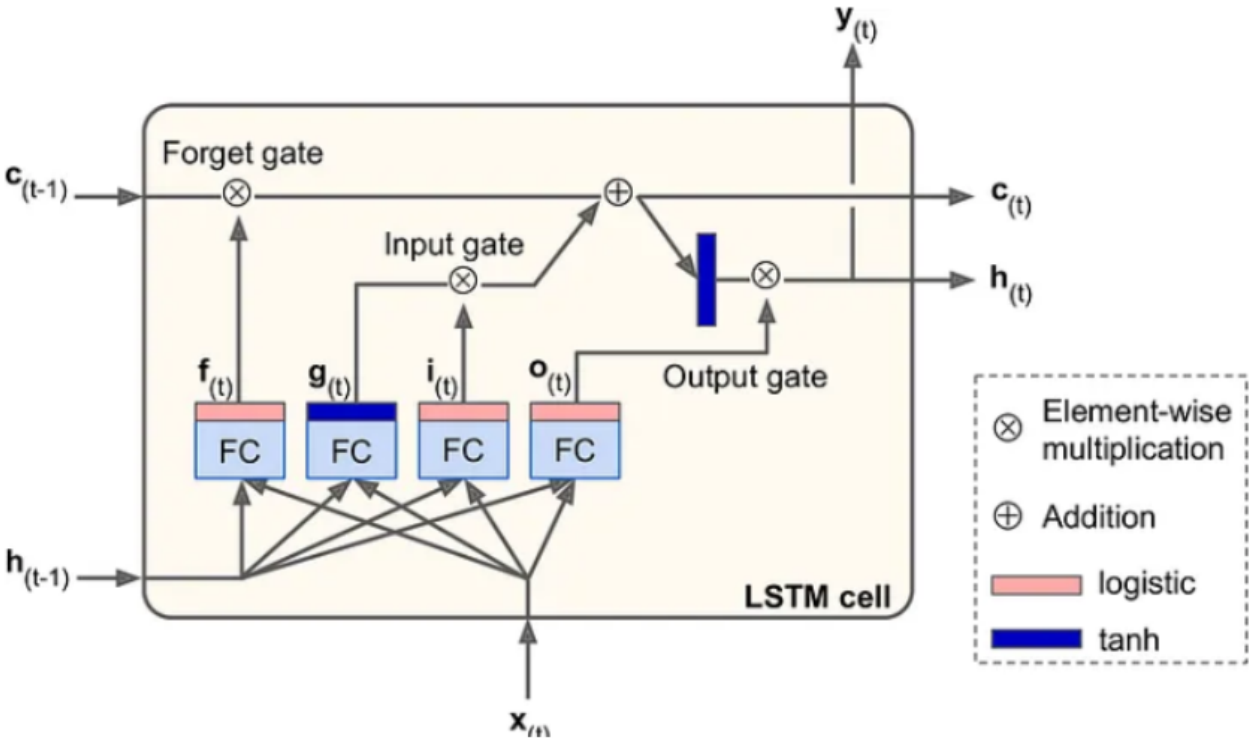


Figure 2.3: LSTM Cell Structure

These mechanisms allow LSTMs to selectively filter and maintain important information, improving performance in tasks requiring long-term dependencies [25].
In LSTM, the input and output are vectors of the same size. The forget gate decides which past information to keep by combining the current input X(t) and the previous hidden state h(t-1). The input gate adds new information to the cell state, while the forget gate removes unnecessary parts. The updated cell state C(t) is used to generate the output. The output gate applies a sigmoid function and multiplies it with the tanh of the new cell state to produce the hidden state h(t), which is the LSTM's output [26].

2.4 Hybrid CNN-LSTM Model

The hybrid CNN-LSTM deep learning model combines and leverages the strengths of both architectures CNN and LSTM networks . First CNN layers are used to extrac high-level spatial features from inputs data through convolution and pooling layers. Then these features are passed to LSTM layers, to capture temporal dependencies. This combination allows the model to learn both spatial and sequential patterns, making it ideal for time-series classification tasks such as NIDS. The final classification is typically performed using the fully connected layers with an activation function like SoftMax or sigmoid [26].
The architecture below presented in figure 2.4 is an example of a hybrid CNN-LSTM model, structured in three repeating stages that combine convolutional and recurrent layers for sequential feature extraction and classification.

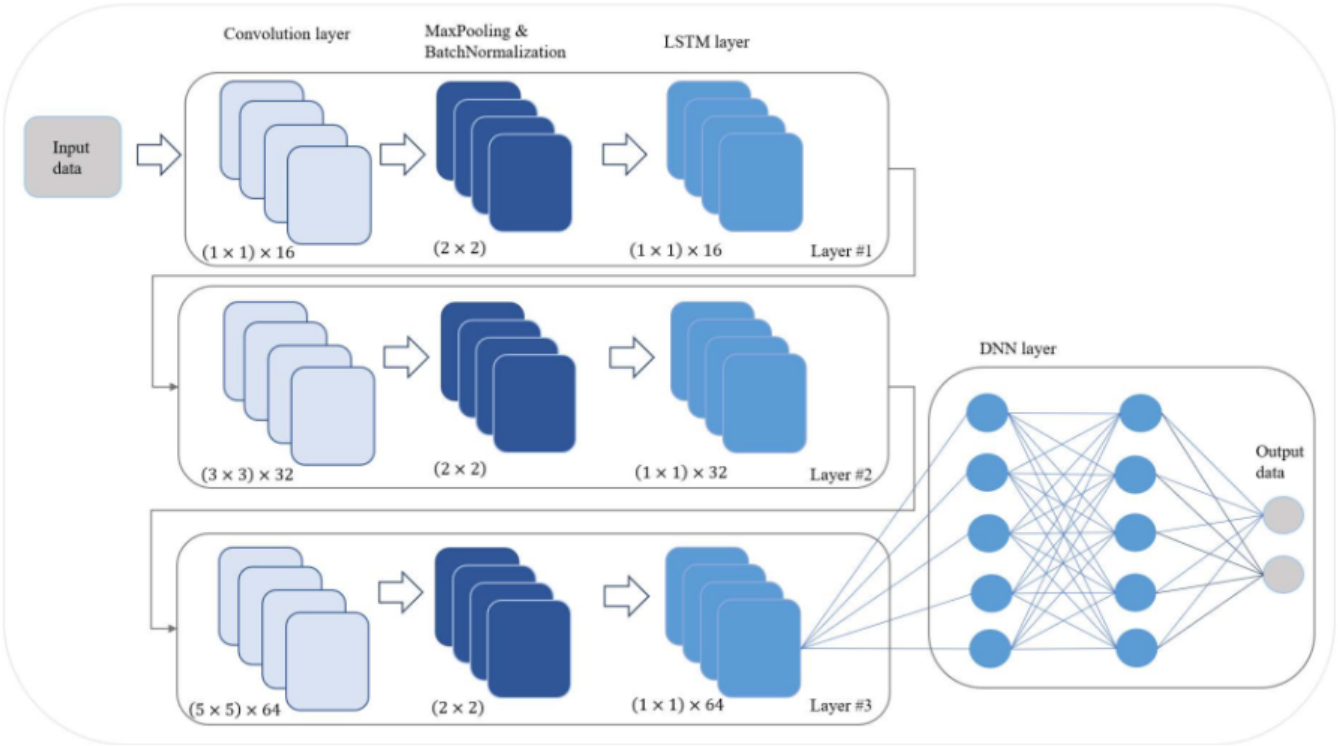


Figure 2.4: An example of a Hybrid CNN-LSTM Model [26]

2.5 Conclusion

This chapter provided an overview of deep learning approaches for IDS, focusing on CNNs, RNNs, and LSTMs roles and architectures. As a result CNNs were effective in extracting spatial features from network traffic, while LSTMs are designed for learning temporal dependencies, which are important for detecting sequential patterns. Then we explored the hybrid CNN-LSTM model that combines the strengths of both models to enhance intrusion detection accuracy. By leveraging these models, IDS can better identify complex and evolving threats such as DDoS attacks.

Chapter 3

Methodology

3.1 Introduction

This chapter outlines the overall methodology used to design and implement our Hybrid AI-SIEM Framework. It briefly presents the architectural structure, component choices, and integration strategies that support the combination of traditional rule-based detection with artificial intelligence. The approach includes containerized deployment using Docker, the use of a CNN-LSTM model for anomaly detection, and integration with Wazuh and complementary tools. The specific technologies, configurations, and design decisions are discussed in detail in the following sections.

3.2 Project Overview : The Hybrid AI-SIEM Framework

Based on the study of various available solutions presented, and after conducting a comparative analysis in terms of operating systems, deployment, advantages, and limitations, we have chosen to implement this project using Wazuh. It's open-source, flexible, and integrates well with other security tools, making it a cost-effective choice without compromising on essential features. It provides strong capabilities like log analysis, file integrity monitoring, Incident response ,regulatory compliance and intrusion detection, which are crucial for effective security monitoring. While it may need some fine-tuning to optimize detection for advanced threats, its scalability and active community support make it a solid choice for both cloud and on-premise environments.

The framework integrates the Wazuh SIEM platform with a custom-built Convolutional Neural Network–Long Short-Term Memory (CNN-LSTM) model for advanced detection of distributed denial-of-service (DDoS) attacks. This architecture aims to strengthen threat identification and response mechanisms by fusing signature-based monitoring with machine learning–driven anomaly detection. The system architecture consists of the following key layers as illustrated in Figure 3.1:

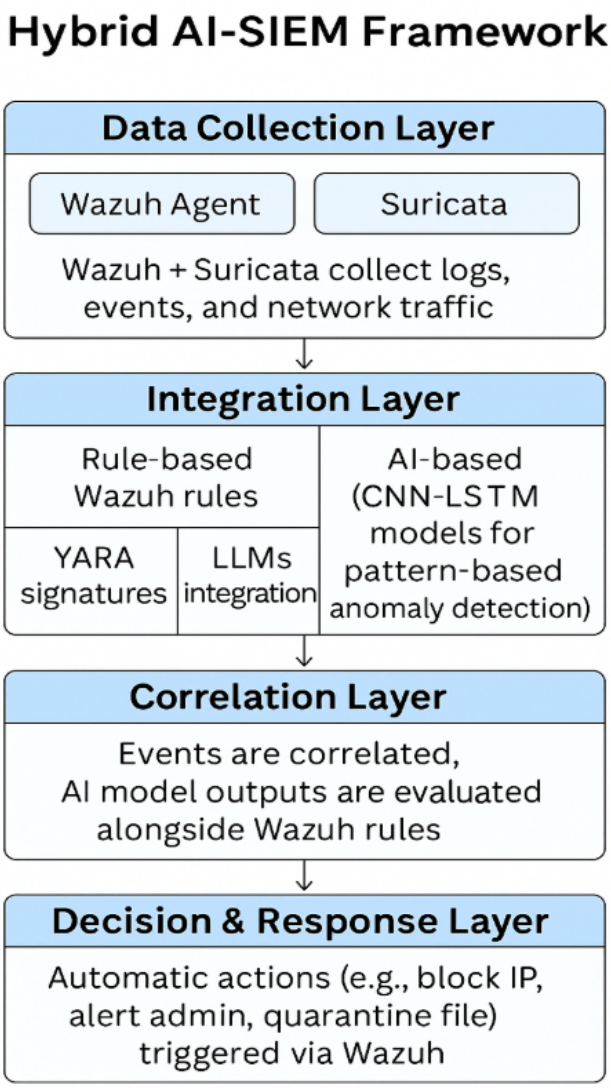


Figure 3.1: Hybrid AI-SIEM Framework Architecture.

3.2.1 Data Collection Layer :

Handled by Wazuh Core Components and IA-Based detection Modules

1. **Wazuh SIEM :** The backbone of the system, includes:
   - **Wazuh Manager :** Central engine for log analysis, rule evaluation, and alert generation.
   - **Wazuh Agents :** Installed on monitored endpoints to collect security event data and forward it to the manager.
   - **Decoders :** Parse incoming logs into structured formats.
   - **Rules :** Detect known patterns and generate alerts based on predefined conditions.
   - **Active Response Module :** Automatically executes actions when specific alerts are triggered.
2. **AI-Based Detection Module (CNN-LSTM) :**This deep learning model is designed to detect anomalous patterns indicative of DDoS attacks. It processes network traffic features, identifies attack signatures not easily captured by static rules, and generates a detection verdict that is injected into the Wazuh alerting pipeline.

3.2.2 Integration Layer :

This connects the AI module with Wazuh. The CNN-LSTM detection output is formatted and fed into Wazuh either through custom decoders or as enriched alert metadata, allowing Wazuh to act on the model's predictions alongside its native rules. In addition to the AI module, multiple integrations were incorporated into Wazuh to enhance detection capabilities and expand coverage of threat vectors:

- **Suricata Integration :**Network Intrusion Detection System (NIDS) integration to detect network-based attacks such as port scans, exploit attempts, and suspicious traffic patterns.
- **YARA Integration :** Deployed to identify malware and suspicious files based on custom and community-provided YARA rulesets.
- **Large Language Models (LLMs) :** Leveraged to enrich alerts by providing contextual explanations, summarizing potential threats, and assisting in prioritizing responses.

3.2.3 Correlation Layer :

This layer merges outputs from both traditional rule-based detection and AI-based analysis. Events are correlated to identify complex attack patterns, with AI model predictions evaluated alongside Wazuh-generated alerts to improve accuracy and reduce false positives.

3.2.4 Decision and Response Layer :

Facilitates alert visualization, automated responses, and analyst-driven actions. Wazuh dashboards, response modules (such as blocking malicious IPs or isolating hosts) are central in supporting rapid incident response and mitigation. The diagram below illustrates the high-level architecture and interaction between components:

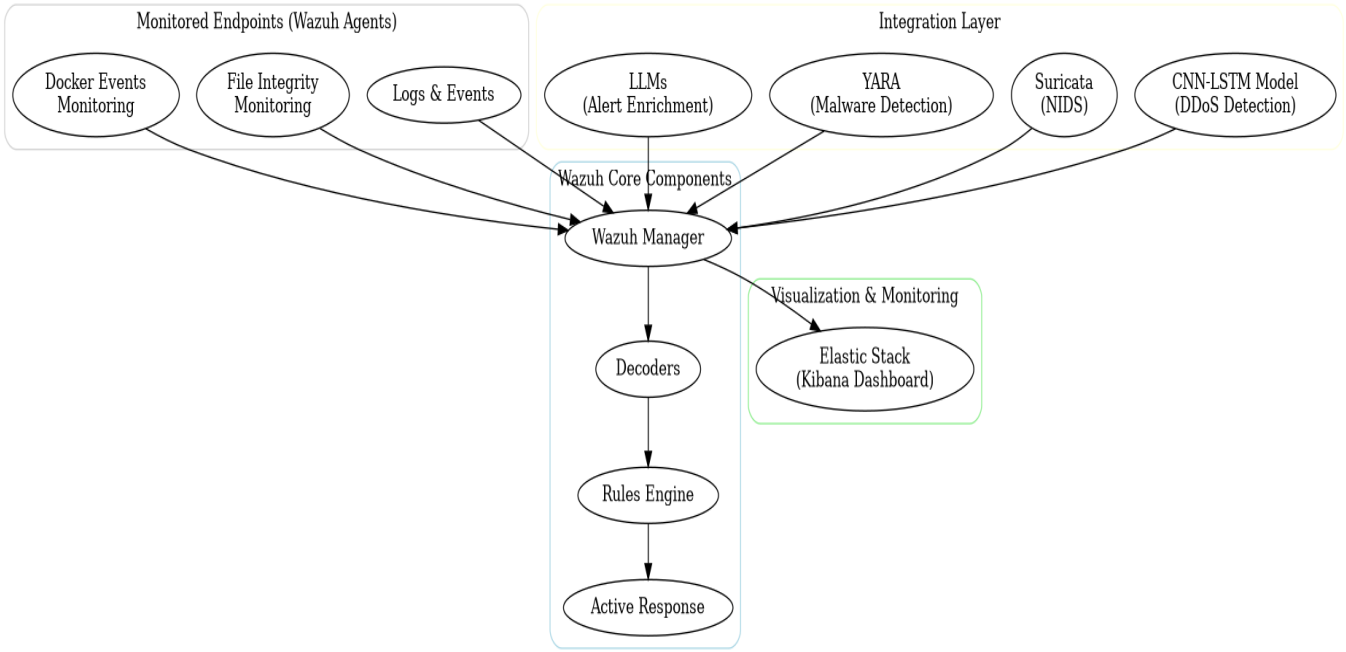


Figure 3.2: Hybrid AI-SIEM Framework High Level Layers Architecture

3.3 Experimental Environment and Design Rationale

The following sections of this chapter, outlines the methodological choices related to the environment configuration, tool selection and model structure for the proposed security monitoring framework.

3.3.1 Docker and Docker Compose

To deploy the Wazuh platform efficiently, we used Docker and Docker Compose. Wazuh consists of multiple component such as the manager, indexer, and dashboard which benefit from being containerized into isolated, lightweight environments. This approach avoids conflicts and simplifies dependency management improving resource utilization, especially on machines with limited CPU or memory.

Docker is an open platform for developing and running applications that enables the separation of applications from the underlying infrastructure, allowing for faster and more reliable software delivery. With docker, infrastructure can be managed similarly to application code [28].

Docker Compose is a tool that makes it easy to define and manage multi-container applications. Instead of starting each container manually, we can describe the entire application stack services, networks, and volumes in a single YAML file called docker-compose.yml [28]. The docker compose architecture is described in figure 3.3.

Docker Compose was used to define and orchestrate the entire wazuh stack through a single configuration file, making the deployment process faster, reproducible, and easier to scale or test across different environments.

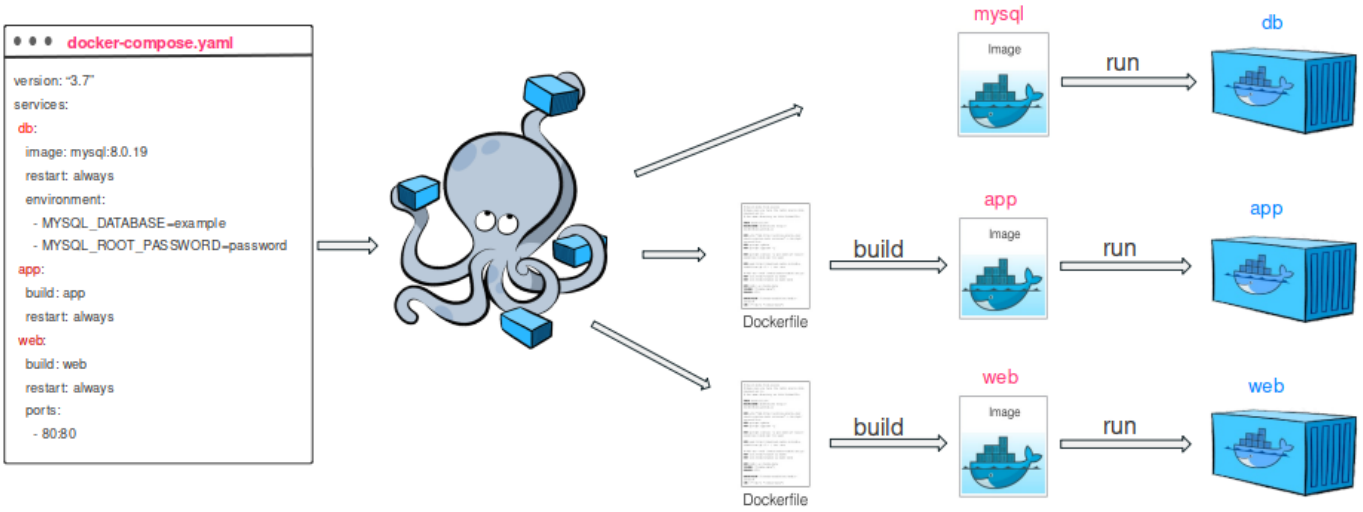


Figure 3.3: Docker-compose architecture.

3.3.2 Network-infrastructure Simulation

In order to test and validate the security monitoring architecture proposed for this project, a comprehensive network architecture was developed using Docker and its networking technology. The simulation accurately reflects the planned real-world environment by incorporating layered firewalls, distinct network segments (DMZ, internal LAN, and management LAN), and the Wazuh security monitoring components deployed within a cluster. The simulated network is presented in Figure 3.4.

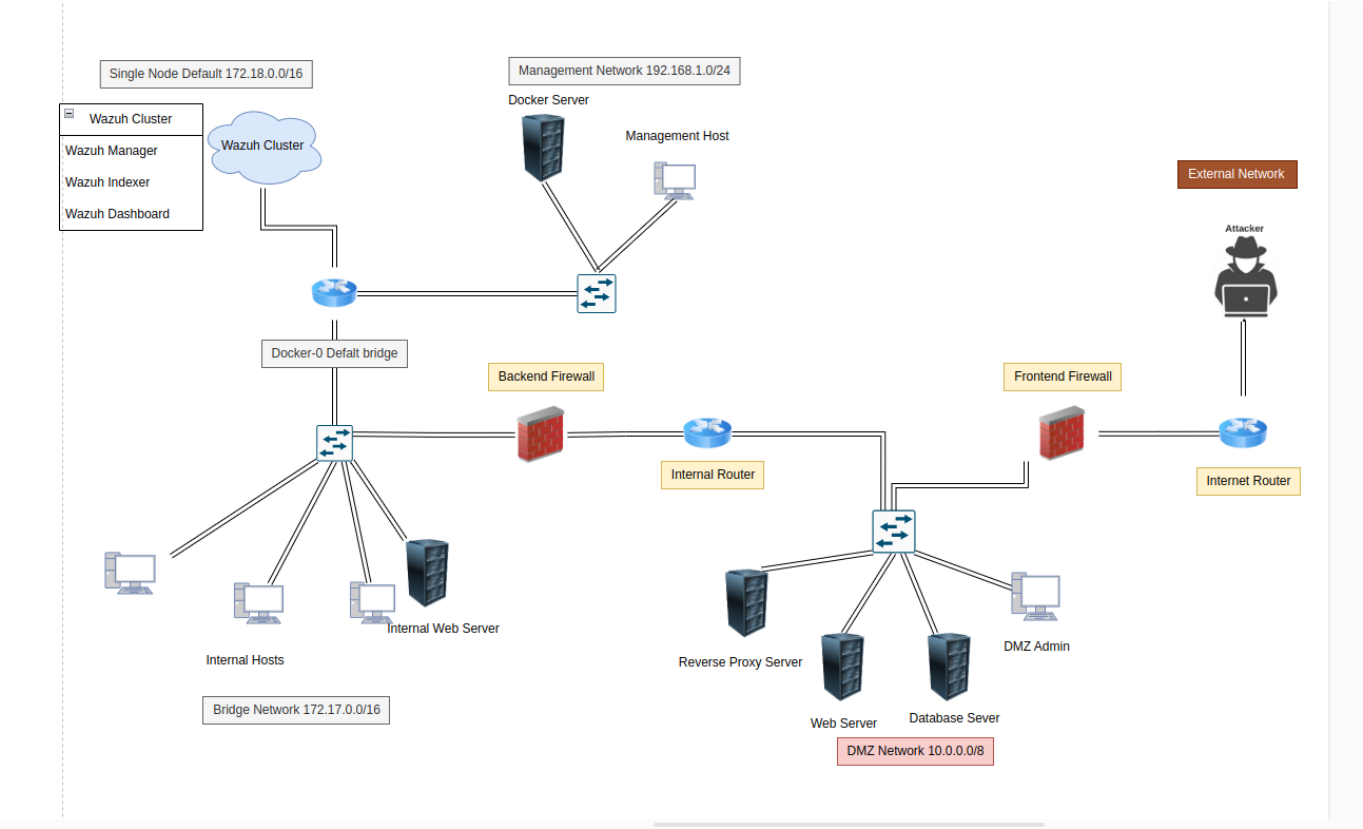


Figure 3.4: Simulated Network infrastructure.

1. **Management Network (192.168.1.0/24) :**Dedicated to administrative access, monitoring, and control of critical infrastructure, it contains :
   - **Management Host :** Ubuntu-based system for administrative tasks (e.g., SSH, Wazuh dashboard access).
   - **Docker server :** Hosts internal services, including a web server and bridge interfaces for network segmentation.
2. **Internal Network (172.16.0.0/24)**
   - **Internal Servers :** Various applications, databases, web servers, or file servers reside in this zone, representing the sensitive part of the company.
   - **Backend Firewall :** Placed between the internal network and the Wazuh cluster, this firewall applies strict rules to control lateral movement and ensure sensitive data is protected.
3. **Wazuh Cluster (172.18.0.0/16) :**
   - **Wazuh Manager :** Centralized system responsible for analyzing security alerts from agents.
   - **Wazuh Indexer :** Stores and indexes the event data from the manager for analysis and correlation.
   - **Wazuh Dashboard :** Web interface (often Kibana-based or Wazuh UI) providing real-time monitoring of security alerts and system status.
4. **DMZ Network (10.0.0.0/24) :**

- **Public-Facing Services :** Web servers and reverse proxies (e.g., NGINX) are hosted in this zone, which is exposed to simulated external traffic.
- **Frontend Firewall :** This firewall sits at the boundary between the DMZ and the simulated Internet, filtering malicious external requests before they reach public services

5. **External Segment :**

- **External Router :** Simulates the Internet by generating external traffic aimed at the DMZ.
- **External Host:** can launch attacks against the DMZ to test the effectiveness of the firewall rules and the alerting capabilities of the Wazuh cluster.

3.4 AI-Based Detection Module (CNN-LSTM)

Traditional rule-based systems, while effective for known patterns, struggle with evolving or high-volume attacks like DDoS. To address this limitation, we integrated a deep learning model based on a CNN-LSTM architecture, trained to detect abnormal patterns in network traffic that indicate DDoS activity. The model complements Wazuh's rules by providing adaptive, data-driven anomaly detection.

3.4.1 Dataset

The model was trained on the CIC-IDS 2017 dataset, a benchmark dataset containing labeled traffic for various attack types, including multiple forms of DDoS. We used the Wednesday dataset . The Wednesday dataset from the CIC-IDS2017 collection is a critical component for training and evaluating intrusion detection systems, particularly for Denial of Service (DoS) attack detection. It offers a comprehensive set of labeled network traffic data that simulates real-world scenarios.
It consists of approximately 12 GB of pcap files and 272 MB of CSV files and total flows is assumed to be around 692,703 network flows it includes several ddos types attacks such as DoS Hulk, DoS GoldenEye, DoS Slowloris, DoS, SlowHTTPTest, Heartbleed, each record in the dataset includes over 80 features extracted using CICFlowMeter. The table 3.1 bellow provides a summary of dataset characteristics :

Table 3.1: Dataset Characteristics Summary

Attribute	**Detail**
Data Volume (PCAP)	Approximately 12 GB
Data Volume (CSV)	272 MB
Total Flows	≈ 692,703 network flows
Attack Types Included	DoS Hulk, DoS GoldenEye, DoS Slowloris, DoS SlowHTTPTest, Heartbleed
Number of Features per Flow	80 features
Feature Categories	• Flow Duration: time span of the flow • Packet Statistics: counts & sizes both ways • Flow Rates: bytes/sec & packets/sec • Header Information: flags, protocols, etc. • Label: benign vs. specific attack type

3.4.1.1 Dataset Cleaning

Several steps of preprocessing were applied to clean the dataset :

1. **Column Cleanup :** Whitespace was removed and column names were standardized for consistency.
2. **Invalid Values :**Negative, infinite, and missing values were detected and removed to ensure valid numeric inputs.
3. **Redundancy Reduction :** Duplicate rows and columns with constant or identical values were dropped.
4. **Irrelevant features :** Columns related to idle and active times were removed due to low relevance to DDoS detection.
5. **Label Validation :** The label column was preserved, standardized, and its class distribution reviewed post-cleaning.

After cleaning, the dataset was saved as a csv file for model training. The following figure shows the label distribution of the dataset before and after applying the cleaning process on the dataset

```
=== Label Distribution ===
 Label
BENIGN              440031
DoS Hulk            231073
DoS GoldenEye        10293
DoS slowloris         5796
DoS Slowhttptest      5499
Heartbleed              11
Name: count, dtype: int64
=== Label Distribution after Cleanning ===
label
BENIGN              176892
DoS Hulk            163457
DoS GoldenEye         7709
DoS slowloris         3746
DoS Slowhttptest      2083
Heartbleed               7
Name: count, dtype: int64
```

Figure 3.5: Label Distribution Before and After Cleaning the Dataset.

3.4.1.2 Handling Class Imbalance

Standard classification models tend to bias toward the majority class. This created an issue of imbalance between 'BENIGN' and 'ATTACK' samples. To expose the model equally to both benign and attack patterns during training , SMOTE (Synthetic Minority Over-sampling Technique) was applied to the training set. This ensured that the model learned from a balanced distribution of classes without compromising test data integrity. Rather than duplicating existing minority samples, SMOTE creates new examples that spread throughout the minority feature space. For each minority sample, randomly choose one of its k neighbors and create a new "synthetic" point along the line segment joining them and it repeats until the minority class count matches the majority class. The following figures show the class distribution before and after dropping the irrelevant features, grouping the attacks and applying Smote.

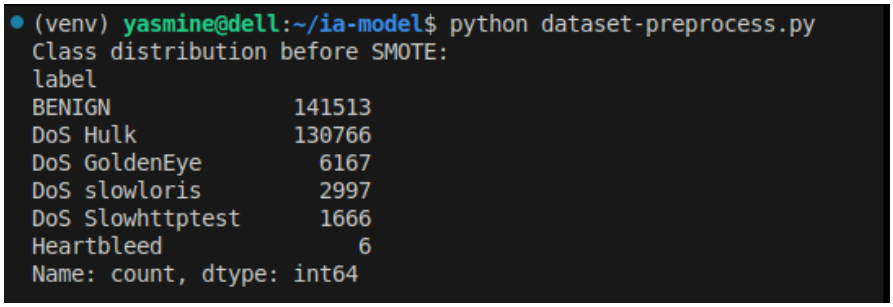


Figure 3.6: Class Distribution Before SMOTE.

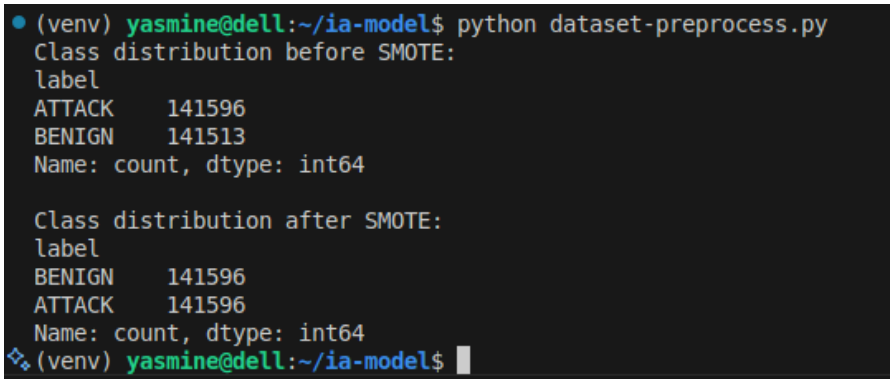


Figure 3.7: Class Distribution After SMOTE.

3.4.1.3 Feature Selection

Feature selection is a critical preprocessing step in machine-learning pipelines because it helps to simplify models, reduce overfitting, and improve both training efficiency and interpretability. By removing irrelevant or redundant variables, we lower the model's complexity and its capacity to memorize noise in the data, leading to better generalization on unseen samples.
In our dataset , we applied two feature selection methods : Chi-Square Test and Mutual Information (Information Gain Ratio).

- **Chi-Square Statistic :** It tests the independence between each feature and the class label. A high X^2 score means the feature's observed values deviate strongly from what you'd expect if it were unrelated to the label.
  Its application consist of :

  1. Discretize any continuous features (X^2 requires non-negative integer counts).
  2. Use sklearn.feature-selection.SelectKBest(chi2, k=35) to compute X^2 for each feature and pick the top 35.
  3. These 35 features are those whose value distributions differ most between Benign and Attack flows.

- **Mutual Information (Information Gain Ratio) :** mutual Information (MI) quantifies how much knowing the feature reduces uncertainty about the label and the Information Gain Ratio (IGR) normalizes MI by the feature's own entropy, penalizing very high-entropy features.
  Its application consist of :

1. Discretize each continuous feature into 10 uniform bins via KBinsDiscretizer.
2. Compute mutual-info-classif(..., discrete-features=True) to get an IGR-style score for each feature.
3. Sort features by descending MI score and select the top 35.

The intersection of the top 35 features from both methods was used for training. (features that are both statistically dependent and highly informative). This reduces false positives from either method alone and focuses on the most robust predictors.
The following screenshot shows the final selected feature after applying intersection between the two methods:

```
Final Selected Features: ['bwd_packet_length_std', 'packet_length_std', 'bwd_packet_length_mean',
 'avg_bwd_segment_size', 'bwd_packet_length_max', 'packet_length_variance', 'average_packet_size'
, 'packet_length_mean', 'fwd_iat_max', 'flow_iat_max', 'max_packet_length', 'fwd_iat_std', 'flow_
iat_std', 'flow_iat_mean', 'fwd_iat_total', 'flow_duration', 'fwd_iat_mean', 'init_win_bytes_forw
ard', 'destination_port', 'bwd_iat_total', 'bwd_iat_max', 'bwd_iat_std', 'bwd_iat_mean', 'init_wi
n_bytes_backward', 'bwd_iat_min']
(venv) yasmine@dell:~/ia-model$
```

Figure 3.8: Final Selected Features.

3.4.1.4 Data Scaling and Reshaping

To prepare the data for neural network training, first features were standardized using StandardScaler. Standardizing our features to zero mean and unit variance ensures that each input dimension contributes equally to the gradient updates, which dramatically speeds up convergence and helps avoid getting stuck in poor local minima—especially important for deep nets that use gradient-based optimizers.
Reshaping to (samples, features, 1) then packages each flow's feature vector as a "single-channel" 1D image, which is exactly what Keras's Conv1D and LSTM layers expect:

- **Conv1D** slides its kernels over the feature dimension, learning local patterns across features.
- **LSTM** then treats the output sequence of these local feature maps as time steps, capturing any temporal dependencies

Together, standardization plus this shape guarantees the network sees data in the right scale and format to learn both spatial (feature) and temporal (sequence) relationships effectively.

3.4.2 Overview of Model Architecture

The proposed hybrid CNN-LSTM architecture is designed to jointly learn spatial-temporal patterns from sequential network flow data. The model combines convolutional layers for local feature extraction and LSTM layers for temporal dependency modeling, optimized for binary classification tasks (e.g., attack detection).
The Figure 3.9 presents a simplified diagram of the model architecture

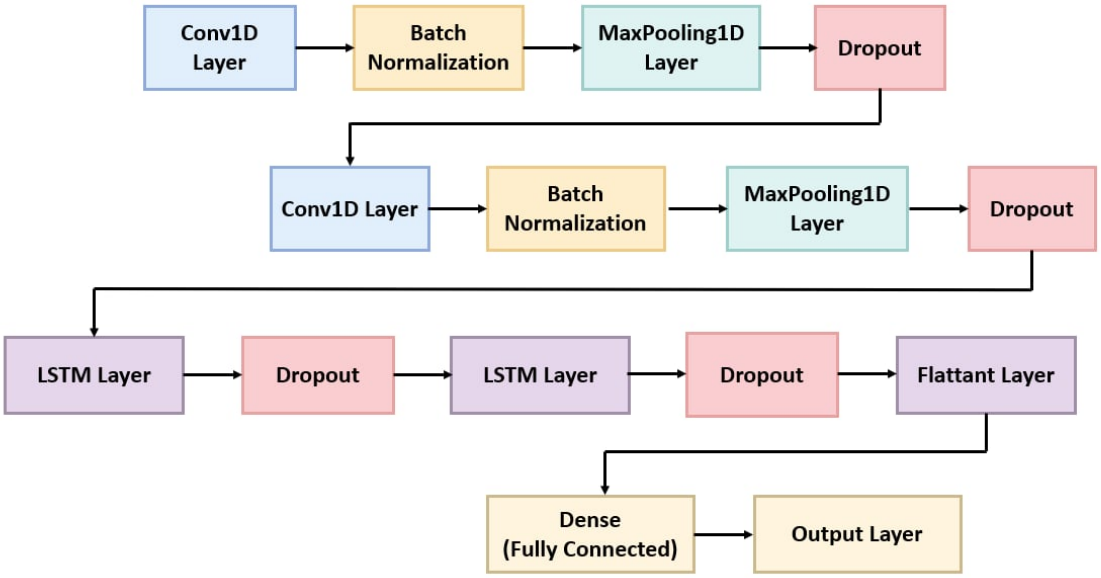


Figure 3.9: Simplified CNN-LSTM model Architecture.

The architecture employs a dual-stage approach: spatial features are first extracted via stacked convolutional blocks, followed by hierarchical LSTM layers to model temporal relationships in network flow sequences. Progressive dropout rates ($0.3 \rightarrow 0.5$) mitigate overfitting risks inherent in security datasets. Precision and recall metrics are prioritized during training to address class imbalance in attack detection scenarios.
The following table outline details on the architecture layers of the cnn-lstm model.

Table 3.2: Architecture Layers of the CNN–LSTM Model

Layer Type	**Parameters**
Input	Raw network flow features
Conv1D	64 filters, kernel size = 3, ReLU activation
Batch Normalization	-
MaxPooling1D	Pool size = 2
Dropout	Rate = 0.3
Conv1D	128 filters, kernel size = 3, ReLU activation
Batch Normalization	-
MaxPooling1D	Pool size = 2
Dropout	Rate = 0.5
LSTM	100 units, return_sequence = True
Dropout	Rate = 0.3
LSTM	50 units, return_sequence = False
Dropout	Rate = 0.4
Flatten	-
Dense	1 unit, sigmoid activation (binary classification)

The training configuration for compiling is outlined in the following table:

Table 3.3: Training Configuration

Attribute	**Configuration**
Optimizer	Adam (adaptive learning rate)
Loss	binary_crossentropy (for two-class detection)
Metrics	Accuracy, Precision, Recall
Early Stopping	monitor = val_loss, patience = 3, restore_best_weights = True

- Using Adam optimizer leverages adaptive per-parameter learning rates and momentum estimates to converge more quickly and robustly across complex, non-convex loss surface, making it a practical default for deep networks. We pair it with binary cross-entropy, which directly optimizes the log-likelihood of correct class probabilities in a two-class setting, ensuring stable gradients when predicting attack vs. benign. Monitoring accuracy, precision, and recall gives a more complete evaluation under class imbalance, accuracy for overall fit, precision for false-positive control, and recall for capturing as many real attacks as possible.
  Finally, EarlyStopping on validation loss with patience=3 and restore-best-weights=True halts training at the sweet spot before overfitting, then rolls back to the best epoch to maximize generalization.

3.5 Training and Evaluation of CNN-LSTM Model

The CNN-LSTM model was trained with early stopping enabled to prevent overfitting. The model was trained on 80% of the SMOTE-balanced dataset, while the remaining 20% of training data was used for validation.
The final training epoch yielded the following.

⇒ Training Accuracy: 98.45

⇒ Validation Accuracy: 99.59

⇒ Training Loss: 0.0350

⇒ Validation Loss: 0.0222

⇒ Precision: 98.31% (Train), 99.65% (Val)

⇒ Recall: 98.60% (Train), 99.52% (Val)

These results demonstrate excellent generalization and robustness of the model, especially in detecting high-volume DDoS traffic with very low false positive and false negative rates.

Figure 3.10 illustrates the evolution of training and validation accuracy and loss across epochs. As shown:

- Accuracy steadily increases for both training and validation sets.

- Loss consistently decreases without sudden spikes or divergence, indicating stable learning behavior and minimal overfitting.
- The validation performance consistently remained slightly ahead of training, which suggests well-regularized generalization.

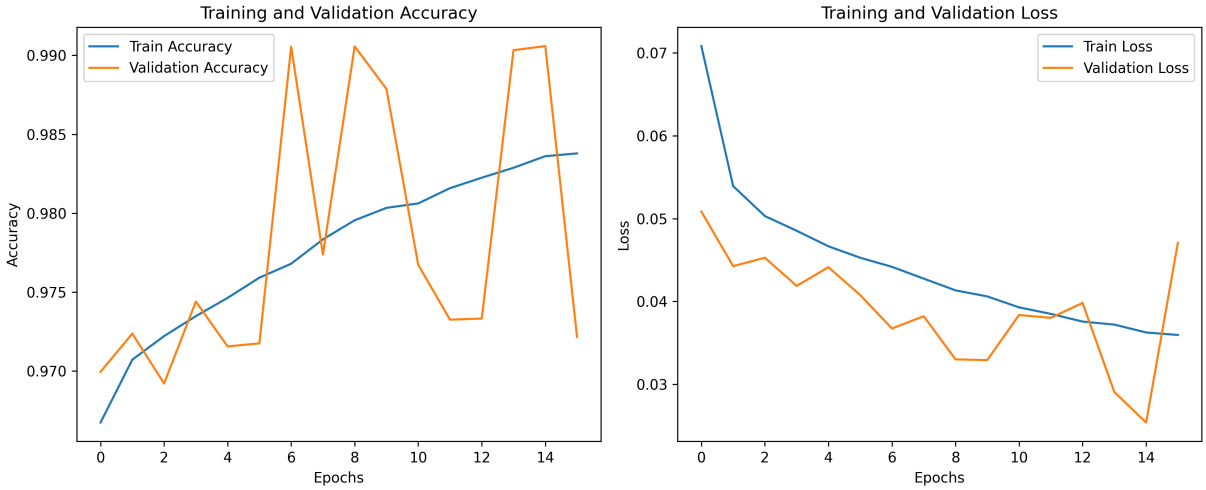


Figure 3.10: Training Curves.

The model was then evaluated on an unseen test set containing 70,778 samples. Performance was exceptionally high, the following table presents the classification rapport, confirmed that both classes — BENIGN and ATTACK — were accurately predicted, with almost no misclassifications as demonstrated in table 3.4.

Table 3.4: Classification Performance Metrics

Metric	**Value**
Accuracy	99.99%
Precision	1.00
Recall	1.00
F1-score	0.98

Figure 3.11 presents the confusion matrix of model predictions, This matrix highlights the model's strength in preserving balance between false positives and false negatives — both are extremely low, a critical factor in cybersecurity applications where either type of error can be costly. Only 668 samples total (502 + 166) were misclassified out of over 70,000, resulting in a false positive rate of 0.71% and a false negative rate of 0.23%, which are negligible in high-throughput network environments.

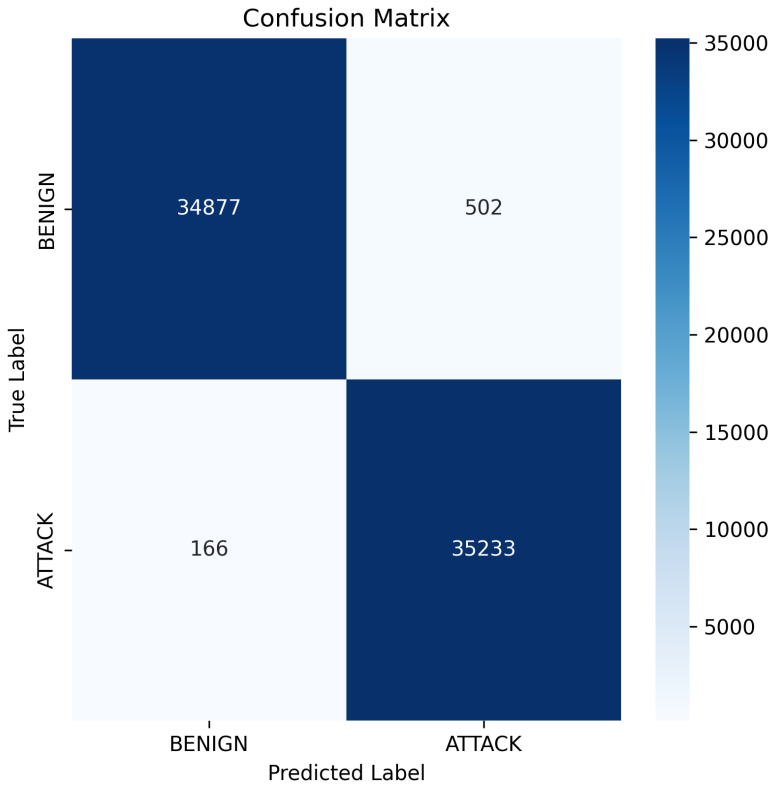


Figure 3.11: Confusion Matrix.

- **Conclusion**
  These evaluation results confirm the suitability of the CNN-LSTM model for real-time DDoS detection in the Hybrid AI-SIEM Framework. Its ability to achieve high recall ensures that attacks are rarely missed, while high precision reduces the risk of false alarms, which can drain analyst resources or trigger unnecessary countermeasures.
  The trained model was saved as cnn-lstm-model.keras, and the scaler was exported via joblib for consistent preprocessing during inference. This allows seamless integration into a real-time detection pipeline.

3.6 Conclusion

This chapter detailed the steps taken to build our hybrid AI-SIEM framework, from designing the architecture to setting up the deployment environment using Docker and Docker Compose. We outlined how the network infrastructure and data collection layer were configured to ensure a smooth and reliable data flow.
We also walked through the development of our AI-based detection system, highlighting the full process from preprocessing the dataset to building the CNN-LSTM model tailored for intrusion detection. Each component was carefully integrated to create a scalable, efficient, and intelligent security solution. With the methodology in place, the next chapter will focus on the implementation phase and present the results of our system in action, evaluating its performance and real-world applicability.

Chapter 4

Implementation and Results

4.1 Introduction

This chapter presents the practical implementation of the Hybrid AI-SIEM Framework and highlights the results obtained through real-world testing. It details how each component, from the Wazuh SIEM platform to the AI detection module was deployed, configured, and integrated to build a unified threat detection system. The results include functional demonstrations of attack detection across multiple vectors, system response behavior, and the performance of the CNN-LSTM model under live conditions. Also we explained in datails the tools integrated and the core security use cases. Each implementation step is followed by validation outputs, including screenshots, detection logs, and alert traces from the Wazuh dashboard.

4.2 Wazuh Implementation

For the deployment setup we used single node deployment which means deploying one Wazuh manager, indexer, and dashboard node , known as Wazuh cluster using docker and docker compose. This process sets up the cluster in isolated Docker containers for streamlined local deployment and testing. The installation and steps are clarified in details in Appendix A. A simplified architecture of Wazuh workflow is presented in figure 4.1 showing the interactions between all components.

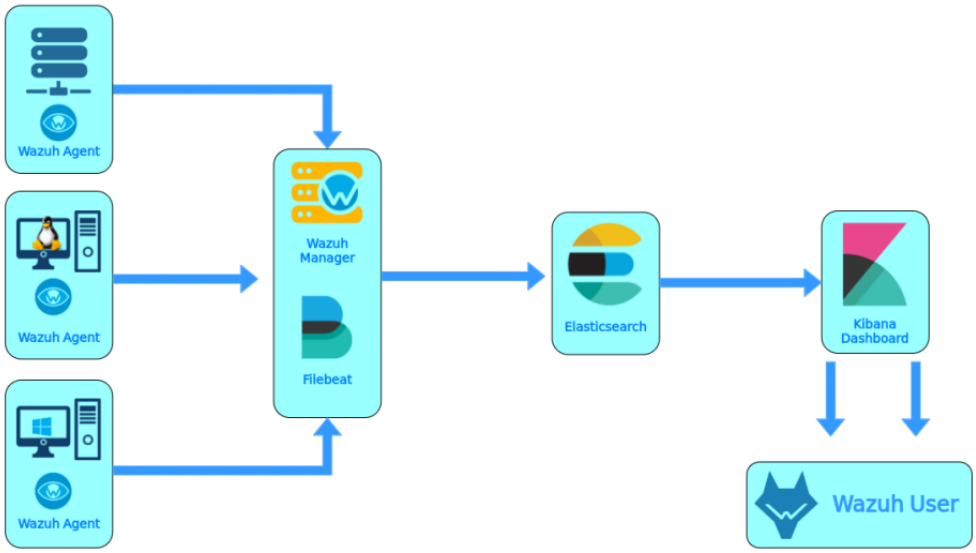


Figure 4.1: Wazuh Workflow

The following Figure 4.2 shows the result of executing docker-compose -d command inside the wazuh single node directory, it illustrates the running status for the three containers confirming that the cluster components were successfully deployed and are operating correctly.

```
hp@HP-EliteBook:~/wazuh-docker/single-node$ docker-compose up -d
WARN[0000] /home/hp/wazuh-docker/single-node/docker-compose.yml: the attribute `version`
 is obsolete, it will be ignored, please remove it to avoid potential confusion
[+] Running 3/3
 ✔ Container single-node-wazuh.indexer-1    Running                    0.0s
 ✔ Container single-node-wazuh.manager-1    Running                    0.0s
 ✔ Container single-node-wazuh.dashboard-1  Running                    0.0s
hp@HP-EliteBook:~/wazuh-docker/single-node$
```

Figure 4.2: Wazuh Cluster Containers Stutus

We access the wazuh dashboard (figures 4.3 and 4.4) via the localhost URL: https://localhost on the administrator host with the credentials: admin:******** as shown on the screenshot below.

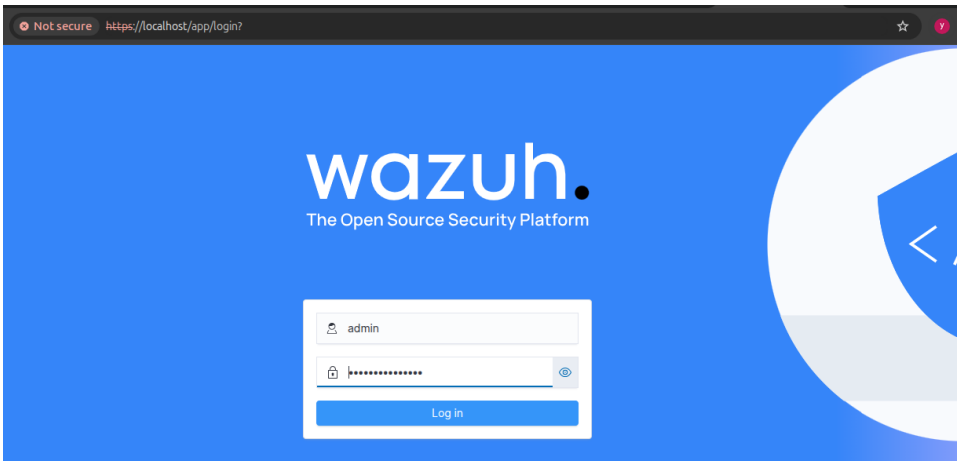


Figure 4.3: Wazuh Dashboard Access

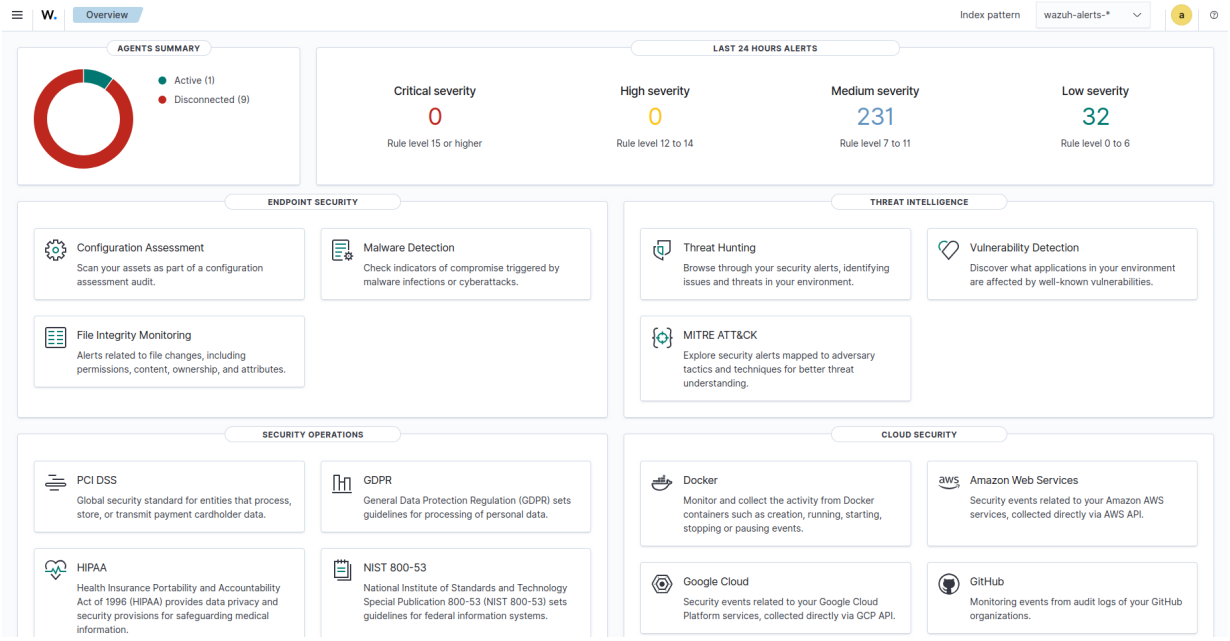


Figure 4.4: Wazuh Dashboard

4.2.1 Docker Compose Configuration

In our project, we used the docker-compose.yml file to deploy a single-node Wazuh platform using the Wazuh three main services: wazuh.manager, wazuh.indexer, and wazuh.dashboard. Each service is running in its own container, based on official Wazuh images.
For each service, we specified the Docker images to use, the required ports to expose, environment variables for internal communication and security, and volumes to store persistent data such as configurations, logs, and certificates. This setup ensures that the system is correctly linked, secure, and ready for monitoring and analysis tasks in a lightweight and portable environment.

The figure 4.5 presents a Snippet of docker-compose.yml wazuh-docker/single-node/docker-compose.yml for the wazuh.manager service

```
services:
  wazuh.manager:
    image: wazuh/wazuh-manager:4.11.0
    hostname: wazuh.manager
    restart: always
    ulimits:
      memlock:
        soft: -1
        hard: -1
      nofile:
        soft: 655360
        hard: 655360
    ports:
      - "1514:1514"
      - "1515:1515"
      - "514:514/udp"
      - "55000:55000"
    environment:
      - INDEXER_URL=https://wazuh.indexer:9200
      - INDEXER_USERNAME=admin
      - INDEXER_PASSWORD=SecretPassword
      - FILEBEAT_SSL_VERIFICATION_MODE=full
      - SSL_CERTIFICATE_AUTHORITIES=/etc/ssl/root-ca.pem
      - SSL_CERTIFICATE=/etc/ssl/filebeat.pem
      - SSL_KEY=/etc/ssl/filebeat.key
      - API_USERNAME=wazuh-wui
      - API_PASSWORD=MyS3cr37P450r.*-
    volumes:
      - wazuh_api_configuration:/var/ossec/api/configuration
      - wazuh_etc:/var/ossec/etc
      - wazuh_logs:/var/ossec/logs
      - wazuh_queue:/var/ossec/queue
```

Figure 4.5: Docker Compose Configuration.

4.2.2 Wazuh Agent Deployment

We deployed the Wazuh agent on all the machines and servers on our infrastructure to be monitored , the deployment steps are detailed in the Appendix A.4. After deployment, to start and connect the agent to our manager on a containerized environment, we have to execute :

```
1 /var/ossec/bin/agent-auth -m <Wazuh-manager-ip> -A <agent-name>
```

Listing 4.1: Command to register the Wazuh agent

Then start the Wazuh agent inside the container with :

```
var/ossec/bin/wazuh-control start
```

Listing 4.2: Command to start Wazuh agent

We check if the deployment was successful through Wazuh dashboard in the same way displayed in the following figure :

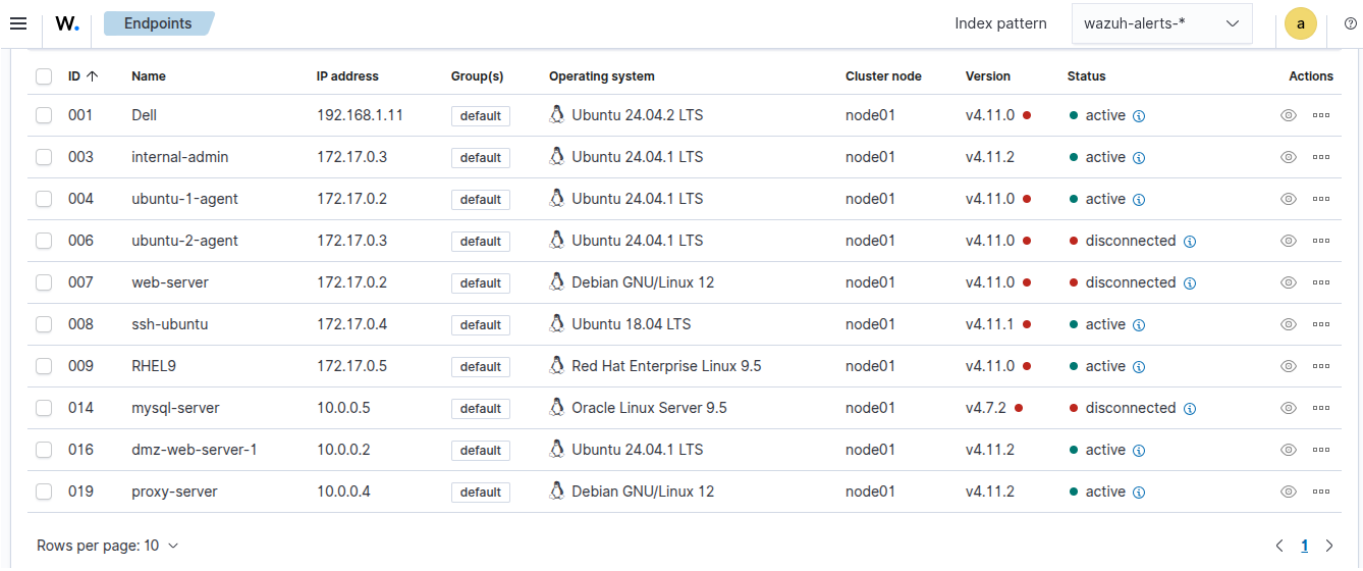


Figure 4.6: Wazuh Agent deployment.

4.3 Integration Layer Implementation

In this section we describe how we implemented the AI model into the Wazuh system. Then, we describe how we tested everything using simulated attacks. We also show how Wazuh works with other tools like Suricata and YARA, and give examples of real security problems it can detect and respond to.

4.3.1 Integration of the AI Model into the Wazuh SIEM Framework

After training and validating the CNN-LSTM model for DDoS detection, the next critical step was to embed its output into the operational workflow of the SIEM system. For this, we designed and implemented two different integration approaches that connect the AI detection system to Wazuh, enabling real-time alerting and automated response.

The figure 4.7 presents our integrations approaches. Both integration methods were tested and validated, offering flexibility for different deployment scenarios. Below, we detail each approach, the components involved, and the rationale behind their use.

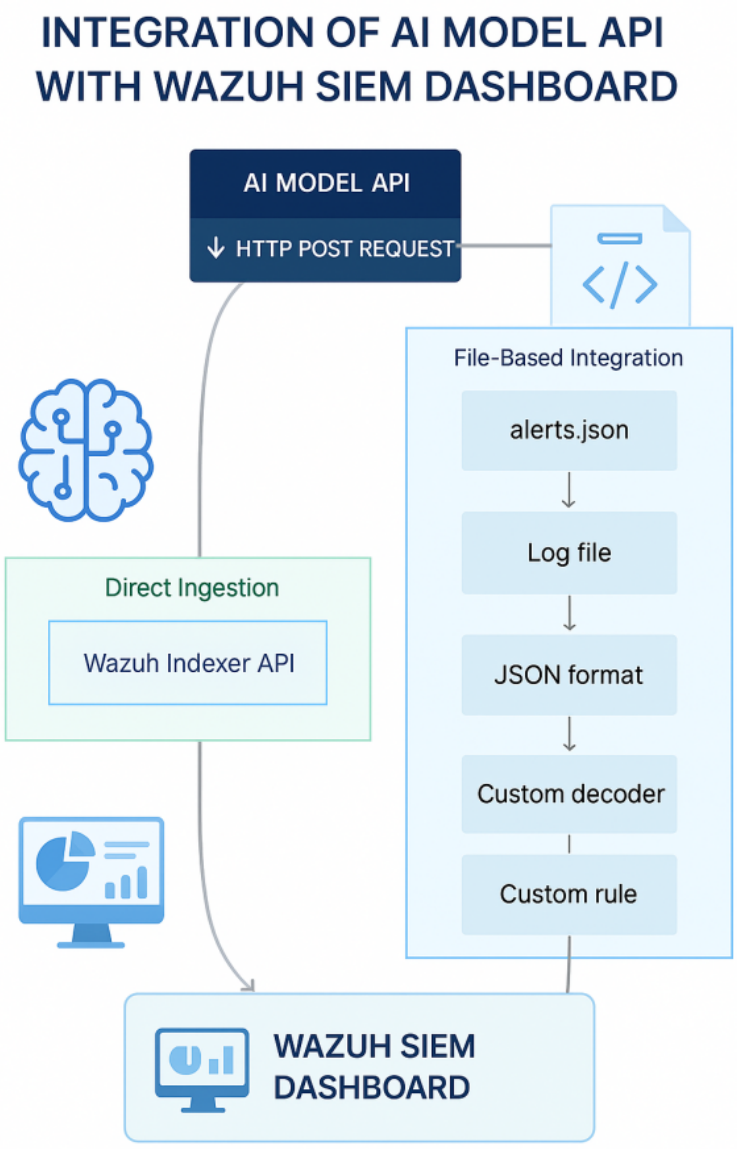


Figure 4.7: AI-Integration Approaches with Wazuh.

4.3.1.1 Deployment of the AI Model as an API Service

To emulate a real-time security monitoring system, the trained CNN-LSTM model was encapsulated in a Python-based RESTful API (using FastAPI). This service exposes an endpoint that accepts network flow data, runs inference, and returns the probability of a DDoS attack.

This architecture allows the model to be queried in near real-time by log collection agents or network monitoring tools and supports horizontal scaling if needed. The following screenshot is a live test of the api service , it shows the handling of prediction requests .

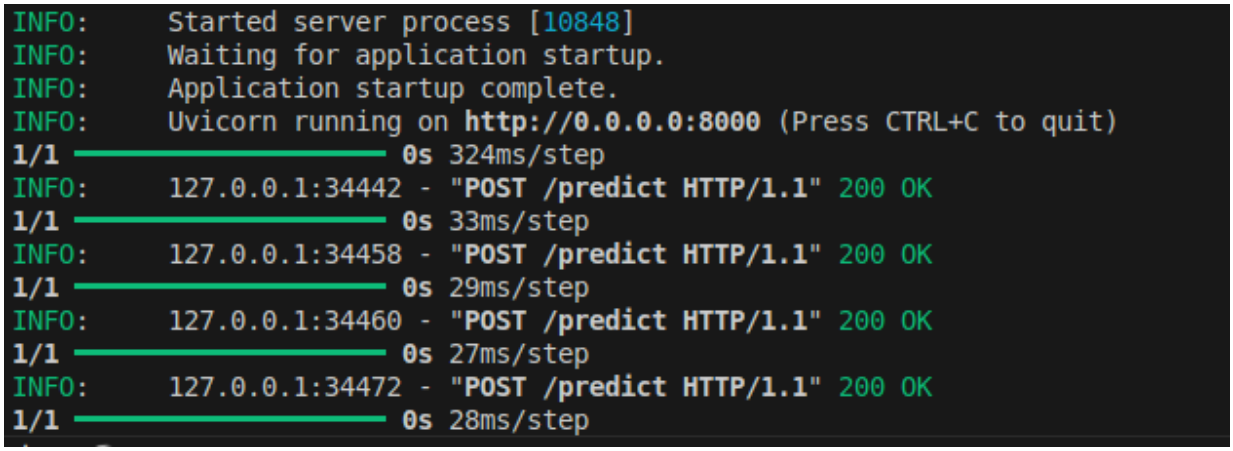


Figure 4.8: Deployment of the AI Model as an API Service.

4.3.1.2 Real-Time Traffic Process and Feature Collection

To validate the detection capabilities of the Hybrid AI-SIEM Framework under realistic conditions, we simulated a network traffic, both benign and attack traffic, and captured it using CICFlowMeter , a Java-based tool developed by the Canadian Institute for Cybersecurity.
We have chosen CICFlowMeter because it is the same tool used to extract features in the CIC-IDS 2017 dataset, which we used for training the CNN-LSTM model. This choice ensures that both the training and testing data follow the same feature schema, eliminating the need for manual reordering or remapping during inference. It also reduces preprocessing overhead and improves model compatibility.
We installed and set up the CICFlowmeter tool on the targeted host. More details will be found on Appendix B.2

- **Proposed solution for real-time data processing :** Real-time data collection wasn't the challenge — the real-time processing of network traffic is what makes this a true real-time threat detection solution. To achieve detection without waiting for the full CSV file to be generated, we developed a custom Python script that continuously monitors the flow output produced by CICFlowMeter. When CICFlowMeter is launched from its directory, it creates a file under /CICFlowMeter/build/distributions/CICFlowMeter-4.0/bin/data/daily/ directory in the format YYYY-MM-DD-Flows.csv, and continuously appends new flow records as network traffic is processed.
  Our script reads this file line-by-line as it is being written, effectively simulating a streaming data pipeline. To ensure that no records are skipped or reprocessed, a lightweight pointer.json file tracks the last processed line. Each new line is:
  - Parsed and transformed into the selected feature vector format,
  - Scaled using the preloaded scaler.joblib used during training,
  - Sent via HTTP POST to the CNN-LSTM FastAPI inference endpoint,
  - Evaluated for DDoS probability in real time.rft

  This flow is illustrated in the diagram provided below :

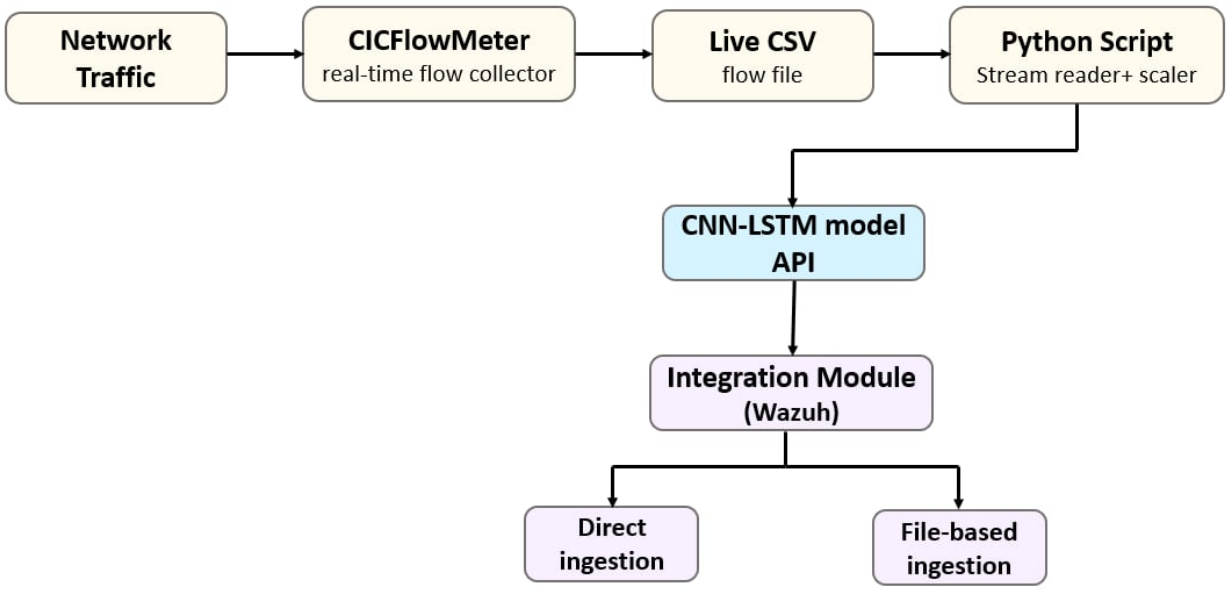


Figure 4.9: Data Real-Time Processing Flow.

If a high-probability attack is detected, the script passes the result to one of the integration modules (either the file-based or direct ingestion method) for alert handling by Wazuh. This process enables continuous, low-latency monitoring without relying on batch-mode processing.
This method bridges the gap between CICFlowMeter's file-based output and the need for real-time AI-driven detection, enabling our framework to function in near-real-time conditions.

4.3.1.3 Method-1 : Direct Ingestion into Wazuh Indexer via HTTP

The approach bypasses file-based communication by sending alerts directly to the Wazuh indexer using an HTTP request. Once the model detects a threat, the python script immediately performs a POST request to Wazuh's ingestion endpoint, submitting the alert in the required format.

- The API model runs continuously and, upon detecting an attack, sends a structured HTTP POST to the Wazuh indexer API.
- The alert is injected directly into Wazuh's index, bypassing the need for intermediate file writing or file monitoring.
- This method makes use of Wazuh's remote log ingestion capabilities via logcollector or syscollector.

4.3.1.4 Method-2 : File-Based Integration

In this approach, when the API detects a DDoS attack, it writes an alert to a file called alerts.json in structured JSON format. This file acts as a communication bridge between the AI layer and Wazuh.

- Wazuh was configured to monitor the alerts.json file as a custom log source. as shown in the figure below

```
<localfile>
   <log_format>json</log_format>
   <location>/home/yasmine/wazuh-docker/single-node/ddos-alerts.json</location>
   <label key="ddos.detector.type">custom_ai</label>
 </localfile>
```

Figure 4.10: Wazuh Configuration to Monitor Json File Alerts.

- A custom rule was written and added to Wazuh's ruleset to detect DDoS alerts and trigger corresponding actions, after a decoder interpreted the structure of the JSON logs of the alert.json file. The rule is shown in the figure below.

```
<group name="ddos,custom_ai,">

  <rule id="100101" level="15">
    <decoded_as>json</decoded_as>
    <field name="event_type">^ddos_detection$</field>
    <description>Custom AI: $(attack_level) DDoS attack detected!
    (Prob:$(probability_percent)%).</description>
    <options>alert_by_email</options>
    <group>ddos_attack,high_probability_attack,</group>
  </rule>
</group>
```

Figure 4.11: Custom Rule to Detect DDoS Attacks.

- Upon detection, the alert is ingested by Wazuh, passed through the rules engine, and displayed on the dashboard.The following figure shows the attack being detected by the api service and write the alert into the file alerts.json

This method is a bit more setup, a little more latency, but keeps all of Wazuh's decoding, correlation, alert grouping and active-response under the Manager's eye , reads, decodes, applies rules, groups events, triggers active-responses, and finally ships to the indexer via its normal pipeline.

4.3.2 Test Environment and Attack Simulation for Both Methods

To evaluate the real-time detection capabilities of the Hybrid AI-SIEM Framework, we created a controlled test environment using Docker containers and virtual hosts. This environment allowed us to simulate both normal user activity and various forms of Distributed Denial-of-Service (DDoS) attacks under realistic conditions.

4.3.2.1 Attacks Simulation

Several types of DDoS attacks were launched to test the robustness of the CNN-LSTM detection model and the responsiveness of the Wazuh integration. These included:

- DDOS Flood Attack using hping3 to overload TCP connection queues:
- HULK Attack which floods the web server with randomized requests.
- SlowHTTPTest to simulate Slowloris-style resource exhaustion.
- UDP and HTTP Floods to test against volumetric and application-layer attacks.

The target machine (172.17.0.2) received high-frequency packets intended to overwhelm TCP queue resources from a kali linux machine that is located on an external network.

CICFlowMeter processed packets into bidirectional flow records. It created a live .csv file containing over 80 statistical flow features, which were continuously updated as traffic flowed through the network. The figure 4.12 below shows CICFlowMeter actively capturing flows during a live Flood DDoS simulation.

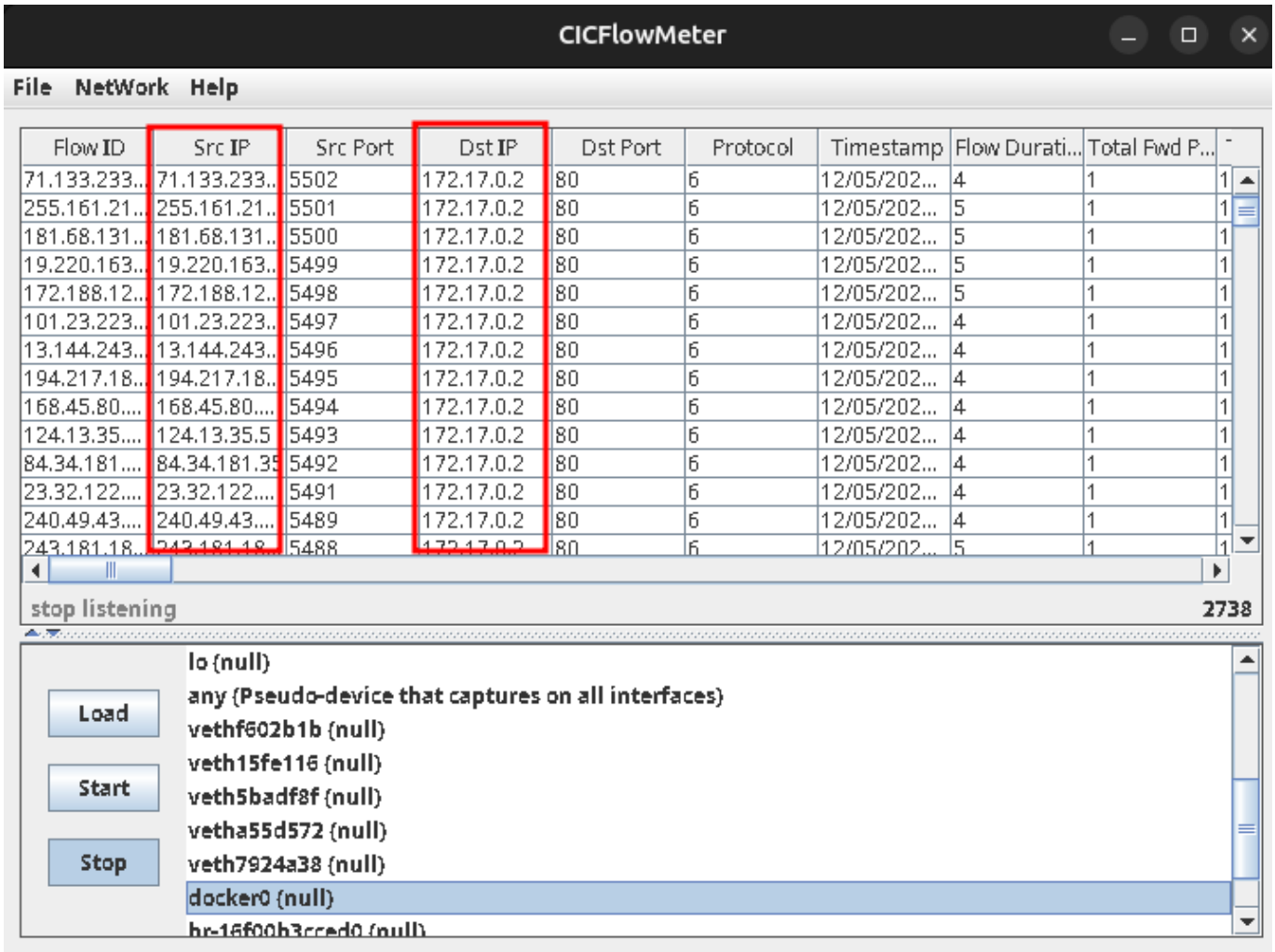


Figure 4.12: Live Capture of CICFlowMeter during DDOS Attacks Simulation

As explained in section 4.3.1.2, a custom Python script monitored the live CSV file in real time, reading each new line as it was written. The detection pipeline included:

- Extracting predefined features from each flow record.
- The scaler object saved from the training phase (scaler.joblib) is loaded and applied.

- Sending feature vectors to the CNN-LSTM inference API.
- Evaluating the returned probability score against set thresholds.
- Forwarding alerts (above threshold 20%) to Wazuh via both:
  - File-based log writing (monitored by Wazuh)
  - Direct ingestion into the Wazuh indexer using HTTP.

This real-time setup allowed us to simulate a production-like detection system capable of generating timely alerts in response to high-risk traffic patterns.

4.3.2.2 Results of the Wazuh-IA Integration

1. **File-Based Integration :**
   The screenshots below shows the line by line data processing and forwarding alerts that are above 0.2 probability as shown in Figure 4.13, Figure 4.14, and Figure 4.15.

```
[Info] Processing line 7186 from '2025-05-12_Flow.csv'...
/venv/lib/python3.12/site-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid featur
e names, but StandardScaler was fitted with feature names
  warnings.warn(
[Info] Predicted DDoS probability: 0.000028
[Info] Probability 0.000 below logging threshold (0.2). No alert.
[Info] Processed 2 new line(s) from 2025-05-12_Flow.csv.
[Info] Total lines processed today (2025-05-12): 7162
[Info] No new lines found in 2025-05-12_Flow.csv. Waiting...
[Info] No new lines found in 2025-05-12_Flow.csv. Waiting...
[Info] No new lines found in 2025-05-12_Flow.csv. Waiting...
[Info] No new lines found in 2025-05-12_Flow.csv. Waiting...
```

Figure 4.13: Data Processing

```
[Info] Processing line 7217 from '2025-05-12_Flow.csv'...
/venv/lib/python3.12/site-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid featur
e names, but StandardScaler was fitted with feature names
  warnings.warn(
[Info] Predicted DDoS probability: 0.454131
[Info] Severity level determined: Low
[Info] Alert written to /home/yasmine/wazuh-docker/single-node/ddos-alerts.json
[Info] Processed 10 new line(s) from 2025-05-12_Flow.csv.
[Info] Total lines processed today (2025-05-12): 7193
[Info] No new lines found in 2025-05-12_Flow.csv. Waiting...
[Info] No new lines found in 2025-05-12_Flow.csv. Waiting...
[Info] No new lines found in 2025-05-12_Flow.csv. Waiting...
```

Figure 4.14: Low DDoS Attack Detected.

```
[Info] Processing line 7187 from '2025-05-12_Flow.csv'...
/venv/lib/python3.12/site-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid featur
e names, but StandardScaler was fitted with feature names
  warnings.warn(
[Info] Predicted DDoS probability: 0.899872
[Info] Severity level determined: High
[Info] Alert written to /home/yasmine/wazuh-docker/single-node/ddos-alerts.json

[Info] Processing line 7188 from '2025-05-12_Flow.csv'...
```

Figure 4.15: High DDoS Attack Detected.

Wazuh displayed the alerts that was triggered after the AI API detected n attack and wrote a structured alert into alerts.json, which Wazuh was monitoring and activated the alert as shown in the bash terminal We visualize the alert in the Wazuh dashboard As illustrated in the figure 4.16 :

Figure 4.16: DDoS Attacks Alerts by The CNN-LSTM Model on The Wazuh Dashboard (File Based Integration Method).

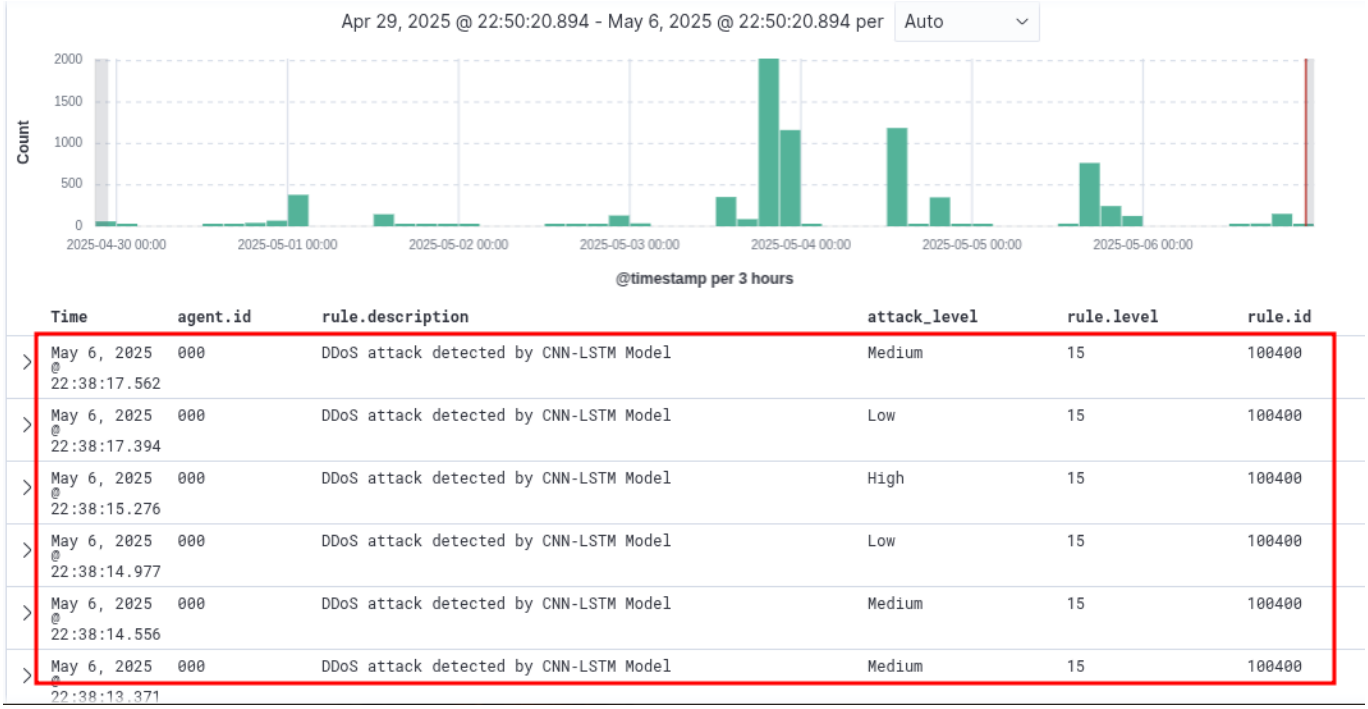
2. **Direct Ingestion to Wazuh Indexer**

   The second screenshots 4.17 shows an alert generated using the direct ingestion method, where the Python script sends the alert to the Wazuh indexer over HTTP and receives the response code from Wazuh indexer , bypassing the log file entirely.

```
[Info] Processing Flow #60...
Flow Context: Proto:6  47.246.7.218:443->192.168.1.11:35174  (Event Time: 03/05/2025 05:52:48 PM)
/venv/lib/python3.12/site-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid
 feature names, but StandardScaler was fitted with feature names
  warnings.warn(
[Info] Flow #60: Predicted DDoS probability: 0.955719
[Info] Flow #60: Severity level determined: High
/venv/lib/python3.12/site-packages/urllib3/connectionpool.py:1064: InsecureRequestWarning: Unverified H
TTPS request is being made to host 'localhost'. Adding certificate verification is strongly advised. Se
e: https://urllib3.readthedocs.io/en/1.26.x/advanced-usage.html#ssl-warnings
  warnings.warn(
[Info] Alert successfully sent to Wazuh indexer. Response: N_isp5YBeyz0M8GWm9By
[Info] Flow #60: Alert written to Wazuh indexer.
```

Figure 4.17: Alert Successfully generated to The Wazuh Indexer.

As illustrated in the figure 4.18, We visualize the alert in the Wazuh dashboard :



Time	agent.id	rule.description	attack_level	rule.level	rule.id
May 6, 2025 @ 22:38:17.562	000	DDoS attack detected by CNN-LSTM Model	Medium	15	100400
May 6, 2025 @ 22:38:17.394	000	DDoS attack detected by CNN-LSTM Model	Low	15	100400
May 6, 2025 @ 22:38:15.276	000	DDoS attack detected by CNN-LSTM Model	High	15	100400
May 6, 2025 @ 22:38:14.977	000	DDoS attack detected by CNN-LSTM Model	Low	15	100400
May 6, 2025 @ 22:38:14.556	000	DDoS attack detected by CNN-LSTM Model	Medium	15	100400
May 6, 2025 @ 22:38:13.371	000	DDoS attack detected by CNN-LSTM Model	Medium	15	100400

Figure 4.18: DDoS Attacks Alerts by The CNN-LSTM Model on The Wazuh Dashboard (Direct Ingestion Method)

4.3.2.3 Conclusion

Both integration methods were successful and demonstrated in our test environment. Each offers trade-offs between simplicity and performance. By implementing both, we show that the framework is adaptable to different operational environments — from small-scale labs to more complex enterprise SIEM pipelines.

4.3.3 Integrations with External Detection Tools

Here, we show how we boosted Wazuh's native strengths by connecting it with specialized tools, making detection more robust and context-rich.

4.3.3.1 Network IDS Integration (Suricata)

Network-based threats often operate below the surface of endpoint logs. To capture these, we integrated Suricata — a powerful network intrusion detection system — with Wazuh. Suricata inspects network traffic and generates rich security insights that Wazuh can analyze the following figure 4.19 describe the suricata integration and alert process.

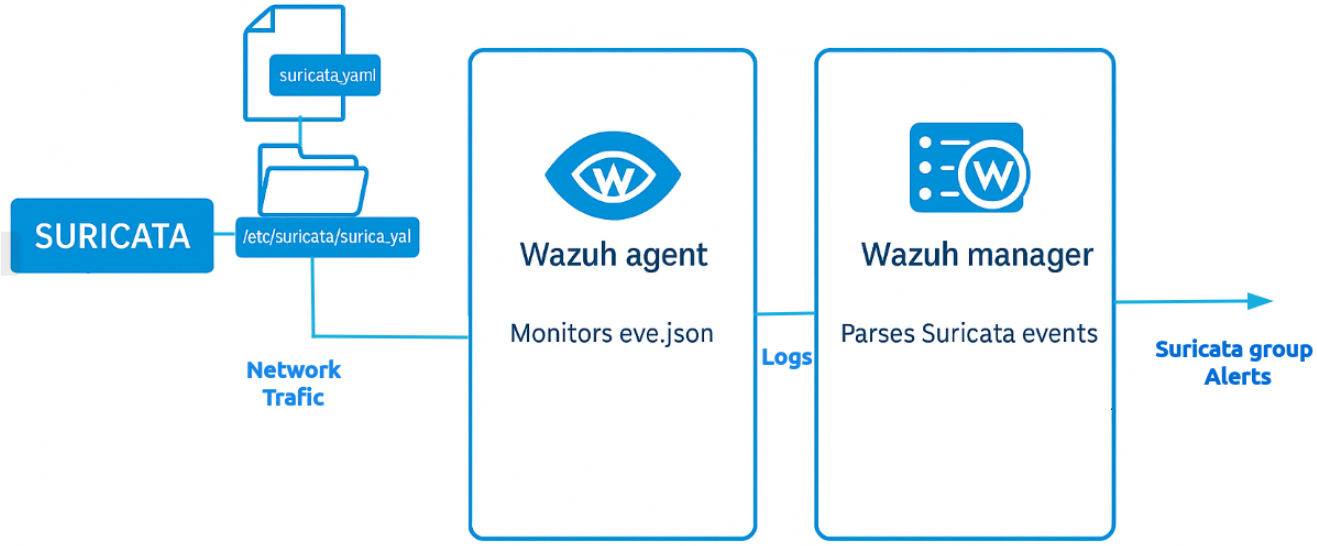


Figure 4.19: Suricata Alert Workflow.

Configuration Steps

We deployed Suricata on an Ubuntu 22.04 endpoint. After installing the Suricata engine and downloading the Emerging Threats ruleset, we configured it to monitor the system's primary network interface and log all findings in JSON format. By adjusting Suricata's configuration (suricata.yaml) as shown in the figure 4.20 , we ensured that both internal (HOME-NET) and external (EXTERNAL-NET) traffic patterns were properly inspected.

```
vars:
  # more specific is better for alert accuracy and performance
  address-groups:
    HOME_NET: "[192.168.0.0/16,10.0.0.0/8,172.16.0.0/12]"
    #HOME_NET: "[192.168.0.0/16]"
    #HOME_NET: "[10.0.0.0/8]"
    #HOME_NET: "[172.16.0.0/12]"
    #HOME_NET: "any"

    EXTERNAL_NET: "!$HOME_NET"
    #EXTERNAL_NET: "any"

    HTTP_SERVERS: "$HOME_NET"
    SMTP_SERVERS: "$HOME_NET"
    SQL_SERVERS: "$HOME_NET"
    DNS_SERVERS: "$HOME_NET"
    TELNET_SERVERS: "$HOME_NET"
    AIM_SERVERS: "$EXTERNAL_NET"
    DC_SERVERS: "$HOME_NET"
    DNP3_SERVER: "$HOME_NET"
    DNP3_CLIENT: "$HOME_NET"
    MODBUS_CLIENT: "$HOME_NET"
    MODBUS_SERVER: "$HOME_NET"
    ENIP_CLIENT: "$HOME_NET"
    ENIP_SERVER: "$HOME_NET"
```

Figure 4.20: Configuration of Suricata.yaml File.

To bring these network insights into Wazuh, we configured the Wazuh agent to read Suricata's JSON log file (/var/log/suricata/eve.json) as illustrated in the following figure 4.21.

```
<localfile>
  <log_format>json</log_format>
  <location>/var/log/suricata/eve.json</location>
</localfile>
```

Figure 4.21: Configuration of Wazuh Agent to Ingest Suricata Logs

Once this pipeline was active, Wazuh could automatically parse Suricata alerts and generate corresponding security events in its dashboard.

Test : We validated the setup by simulating simple network activity (ICMP ping flood) to the monitored endpoint. The wazuh dashboard screenshot illustrated in the figure 4.22 confirms that wazuh successfully picked up the Suricata alerts and made them available for analysis through the Threat Hunting module.

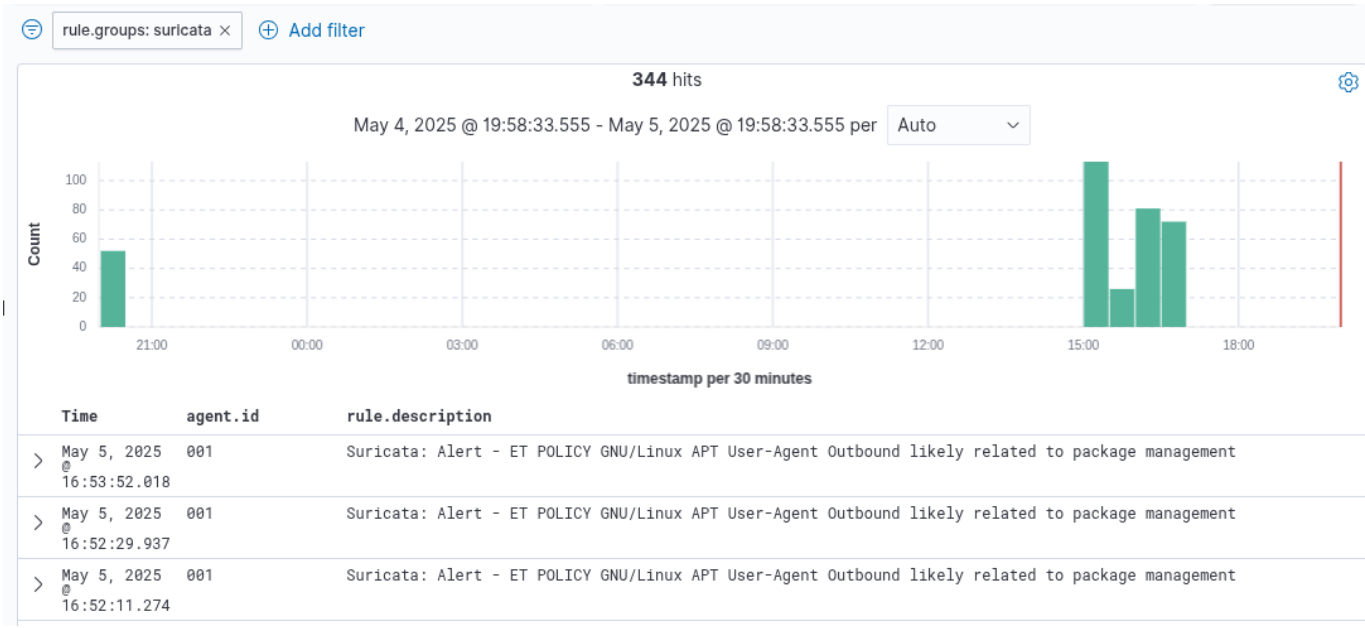


Figure 4.22: Suricata Alerts Through the Threat Hunting module.

Result : Network-based threats like port scans and suspicious traffic were detected and surfaced in Wazuh, expanding our visibility beyond just endpoint activity.

4.3.3.2 Detecting Malware and Suspicious Binaries with YARA Integration

Malware can easily masquerade as ordinary files. To catch these threats, we integrated YARA with Wazuh. YARA scans files for patterns linked to malware, and Wazuh triggers these scans automatically when new or modified files are detected by the File Integrity Monitoring (FIM) module. The figure 4.23 illustrates a diagram that outlines how YARA is integrated with Wazuh modules.

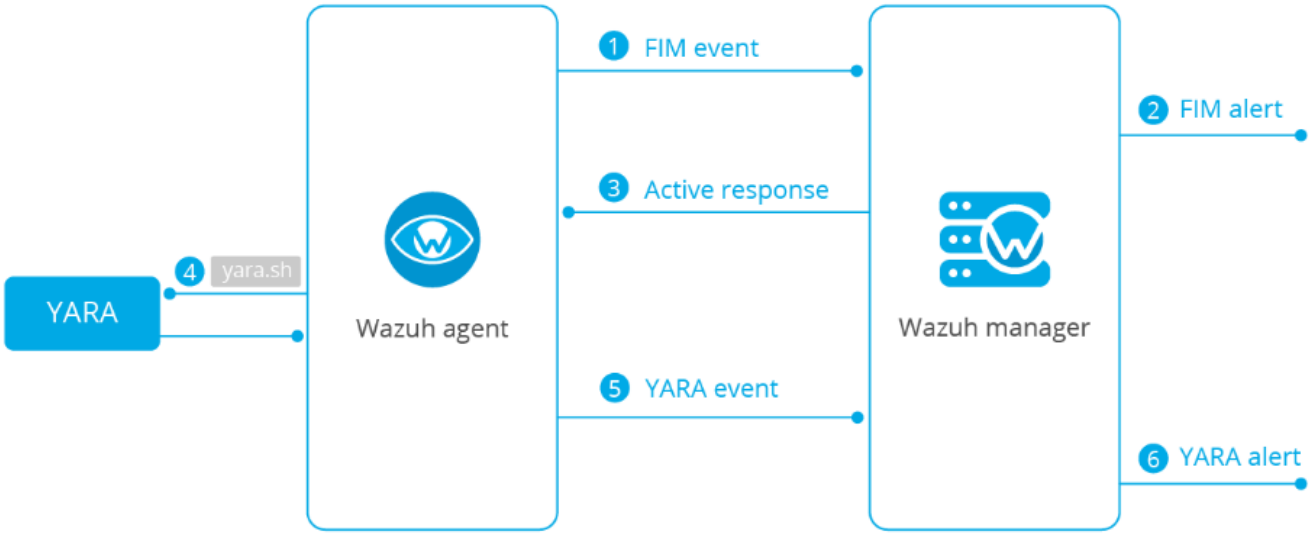


Figure 4.23: Diagram of YARA Integration Stages within Wazuh.

Configuration Steps

- We installed YARA on Ubuntu 22.04 endpoint, loaded a set of malware detection rules from Nextron Systems' Valhalla repository and set the environment and the permissions as shown in the figure 4.24 below :

```
root@dell:/# sudo mkdir -p /tmp/yara/rules
sudo curl 'https://valhalla.nextron-systems.com/api/v1/get' \
-H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8' \
-H 'Accept-Language: en-US,en;q=0.5' \
--compressed \
-H 'Referer: https://valhalla.nextron-systems.com/' \
-H 'Content-Type: application/x-www-form-urlencoded' \
-H 'DNT: 1' -H 'Connection: keep-alive' -H 'Upgrade-Insecure-Requests: 1' \
--data 'demo=demo&apikey=1111111111111111111111111111111111111111111111111111111111111111&format=text'
-o /tmp/yara/rules/yara_rules.yar
```

Figure 4.24: Loading Malware Detection Rules from Nextron Systems Valhalla Repository

- After that we had to write a custom yara.sh script that triggered YARA scans on any new or modified files. The figure 4.25 illustrates a snippet from the yara.sh file .

```
read INPUT_JSON
YARA_PATH=$(echo $INPUT_JSON | jq -r .parameters.extra_args[1])
YARA_RULES=$(echo $INPUT_JSON | jq -r .parameters.extra_args[3])
FILENAME=$(echo $INPUT_JSON | jq -r .parameters.alert.syscheck.path)

# Set LOG_FILE path
LOG_FILE="logs/active-responses.log"

size=0
actual_size=$(stat -c %s ${FILENAME})
while [ ${size} -ne ${actual_size} ]; do
    sleep 1
    size=${actual_size}
    actual_size=$(stat -c %s ${FILENAME})
done

#------------------------ Analyze parameters -------------------------#

if [[ ! $YARA_PATH ]] || [[ ! $YARA_RULES ]]
then
    echo "wazuh-yara: ERROR - Yara active response error. Yara path and rules parameters are mandatory." >> ${LO
    exit 1
fi

#-------------------------- Main workflow --------------------------#

# Execute Yara scan on the specified filename
yara_output="$("${YARA_PATH}"/yara -w -r "$YARA_RULES" "$FILENAME")"

if [[ $yara_output != "" ]]
then
    # Iterate every detected rule and append it to the LOG_FILE
    while read -r line; do
        echo "wazuh-yara: INFO - Scan result: $line" >> ${LOG_FILE}
    done <<< "$yara_output"
```

Figure 4.25: Snippet From The yara.sh File.

Wazuh's Active Response module executed this script whenever the FIM module flagged a change in monitored directories.

- On the Wazuh server side we've added :
  - Rules in local-rules.xml: To detect file changes and YARA scan results, these rules are illustrated in the following figure 4.26.

```
<group name="syscheck,">
  <rule id="100300" level="7">
    <if_sid>550</if_sid>
    <field name="file">/tmp/yara/malware/</field>
    <description>File modified in /tmp/yara/malware/ directory.</description>
  </rule>
  <rule id="100301" level="7">
    <if_sid>554</if_sid>
    <field name="file">/tmp/yara/malware/</field>
    <description>File added to /tmp/yara/malware/ directory.</description>
  </rule>
</group>

<group name="yara,">
  <rule id="108000" level="0">
    <decoded_as>yara_decoder</decoded_as>
    <description>Yara grouping rule</description>
  </rule>
  <rule id="108001" level="12">
    <if_sid>108000</if_sid>
    <match>wazuh-yara: INFO - Scan result: </match>
    <description>File "$(yara_scanned_file)" is a positive match. Yara rule: $(yara_rule)</description>
  </rule>
</group>
```

Figure 4.26: Custom Wazuh Rules for File Changes and YARA Scan Results.

– Decoders in local-decoder.xml: To extract YARA scan results from the active response logs as shown in the following figure.

```
<decoder name="yara_decoder">
  <prematch>wazuh-yara:</prematch>
</decoder>

<decoder name="yara_decoder1">
  <parent>yara_decoder</parent>
  <regex>wazuh-yara: (\S+) - Scan result: (\S+) (\S+)</regex>
  <order>log_type, yara_rule, yara_scanned_file</order>
</decoder>
```

Figure 4.27: Custom Wazuh Decoders for YARA Scan Results Extracting.

– Active Response config in ossec.conf: To ensure the yara.sh script was triggered after FIM events, as depicted in figure 4.28.

```
<command>
    <name>yara_linux</name>
    <executable>yara.sh</executable>
    <extra_args>-yara_path /usr/local/bin -yara_rules /tmp/yara/rules/yara_rules.yar</extra_args>
    <timeout_allowed>no</timeout_allowed>
  </command>

  <active-response>
    <disabled>no</disabled>
    <command>yara_linux</command>
    <location>local</location>
    <rules_id>100300,100301</rules_id>
  </active-response>
```

Figure 4.28: Active Response Configuration for Triggering the yara.sh Script.

Result

As depicted in the figures 4.29 and 4.30 below, when YARA detected malicious files, three different types of malware were detected , Wazuh generated an alert, and the file was automatically deleted to neutralize the threat.

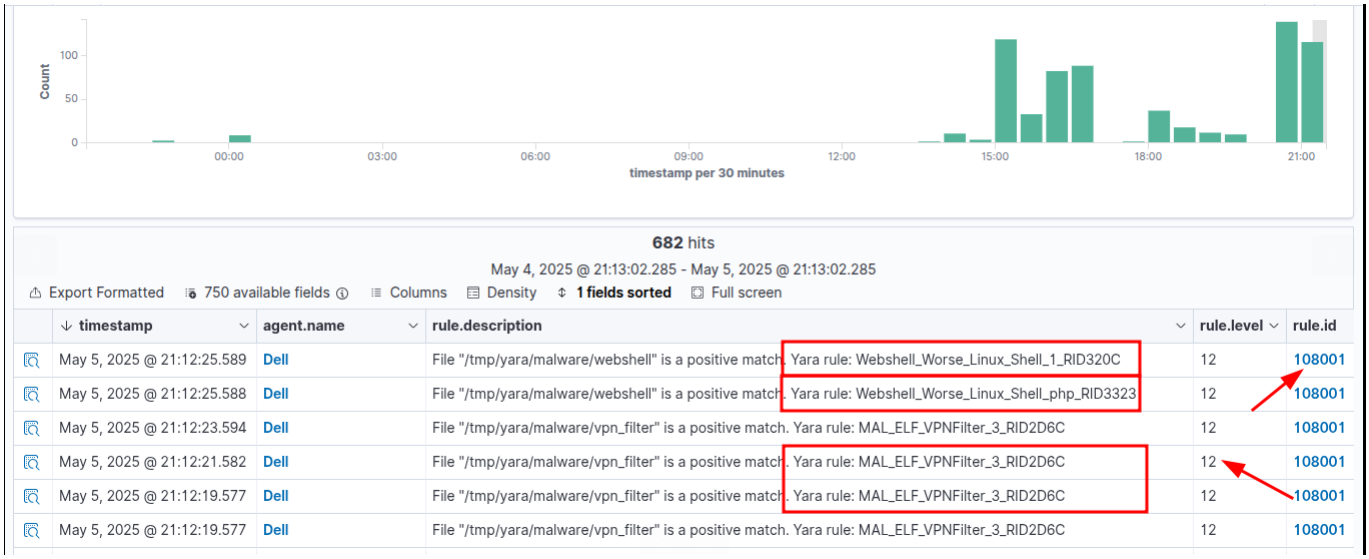


Figure 4.29: YARA Alerts Through the Threat Hunting Module.

Figure 4.30: YARA Alerts Through the Wazuh Discover.

Result : Malware files were detected, deleted, and clearly reported in the Wazuh dashboard.

4.3.3.3 Leveraging LLMs for Alert Enrichment (YARA + ChatGPT API)

Detecting a malicious file is important but understanding its impact matters too. TThat's where large language models (LLMs) come in. We wanted to give our analysts not just an alert but an understandable explanation of what was found and how serious it might be. We integrated large language models (LLMs) into the YARA detection workflow. Specifically, after a YARA detection, our custom script queried ChatGPT to enrich the alert with context about the malware's behavior and impact.

We extended the earlier YARA integration by modifying our yara.sh script as depicted in the figure 4.31. So

that, every time YARA detected something, the script would automatically ask ChatGPT via OpenAI API to explain what the detected malware does and how to handle it, then try to delete the malicious file.

```
#------------------------- Configuration -------------------------#

# ChatGPT API key
API_KEY=[redacted]
OPENAI_MODEL="gpt-3.5-turbo"


# Set LOG_FILE path
LOG_FILE="logs/active-responses.log"

#------------------------- Gather parameters -------------------------#

# Extra arguments
read INPUT_JSON
YARA_PATH=$(echo $INPUT_JSON | jq -r .parameters.extra_args[1])
YARA_RULES=$(echo $INPUT_JSON | jq -r .parameters.extra_args[3])
FILENAME=$(echo $INPUT_JSON | jq -r .parameters.alert.syscheck.path)

size=0
```

Figure 4.31: Yara.sh Script Modifications.

Configuration Steps :

- We created an OpenAI API key and plugged it into the script.
- We used the GPT-3.5-turbo model and fed it clean prompts that included the description from the YARA rule.
- We made sure our YARA rules always had a description — ChatGPT needed that info to explain things clearly.

  On the Wazuh server, we didn't need to change much:

- The custom rules and decoders we built earlier (in 4.1.2) with some adjustments illustrated in the figure 4.32, already knew how to handle the YARA logs.

```
<decoder name="yara_decoder">
  <prematch>wazuh-yara:</prematch>
</decoder>

<decoder name="yara_decoder1">
  <parent>yara_decoder</parent>
  <regex>wazuh-yara: (\S+) - Scan result: (\S+) (\S+)</regex>
  <order>log_type, yara_rule, yara_scanned_file</order>
</decoder>

<decoder name="YARA_child">
  <parent>YARA_decoder</parent>
  <regex type="pcre2">wazuh-YARA: (\S+)</regex>
  <order>YARA.log_type</order>
</decoder>

<decoder name="YARA_child">
  <parent>YARA_decoder</parent>
  <regex type="pcre2">Scan result: (\S+)\s+</regex>
  <order>YARA.rule_name</order>
</decoder>
```

Figure 4.32: Adjustments of The Custom Decoders.

The enriched ChatGPT responses were added to the same log lines, so everything flowed smoothly. The Figure (4.33 ; 4.34) show the API queries and the active responses.

t	data.alert.signature	ET INFO OpenAI API Domain in DNS Lookup (api .openai .com)
t	data.alert.signature_id	2044998
t	data.app_proto	dns
t	data.dest_ip	192.168.1.1
t	data.dest_port	53
t	data.direction	to_server
#	data.dns.query.id	64,760
#	data.dns.query.opcode	0
t	data.dns.query.rrname	api.openai.com
t	data.dns.query.rrtype	AAAA
#	data.dns.query.tx_id	0
t	data.dns.query.type	query

Figure 4.33: Open-AI API Query.

Figure 4.34: Real Time Active Response : Deleting Malware Files.

4.3.4 Core Security Monitoring Use Cases

This section walks through the real-world security scenarios we implemented to test and demonstrate the capabilities of our hybrid AI-SIEM framework. Each example reflects practical threats that organizations commonly face. For every case, we describe the problem, how we configured Wazuh to detect and respond to it, and share the outcome, backed by actual alerts and screenshots from our system.

4.3.4.1 Blocking Malicious Actors and Brute-force Attacks

In this section, we demonstrate two real-world use cases we implemented to detect and block malicious IP addresses on Ubuntu systems. Both cases rely on Wazuh's CDB lists, custom rules, and its Active Response capability to automatically block attacker IPs and reduce risk effectively.

A. Blocking a Known Malicious Actor

Problem : Prevent a known malicious IP address from accessing the Ubuntu endpoint.

Problem Solution :
We configured Wazuh to monitor web server logs and block any IP address that matched an external threat intelligence list. This mimics a real-world situation where an IP flagged in an intelligence feed tries to interact with our systems.

- Configured Wazuh agent on Ubuntu to monitor Apache access logs by the local file bloc illustrated in figure 4.35 to the ossec.conf file :

```
<localfile>
  <log_format>apache</log_format>
  <location>/var/log/apache2/access.log</location>
</localfile>
```

Figure 4.35: Configuration of Wazuh Agent to Monitor Web Server Logs.

- Downloaded the AlienVault IP reputation database and added the attacker's IP address as shown in the following terminal screenshot FIGURE 4.36.

```
root@dell:/home/yasmine# docker exec -it single-node-wazuh.manager-1 bash
bash-5.2# echo 172.17.0.4 >> /var/ossec/etc/lists/alienvault_reputation.ipset
bash-5.2# echo 200.14.10.10 >> /var/ossec/etc/lists/alienvault_reputation.ipset
```

Figure 4.36: Downloading The AlienVault IP Reputation Database.

- Created a custom detection rule in local-rules.xml to trigger an alert for blacklisted IPs as shown in the following figure :

```
<group name="attack,">
  <rule id="100100" level="10">
    <if_group>web|attack|attacks</if_group>
    <list field="srcip" lookup="address_match_key">etc/lists/blacklist-alienvault</list>
    <description>IP address found in AlienVault reputation database.</description>
  </rule>
</group>
```

Figure 4.37: Wazuh Rule Configuration for Blacklisted IPs Alerts.

- Configured Active Response in ossec.conf to block flagged IPs using iptables as depicted in the following figure 4.38.

```
<active-response>
   <disabled>no</disabled>
   <command>firewall-drop</command>
   <location>local</location>
   <rules_id>100100</rules_id>
   <timeout>60</timeout>
 </active-response>
```

Figure 4.38: Active Response Configuration to Block Flagged IPs.

Result

Wazuh detected and blocked the malicious IP based on the reputation list, preventing further access to the server as shown in the following dashboard screenshot(Figure 4.39).

285 hits

May 4, 2025 @ 15:01:59.068 - May 5, 2025 @ 15:01:59.068

Export Formatted | 750 available fields | Columns | Density | 1 fields sorted | Full screen

timestamp	agent.name	rule.description	rule.level	rule.id
May 5, 2025 @ 15:01:25.816	ubuntu-2-agent	Host Blocked by firewall-drop Active Response	3	651
May 5, 2025 @ 15:01:25.779	ubuntu-2-agent	IP address found in AlienVault reputation database.	10	100100
May 5, 2025 @ 15:00:35.768	ubuntu-2-agent	Host Blocked by firewall-drop Active Response	3	651
May 5, 2025 @ 15:00:35.731	ubuntu-2-agent	IP address found in AlienVault reputation database.	10	100100
May 5, 2025 @ 15:00:27.730	ubuntu-2-agent	Host Blocked by firewall-drop Active Response	3	651
May 5, 2025 @ 15:00:25.721	ubuntu-2-agent	IP address found in AlienVault reputation database.	10	100100

Figure 4.39: Alert triggered from blacklisted IP (Rule ID 100100)

A. Blocking a Known Malicious Actor

Problem : Detects and blocks IP addresses attempting SSH brute-force attacks.

Problem Solution :

- Installed Hydra on the attacker machine to launch SSH brute-force attempts, and ensured SSH was running and monitored on the victim machine.
- Leveraged Wazuh's built-in rule for SSH brute-force detection (Rule ID: 5763), and configured Active Response to block detected IPs using firewall-drop as depicted in the figure 4.40.

```
<command>
  <name>firewall-drop</name>
  <executable>firewall-drop</executable>
  <timeout_allowed>yes</timeout_allowed>
</command>

<active-response>
  <disabled>no</disabled>
  <command>firewall-drop</command>
  <location>local</location>
  <rules_id>5763</rules_id>
  <timeout>180</timeout>
</active-response>
```

Figure 4.40: Active Response Configuration to Block Detected IPs.

Test : Hydra launched brute-force SSH attacks against the Ubuntu victim as shown in the following figure.

```
root@65128bee5100:/# hydra -t 4 -l dell -P passwd_list.txt 192.168.1.11 ssh
Hydra v8.6 (c) 2017 by van Hauser/THC - Please do not use in military or secret
service organizations, or for illegal purposes.

Hydra (http://www.thc.org/thc-hydra) starting at 2025-05-05 17:16:02
[DATA] max 4 tasks per 1 server, overall 4 tasks, 10 login tries (l:1/p:10), ~3
tries per task
[DATA] attacking ssh://192.168.1.11:22/
```

Figure 4.41: SSH Brute-Force Attack Simulation on Ubuntu Using Hydra.

After multiple failed login attempts, Wazuh detected the pattern and blocked the attacker's IP for 60 seconds as depicted in the following output.

```
root@65128bee5100:/# ping 192.168.1.11
PING 192.168.1.11 (192.168.1.11) 56(84) bytes of data.


^C
--- 192.168.1.11 ping statistics ---
32 packets transmitted, 0 received, 100% packet loss, time 31782ms
```

Figure 4.42: Attacker Blocked for 60 seconds.

Result

As depicted in figure 4.43 Wazuh detected and blocked the brute-force attacker's IP automatically.

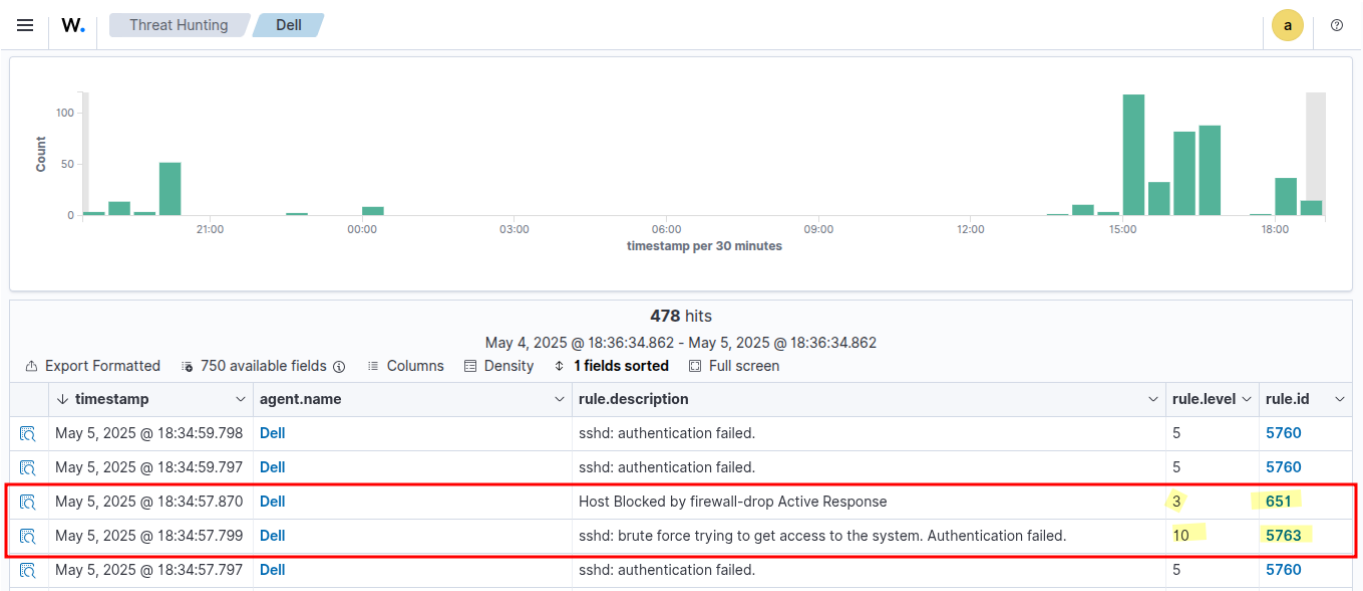


Figure 4.43: Alert for SSH brute-force detection And Active response .

4.3.4.2 Monitoring Docker Events

In this section, we demonstrate how we configured Wazuh to monitor Docker activities on an Ubuntu host running Docker containers. This use case is especially relevant because our entire Hybrid AI-SIEM frame-

work itself is deployed in a Dockerized environment. Monitoring Docker activities in such setups helps detect suspicious or risky container operations, ensuring both the application and the infrastructure remain secure.

Problem : Monitor Docker activity on a host to detect suspicious or risky container operations.

Problem Solution :
On the Docker host machine: Ubuntu 22.04 (Docker installed + Wazuh agent), we configured Wazuh's Docker module to collect and alert on Docker-related events happening on our Ubuntu server hosting Docker containers.
We Configured the Wazuh agent to enable the Docker listener by adding the block illustrated in the figure 4.44 to ossec.conf file :

```
<wodle name="docker-listener">
  <interval>1m</interval>
  <attempts>5</attempts>
  <run_on_start>yes</run_on_start>
  <disabled>no</disabled>
</wodle>
```

Figure 4.44: Enabling Docker listener.

Then we configured the Wazuh agent to send all the runtime logs from docker server to Wazuh manager via syslog module as shown in figure 4.45.

```
<localfile>
  <log_format>syslog</log_format>
  <location>/var/lib/docker/containers/*/*-json.log</location>
</localfile>
```

Figure 4.45: Agent Configuration Via Syslog.

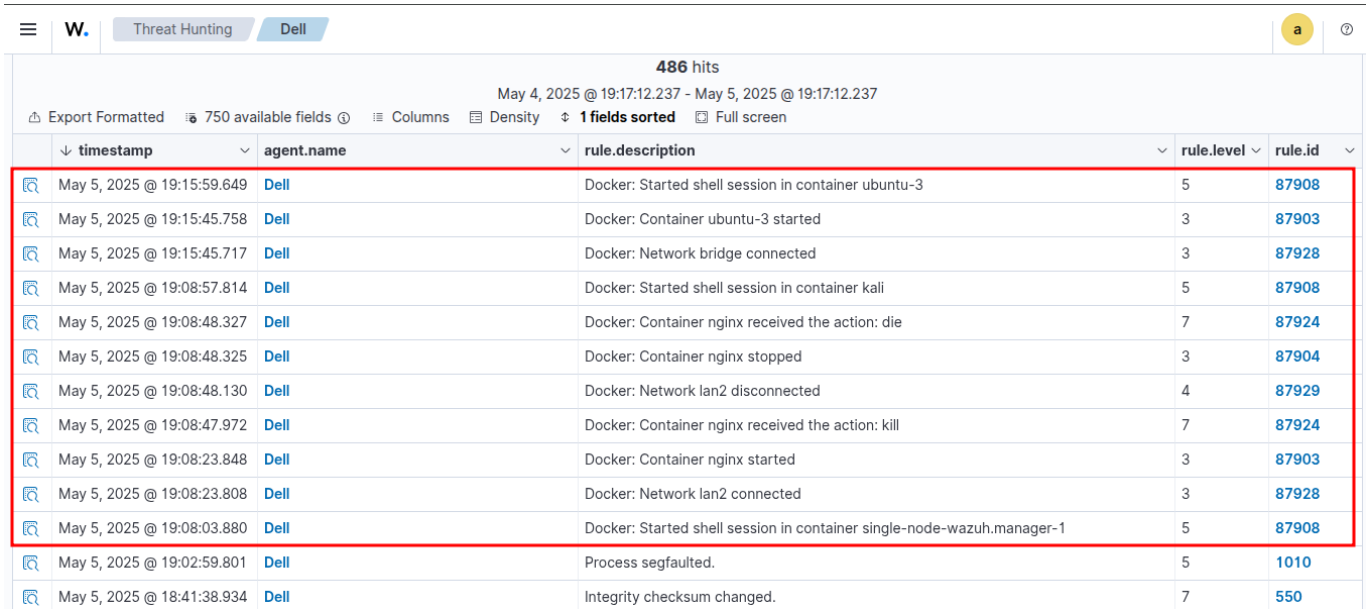
Test : after activating DockelListnet on python virtual environnement We performed several Docker activities on the Ubuntu Docker host to generate detectable events.
These actions in the screenshot below (Figure 4.46)present normal and potentially sensitive container operations, all of which were captured by Wazuh.

Figure 4.46: Docker Listener Capturing Real-Time Docker Events.

Result

Wazuh successfully logged and generated alerts on Docker events like container creation, command execution inside containers, and container removal. This real-time visibility is especially important given that our entire system architecture leverages Docker containers.

On the dashboard we visualize clearly the alerts of several container operations as illustrated in the following screenshot(Figure 4.47).



timestamp	agent.name	rule.description	rule.level	rule.id
May 5, 2025 @ 19:15:59.649	Dell	Docker: Started shell session in container ubuntu-3	5	87908
May 5, 2025 @ 19:15:45.758	Dell	Docker: Container ubuntu-3 started	3	87903
May 5, 2025 @ 19:15:45.717	Dell	Docker: Network bridge connected	3	87928
May 5, 2025 @ 19:08:57.814	Dell	Docker: Started shell session in container kali	5	87908
May 5, 2025 @ 19:08:48.327	Dell	Docker: Container nginx received the action: die	7	87924
May 5, 2025 @ 19:08:48.325	Dell	Docker: Container nginx stopped	3	87904
May 5, 2025 @ 19:08:48.130	Dell	Docker: Network lan2 disconnected	4	87929
May 5, 2025 @ 19:08:47.972	Dell	Docker: Container nginx received the action: kill	7	87924
May 5, 2025 @ 19:08:23.848	Dell	Docker: Container nginx started	3	87903
May 5, 2025 @ 19:08:23.808	Dell	Docker: Network lan2 connected	3	87928
May 5, 2025 @ 19:08:03.880	Dell	Docker: Started shell session in container single-node-wazuh.manager-1	5	87908
May 5, 2025 @ 19:02:59.801	Dell	Process segfaulted.	5	1010
May 5, 2025 @ 18:41:38.934	Dell	Integrity checksum changed.	7	550

Figure 4.47: Alerts Triggered from Monitored Docker Events.

4.3.4.3 Detecting Web Application Attacks (SQL Injection Shellshock)

In this section, we demonstrate how we used Wazuh to detect two classic web application attack techniques — SQL Injection and Shellshock — by monitoring Apache web server logs on our Ubuntu system. Both scenarios reflect real-world attacks that exploit web applications, and we show how Wazuh's built-in detection rules helped us catch them in action.

A. Detecting SQL Injection Attacks

Problem : Spot SQL injection attempts in web requests targeting our Apache server.

Problem Solution

We configured Wazuh to watch for suspicious SQL patterns in Apache logs, such as SELECT and UNION, that are commonly used in injection attacks following these steps :

- Installed Apache on Ubuntu and made sure the web server was accessible
- Configured the Wazuh agent to monitor the Apache access logs in /var/log/apache2/access.log using the Apache log format setting using the block in the following figure.

```
<localfile>
  <log_format>apache</log_format>
  <location>/var/log/apache2/access.log</location>
</localfile>
```

Figure 4.48: Wazuh Agent Configuration of to Monitor Apache access Logs.

Test : From the attacker machine, we simulated an SQL injection attempt by sending this crafted request as illustrated in the following figure.

```
root@dell:/home/yasmine# docker exec -it RHEL9 bash
[root@2f708ac60cc0 /]# curl -XGET "http://172.17.0.2/users/?id=SELECT+*+FROM+users";
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>404 Not Found</title>
</head><body>
<h1>Not Found</h1>
<p>The requested URL was not found on this server.</p>
<hr>
<address>Apache/2.4.58 (Ubuntu) Server at 172.17.0.2 Port 80</address>
</body></html>
[root@2f708ac60cc0 /]#
```

Figure 4.49: SQL injection attempt Simulation.

Result

The two figures 4.50 and 4.51 below confirm that wazuh scanned the logs and raised alerts based on SQL injection detection rule with the rule ID: 31103 for suspicious attempts.

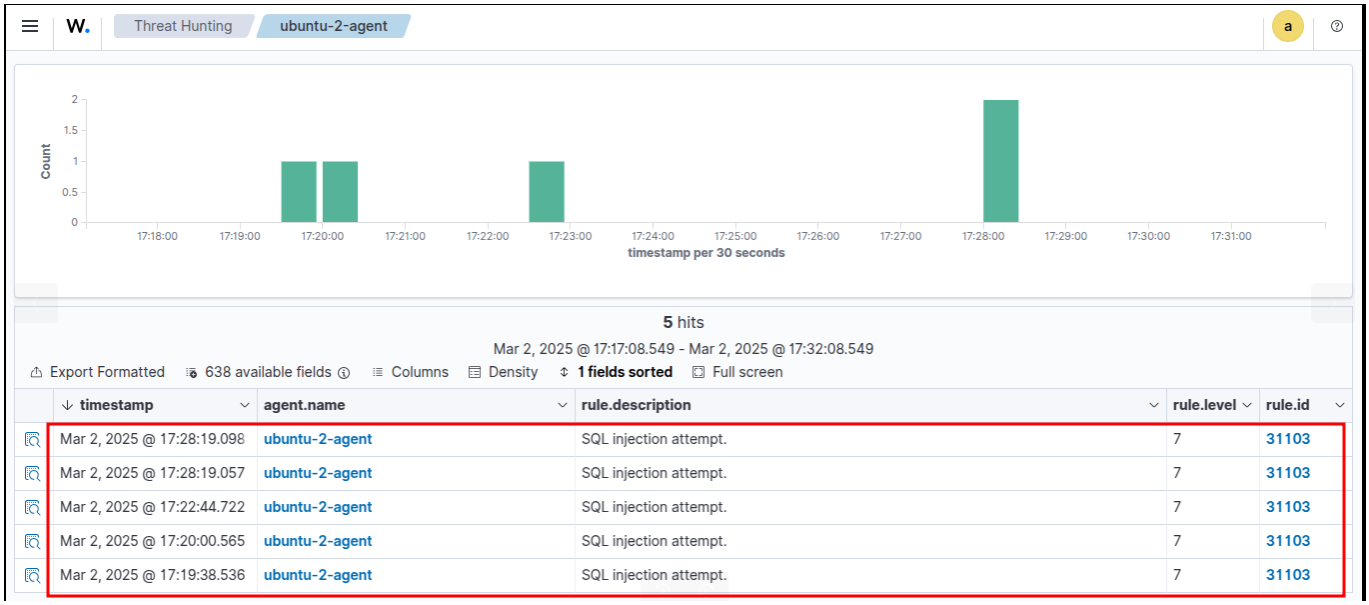


Figure 4.50: Alert triggered by SQL injection attempt Through the Threat Hunting Module.

Figure 4.51: Alert triggered by SQL injection attempt (Rule ID 31103).

B. Detecting Shellshock Attacks

Problem : Catch Shellshock exploitation attempts (CVE-2014-6271) against the Apache server.

Problem Solution :

Apache server was already set up and monitored from the previous SQL injection use case, we reused the same configuration and focused on testing detection of a Shellshock payload.

Test : From the attacker machine, we fired off a simulated Shellshock attack using this command:

```
sudo curl -H "User-Agent: () { :; }; /bin/cat /etc/passwd" http://    <The victim IP>
```

Listing 4.3: Command to simulate Shellshock attack

Result

The two figures 4.52 and 4.53 below confirm that Wazuh detected the Shellshock attempt and immediately alerted us about the exploit attempt targeting the server.

Figure 4.52: Alert Triggered by Shellshock Attack Attempt.

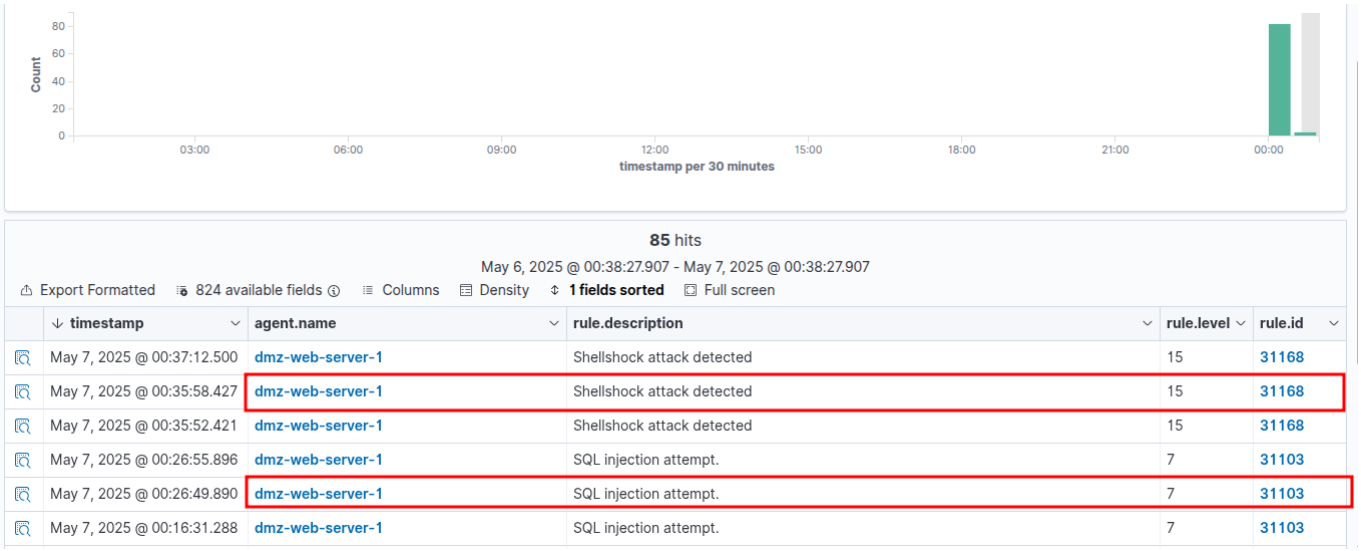


Figure 4.53: Alert Triggered by Shellshock Attack Attempt on Threat Hunting Module.

4.3.4.4 Monitoring Execution of Malicious Commands

In this section, we show how we configured Wazuh to detect when potentially dangerous or sensitive commands are run on our Ubuntu system. This is a critical use case because attackers often execute commands like netcat or nmap for reconnaissance, backdoors, or lateral movement once they've gained a foothold.

Problem : Know immediately when an attacker runs risky shell commands on a monitored system.

Problem Solution

We set up Wazuh to watch for and alert us when certain high-risk commands were executed on our Ubuntu endpoint machine.

- We enabled process monitoring in the Wazuh agent by adding the configuration in the following figure to ossec.conf file.

```
<localfile>
  <log_format>audit</log_format>
  <location>/var/log/audit/audit.log</location>
</localfile>
```

Figure 4.54: Enabling Process Monitoring.

- We created a custom detection rule in local-rules.xml to alert when these commands were run as depicted in the following figure.

```
<group name="audit">
  <rule id="100210" level="12">
      <if_sid>80792</if_sid>
      <list field="audit.command" lookup="match_key_value" check_value="red">etc/lists/suspicious-programs</list>
      <description>Audit: Highly Suspicious Command executed: $(audit.exe)</description>
      <group>audit_command,</group>
  </rule>
</group>
```

Figure 4.55: Custom Rule to Detect Suspicious Command Execution.

- We created a lookup file /var/ossec/etc/lists/suspicious-programs with the contents illustrated in the following figure 4.56.

```
hp@HP-EliteBook: ~
  GNU nano 5.8                          /var/ossec/etc/lists/suspicious-programs
#creating a list of suspecious progs in /var/ossec/etc/lists/suspicious-programs#
ncat:yellow
nc:red
tcpdump:orange
```

Figure 4.56: Lookup File For Suspicious Programs.

- And added the CDB list to the <ruleset> section of the Wazuh server's ossec.conf as depicted in the following figure 4.57.

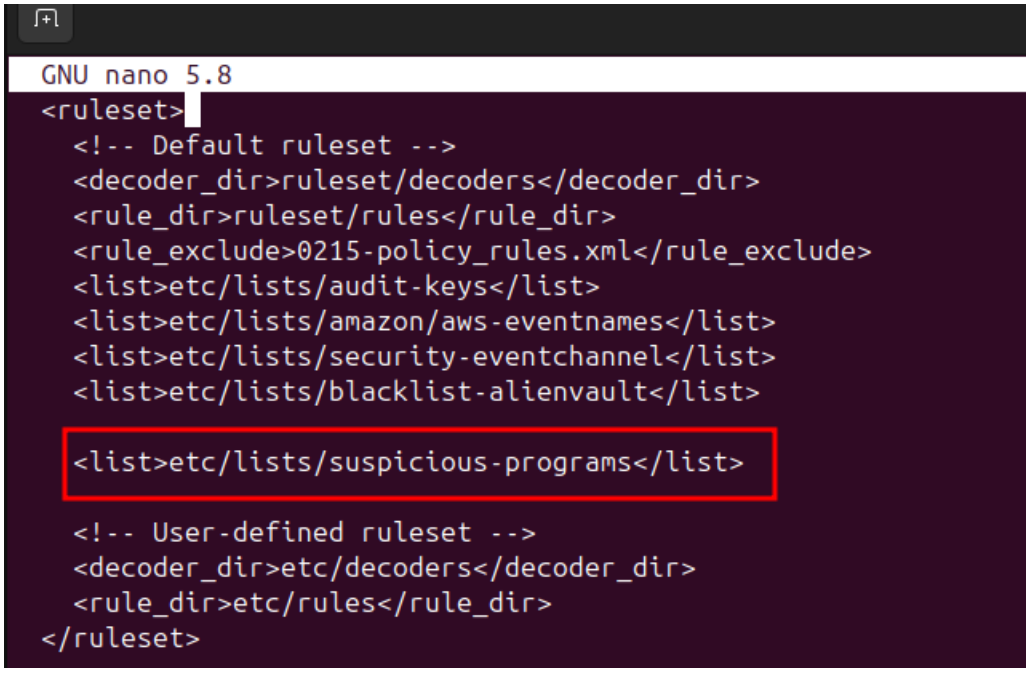
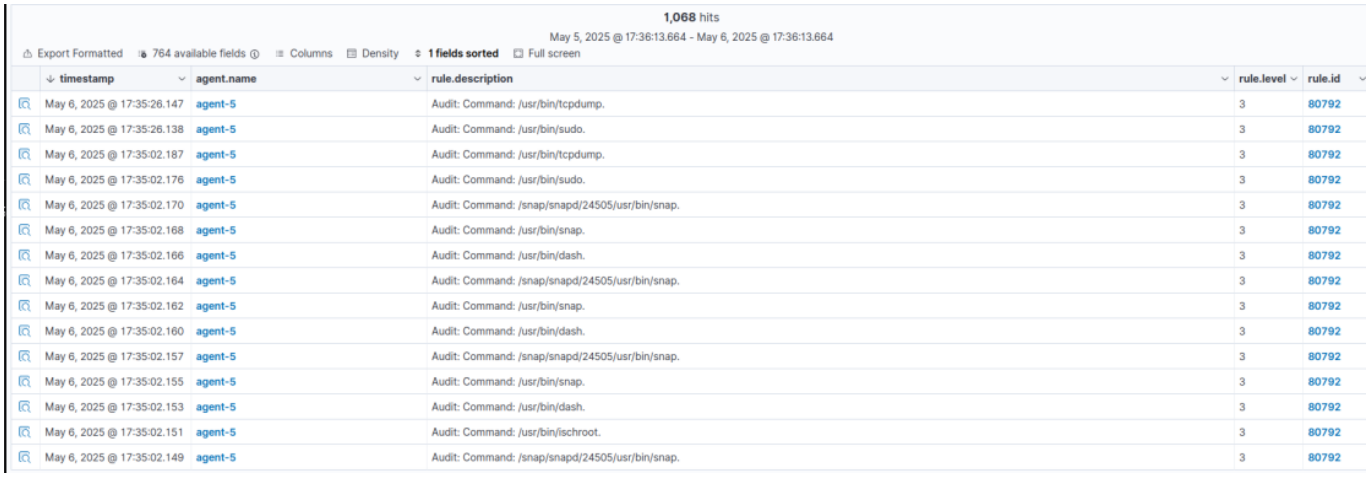


Figure 4.57: Configuration of CDB Lists on the wazuh server's Rulest Section.

Test : We simulated attacker behavior by running the suspicious commands on the monitored Ubuntu machine.

Result

The figure 4.58 confirms that Wazuh successfully flagged these suspicious command executions, showing us exactly what was run and when — which would help us respond quickly in a real attack scenario.



1,068 hits

May 5, 2025 @ 17:36:13.664 - May 6, 2025 @ 17:36:13.664

Export Formatted | 764 available fields | Columns | Density | 1 fields sorted | Full screen

timestamp	agent.name	rule.description	rule.level	rule.id
May 6, 2025 @ 17:35:26.147	agent-5	Audit: Command: /usr/bin/tcpdump.	3	80792
May 6, 2025 @ 17:35:26.138	agent-5	Audit: Command: /usr/bin/sudo.	3	80792
May 6, 2025 @ 17:35:02.187	agent-5	Audit: Command: /usr/bin/tcpdump.	3	80792
May 6, 2025 @ 17:35:02.176	agent-5	Audit: Command: /usr/bin/sudo.	3	80792
May 6, 2025 @ 17:35:02.170	agent-5	Audit: Command: /snap/snapd/24505/usr/bin/snap.	3	80792
May 6, 2025 @ 17:35:02.168	agent-5	Audit: Command: /usr/bin/snap.	3	80792
May 6, 2025 @ 17:35:02.166	agent-5	Audit: Command: /usr/bin/dash.	3	80792
May 6, 2025 @ 17:35:02.164	agent-5	Audit: Command: /snap/snapd/24505/usr/bin/snap.	3	80792
May 6, 2025 @ 17:35:02.162	agent-5	Audit: Command: /usr/bin/snap.	3	80792
May 6, 2025 @ 17:35:02.160	agent-5	Audit: Command: /usr/bin/dash.	3	80792
May 6, 2025 @ 17:35:02.157	agent-5	Audit: Command: /snap/snapd/24505/usr/bin/snap.	3	80792
May 6, 2025 @ 17:35:02.155	agent-5	Audit: Command: /usr/bin/snap.	3	80792
May 6, 2025 @ 17:35:02.153	agent-5	Audit: Command: /usr/bin/dash.	3	80792
May 6, 2025 @ 17:35:02.151	agent-5	Audit: Command: /usr/bin/ischroot.	3	80792
May 6, 2025 @ 17:35:02.149	agent-5	Audit: Command: /snap/snapd/24505/usr/bin/snap.	3	80792

Figure 4.58: Alert Triggered by Execution of Suspicious Command.

4.3.4.5 Disabling a Linux User Account with Active Response

In this use case we demonstrate how Wazuh can automatically respond to by disabling a user account after multiple failed login attempts.

Problem : Repeated failed login attempts on a Linux user account may indicate a brute-force attack.

We want Wazuh to automatically disable any account under attack.

Problem Solution

We configured a custom Wazuh rule to detect multiple failed login attempts on the same user and used the disable-account Active Response command to lock that user's account temporarily.

- On the Wazuh Manager we Added a custom rule in local-rules.xml to detect brute-force attempts. The rule illustrated in the figure 4.59, triggers if there are 3 failed logins within 2 minutes on the same user.

```
<group name="pam,syslog,">
  <rule id="120100" level="10" frequency="3" timeframe="120">
    <if_matched_sid>5503</if_matched_sid>
    <description>Possible password guess on $(dstuser): 3 failed logins in a short period of time</description>
    <mitre>
      <id>T1110</id>
    </mitre>
  </rule>
</group>
```

Figure 4.59: Rule Appears if There Three Failed Logins.

- We Configured the Active Response module as depicted in Figure 4.60 to use that command when the custom rule fires:

```
<ossec_config>
  <active-response>
    <disabled>no</disabled>
    <command>disable-account</command>
    <location>local</location>
    <rules_id>120100</rules_id>
    <timeout>300</timeout>
  </active-response>
<ossec_config>
```

Figure 4.60: Active Response Configuration to Disable Account.

Test

On the endpoint we created two users and logged in as user1, then attempted to switch to user2 three times with an incorrect password. We checked that the account user2 was successfully disabled using the command: sudo passwd –status user2

The L in the output shown below in Figure 4.61 indicates the account is locked.

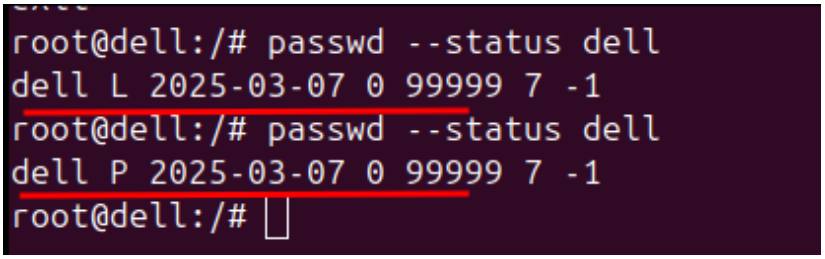


Figure 4.61: Account Locked Output.

Result

As depicted in Figure 4.62, Wazuh successfully detected the brute-force login attempts and automatically disabled the targeted user account for 5 minutes using Active Response. After the timeout period, the account was re-enabled.

Mar 7, 2025 @ 15:07:05.328	Dell	User missed the password to change UID (user id).	5	5301
Mar 7, 2025 @ 15:07:05.287	Dell	Active response: active-response/bin/disable-account - add	3	657
Mar 7, 2025 @ 15:07:03.313	Dell	Possible password guess on dell: 3 failed logins in a short period of time	10	120100
Mar 7, 2025 @ 15:06:59.309	Dell	User missed the password to change UID (user id).	5	5301
Mar 7, 2025 @ 15:06:57.306	Dell	PAM: User login failed.	5	5503
Mar 7, 2025 @ 15:06:51.300	Dell	User missed the password to change UID (user id).	5	5301
Mar 7, 2025 @ 15:06:49.298	Dell	PAM: User login failed.	5	5503

Figure 4.62: Alert Triggered by Rule-ID 120100 with Active Response Disabling User Account.

4.3.4.6 Detecting Suspicious Binaries

Problem : Attackers often try to hide their activity by modifying trusted system binaries. Like, replacing the w or ps commands with a custom script as binaries that can act like normal tools but secretly log activity, open backdoors, or run hidden processes.

Problem Solution

We used Wazuh's Rootcheck module illustrated in figure 4.63 to detect this kind of binary tampering. This module runs periodic scans to look for hidden files,suspicious ports, and known rootkit signatures.

```
<!-- Policy monitoring -->
<rootcheck>
  <disabled>no</disabled>
  <check_files>yes</check_files>
  <check_trojans>yes</check_trojans>
  <check_dev>yes</check_dev>
  <check_sys>yes</check_sys>
  <check_pids>yes</check_pids>
  <check_ports>yes</check_ports>
  <check_if>yes</check_if>

  <!-- Frequency that rootcheck is executed - every 12 hours -->
  <frequency>43200</frequency>

  <rootkit_files>etc/shared/rootkit_files.txt</rootkit_files>
  <rootkit_trojans>etc/shared/rootkit_trojans.txt</rootkit_trojans>
```

Figure 4.63: Rootcheck Block

Test

As illustrated in the figiure 4.64, we simulated a trojan ta the legitimate /usr/bin/w binary and replaced it with a fake shell script that silently created a file and logged a message (acting like malware).

```
root@8799d4422439:/#  cp -p /usr/bin/w /usr/bin/w.copy
root@8799d4422439:/# tee /usr/bin/w << EOF
!/bin/bash
echo "`date` this is evil" > /tmp/trojan_created_file
echo 'test for /usr/bin/w trojaned file' >> /tmp/trojan_created_file
Now running original binary
/usr/bin/w.copy
EOF
```

Figure 4.64: Malware Test Simulation

Result

The rootcheck scanned the system, noticed the tampering, and sent an alert to the Wazuh dashboard as depicted in the following figure 4.65.

	Time	agent.id	rule.description	full_log	rule.level
>	May 5, 2025 @ 20:24:08.421	004	Host-based anomaly detection event (rootcheck).	Trojaned version of file '/bin/w' detected. Signature used: 'uname -a|proc\.h|bash' (Generic).	7
>	May 5, 2025 @ 20:24:08.421	004	Host-based anomaly detection event (rootcheck).	Trojaned version of file '/usr/bin/w' detected. Signature used: 'uname -a|proc\.h|bash' (Generic).	7

Figure 4.65: Rootcheck Alert

4.4 Conclusion

In conclusion, this chapter presented the implementation of our AI-enhanced SIEM framework, combining Wazuh deployment via Docker with a trained CNN-LSTM model for real-time threat detection. We integrated external tools like Suricata and YARA to enrich and contextualize alerts. The core security use cases were successfully addressed. Overall, the results demonstrated the effectiveness, scalability, and reliability of our solution in enhancing security operations within modern infrastructures.

Conclusion

This thesis proposed and implemented a Hybrid AI-SIEM Framework that combines traditional rule-based detection with artificial intelligence–based analysis to improve the detection and response capabilities of modern security monitoring systems. At the heart of the solution is the open-source Wazuh SIEM platform, chosen for its flexibility, modular architecture, and compatibility with external tools. We extended its functionality by integrating a CNN-LSTM deep learning model capable of detecting Distributed Denial-of-Service (DDoS) attacks in near real time.

The deployment of this system involved containerizing key components using Docker and building an AI inference API service with FastAPI. Real-time traffic was collected and transformed into flow-based features using CICFlowMeter, allowing the AI model to operate on live data streams. The model's predictions were fed back into Wazuh through two integration methods: a file-based approach and direct ingestion into the indexer. Both methods were tested successfully.

To enhance detection coverage, we also integrated Suricata for network intrusion detection, YARA for malware scanning, and large language models (LLMs) for enriching alerts with contextual information. The system was evaluated in a simulated environment using a variety of attacks, including SYN flood, HULK, and SlowHTTP DDoS, SQL injection, Shellshock, SSH brute-force, malware execution, and port scanning. Wazuh's active response features were configured to block malicious IPs, deleting malwares and respond automatically to several threat types, contributing to an adaptive and layered defense system.
While the framework successfully combined AI and SIEM technologies, certain limitations remain. The AI model currently focuses only on DDoS detection, and active response for DDoS-level attacks (such as BGP-based blackholing) was not implemented due to infrastructure constraints.

In conclusion, this work demonstrates how combining rule-based SIEM technology with machine learning and external tools creates a more adaptive, intelligent, and responsive cybersecurity framework.

Future Work and Perspectives

Future improvements should focus on extending the AI model beyond DDoS detection. This could be achieved by retraining the existing CNN-LSTM architecture on richer, multi-class datasets that include threats such as data exfiltration, ransomware, and insider activity. Broadening the model's detection scope would increase its practical value in complex, real-world environments.

Another important direction is shifting from passive alerting to active mitigation, particularly for high-impact threats like DDoS. While the current system successfully generates real-time alerts, it does not initiate direct response actions. Blocking thousands of source IPs in a flood scenario is often ineffective and impractical. A more scalable solution would involve configuring Wazuh's Active Response module to trigger BGP blackhole routing, isolating malicious traffic at the network level.. Due to infrastructure and resource limitations, this was not implemented in the current project but remains a strong candidate for future development.

Additionally, improving the overall security posture of the system should involve stronger integration at both the network and host layers. Connecting Wazuh with firewalls, routers, and endpoint controls can provide an extra layer of automated defense—enabling early prevention, faster containment, and more coordinated responses to emerging threats.

Appendix A

Software Deployments

A.1 Wazuh Deployment Steps

OS Requirements

Wazuh Docker deployment requires a Linux-based system with `amd64` architecture and a kernel version of 3.10 or later. We used Ubuntu 22.04 LTS.

Memory Requirements

To run Wazuh properly in Docker, the system should have at least 6 GB of RAM. Also, the memory map count must be at least 262144:

```
sysctl -w vm.max_map_count=262144
```

Listing A.1: Update memory map count

Steps

Step 1: Clone the Wazuh Docker Repository

```
git clone https://github.com/wazuh/wazuh-docker.git -b v4.11.0
cd wazuh-docker/single-node
```

Listing A.2: Clone Wazuh repository

Step 2: Generate SSL Certificates

```
cd config/wazuh_indexer_ssl_certs
docker-compose -f generate-indexer-certs.yml run --rm generator
```

Listing A.3: Generate Wazuh SSL certificates

Step 3: Deploy the Wazuh Stack

```
docker-compose up -d
```

Listing A.4: Deploy Wazuh stack

A.2 Wazuh Agent Installation

The agent can be deployed on a host or container. Use the following commands:

```
wget https://packages.wazuh.com/4.x/apt/pool/main/w/wazuh-agent/wazuh-agent_4.11.0-1
   _amd64.deb
dpkg --install wazuh-agent_4.11.0-1_amd64.deb
```

Listing A.5: Install Wazuh agent

Then edit `/var/ossec/etc/ossec.conf` to point to the Wazuh manager's IP address.

A.3 Java CICFlowMeter Setup

Prerequisites

- Java Development Kit (JDK 8+)
- Maven
- `libpcap-dev` (Linux)

Installation Steps

Step 1: Clone the Repository

```
git clone https://github.com/ahlashkari/CICFlowMeter.git
cd CICFlowMeter
```

Listing A.6: Clone CICFlowMeter

Step 2: Install jnetpcap Locally with Maven

```
cd jnetpcap/linux/jnetpcap-1.4.r1425
mvn install:install-file -Dfile=jnetpcap.jar -DgroupId=org.jnetpcap \
 -DartifactId=jnetpcap -Dversion=1.4.1 -Dpackaging=jar
```

Listing A.7: Install jnetpcap with Maven

Step 3: Build the Executable Package

```
chmod +x gradlew
./gradlew distZip
```

Listing A.8: Build CICFlowMeter

Step 4: Extract the Build

Navigate to `CICFlowMeter/build/distributions/` and extract the zip file.

Step 5: Configure Native Libraries

```
mkdir -p CICFlowMeter-4.0/lib/native/
cp jnetpcap/linux/jnetpcap-1.4.r1425/libjnetpcap*.so CICFlowMeter-4.0/lib/native/
chmod +rx CICFlowMeter-4.0/lib/native/libjnetpcap*.so
```

Listing A.9: Configure native libraries

Step 6: Run CICFlowMeter

```
cd CICFlowMeter-4.0/bin
./CICFlowMeter
```

Listing A.10: Run CICFlowMeter

CICFlowMeter writes live flow data into CSV files, which are then consumed by our detection pipeline (see Section 4.4).

Bibliography

[1] A.Bendovschi Cyber-Attacks – Trends, Patterns and Security Countermeasures, Procedia Economics and Finance,Volume 28,2015,Pages 24-31, ISSN 2212-5671, https://doi.org/10.1016/S2212-5671(15)01077-1.

[2] V. Shanmugam, R. Razavi-Far, and E. Hallaji, "Addressing class imbalance in intrusion detection: A comprehensive evaluation of machine learning approaches," *Electronics*, vol. 14, no. 1, Art. no. 69, 2025, doi: 10.3390/electronics14010069.

[3] P. Vanin, T. Newe, L. L. Dhirani, E. O'Connell, D. O'Shea, B. Lee, and M. Rao, "A study of network intrusion detection systems using artificial intelligence/machine learning," *Applied Sciences*, vol. 12, no. 22, Art. no. 11752, 2022, doi: 10.3390/app122211752.

[4] A. Jamalipour and S. Murali, "A taxonomy of machine-learning-based intrusion detection systems for the Internet of Things: A survey," *IEEE Internet of Things Journal*, vol. 9, pp. 9444–9466, 2021, doi: 10.1109/JIOT.2021.3115621.

[5] C.-M. Ou, "Host-based intrusion detection systems inspired by machine learning of agent-based artificial immune systems," in *Proc. 2019 IEEE Int. Symp. Innovations in Intelligent SysTems and Applications (INISTA)*, Sofia, Bulgaria, 3–5 July 2019, pp. 1–5, doi: 10.1109/INISTA.2019.8778335.

[6] A. Nisioti, A. Mylonas, P. Yoo, and V. Katos, "From intrusion detection to attacker attribution: A comprehensive survey of unsupervised methods," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3369–3388, 2018, doi: 10.1109/COMST.2018.2854721.

[7] E. Kostrecová and H. Bínová, "Security information and event management," Indian Journal of Research, vol. 4, no. 2, pp. 119–120, Feb. 2015.

[8] M. A. B. M. Sulaiman, M. A. Khairuddin, M. R. M. Isa, M. N. Ismail, M. A. M. Shukran, and A. A. B. Sajak, "SIEM network behaviour monitoring framework using deep learning approach for campus network infrastructure," International Journal of Electrical and Computer Engineering Systems, vol. 12, no. SI.2, Nov. 2021, doi: 10.32985/ijeces.12.si.2.

[9] G. González-Granadillo, S. González-Zarzosa, and R. Diaz, "Security information and event management (SIEM): Analysis, trends, and usage in critical infrastructures," Sensors, vol. 21, no. 4759, 2021. Available: https://doi.org/10.3390/s21144759

[10] K. Bezas and F. Filippidou, "Comparative analysis of open-source security information and event management systems (SIEMs)," Indonesian Journal of Computer Science, vol. 12, no. 2, p. 452, 2023. [Online]. Available: https://ijcs.stmikindonesia.ac.id

[11] M. Vielberth and G. Pernul, "A security information and event management pattern," in 12th Latin American Conference on Pattern Languages of Programs (SugarLoafPLoP 2018), Nov. 2018.

[12] M. Sheeraz, M. A. Paracha, M. U. Haque, M. H. Durad, S. M. Mohsin, S. S. Band, and A. Mosavi, "Effective security monitoring using efficient SIEM architecture," Human-centric Computing and Information Sciences, vol. 13, no. 23, May 2023, doi: 10.22967/HCIS.2023.13.023.

[13] M. A. Bouraoua and Y. W. Hallel, Étude et mise en place d'une solution SIEM au sein de la société NAFTAL, Master's thesis, Univ. Saad Dahlab de Blida, Blida, Algeria, 2022.

[14] Kimanzi, R., Kimanga, P., Cherori, D., and Gikunda, P. K. (2024). Deep Learning Algorithms Used in Intrusion Detection Systems - A Review. arXiv preprint arXiv:2402.17020. https://arxiv.org/abs/2402.17020

[15] Salman Muneer,Umer Farooq, Atifa Athar ,Muhammad Ahsan Raza, Taher M. Ghazal ,and Shadman Sakib.A Critical Review of Artificial Intelligence Based Approaches in Intrusion Detection: A Comprehensive Analysis,Journal of EngineeringVolume 2024, Article ID 3909173, 16 pages

[16] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *The Journal of Physiology*, vol. 195, pp. 215–243, 1968.

[17] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional neural networks: An overview and application in radiology," *Insights into Imaging*, vol. 9, pp. 611–629, 2018.

[18] H. Zhang, L. Huang, C. Q. Wu, and Z. Li, "An effective convolutional neural network based on SMOTE and Gaussian mixture model for intrusion detection in imbalanced dataset," *Computer Networks*, vol. 177, p. 107315, 2020.

[19] L. Mohammadpour, T. C. Ling, C. S. Liew, and A. Aryanfar, "A survey of CNN-based network intrusion detection," *Applied Sciences*, vol. 12, no. 16, p. 8162, 2022. Available: https://doi.org/10.3390/app12168162

[20] Laith Alzubaidi, Jinglan Zhang, Amjad J Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, José Santamaría, Mohammed A Fadhel, Muthana Al-Amidie, and Laith Farhan. Review of deep learning: concepts, cnn architectures, challenges, applications, future directions. Journal of Big Data, 8(1):1–74, 2021.

[21] P. Liu, "An intrusion detection system based on convolutional neural network," in Proc. 2019 11th Int. Conf. Comput. Autom. Eng. (ICCAE), Perth, Australia, Feb. 23–25, 2019, pp. 62–67.

[22] Y. Yu, X. Si, C. Hu, and J. Zhang, "A review of recurrent neural networks: LSTM cells and network architectures," Neural Comput., vol. 31, pp. 1235–1270, 2019.

[23] P. Gupta, "Recurrent neural networks in machine learning," *Medium*, Dec. 26, 2023. Available: https://medium.com/@prashantgupta17/recurrent-neural-networks-in-machine-learning-759a943fa759

[24] G. Van Houdt, C. Mosquera, and G. Nápoles, "A review on the Long Short-Term Memory model," *Artif. Intell. Rev.*, vol. 53, no. 1, pp. 1–27, 2020. Available: https://doi.org/10.1007/s10462-020-09838-1

[25] F. M. Shiri, T. Perumal, N. Mustapha, and R. Mohamed, "A comprehensive overview and comparative analysis on deep learning models," Faculty of Computer Science and Information Technology, University Putra Malaysia (UPM), Serdang, Malaysia, 2023.

[26] A. Halbouni, T. S. Gunawan, M. H. Habaebi, M. Halbouni, M. Kartiwi, and R. Ahmad, "CNN-LSTM: Hybrid deep neural network for network intrusion detection system," *IEEE Access*, vol. 10, pp. 100421–100432, Sep. 2022, doi: 10.1109/ACCESS.2022.3206425.

[27] J.-S. Pan, F. Fan, S.-C. Chu, H.-Q. Zhao, and G.-Y. Liu, "A lightweight intelligent intrusion detection model for wireless sensor networks," *Security and Communication Networks*, vol. 2021, Article ID 5540895, 15 pages, 2021, doi: 10.1155/2021/5540895.

[28] Docker Documentation, Docker, Inc. Available: https://docs.docker.com

[29] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, and F. Ahmad, "Network intrusion detection system: A systematic study of machine learning and deep learning approaches," *Transactions on Emerging Telecommunications Technologies*, vol. 32, p. e4150, 2021, 10.1002/ett.4150.

[30] Wazuh, *Wazuh Documentation*, Wazuh, Inc. Available: https://documentation.wazuh.com/.

[31] Z. Chen, M. Simsek, B. Kantarci, P. Djukic *et al.*, "Host-Based Network Intrusion Detection via Feature Flattening and Two-stage Collaborative Classifier," *arXiv preprint arXiv:2306.09451*, Jun. 2023. Available: https://arxiv.org/abs/2306.09451

[32] J. Sani, "Improved log monitoring using host-based intrusion detection system," Adv. Int. J. Multidiscip. Res., vol. 1, no. 1, Jul.–Aug. 2023.