A DISTRIBUTED SECURITY INFORMATION AND EVENT MANAGEMENT (SIEM) SOLUTION USING WAZUH

BY

MR. PURINAT	**PATTANAKEAW**	**6488051**
MR. KITIPHUM	**MUEANGTHONGKHAM**	**6488123**
MRS. PRAKAIKAEW	**RATCHAWONGSA**	**6488194**

ADVISOR
LECT. PAGAPORN PENGSART

A Senior Project Submitted in Partial Fulfillment of the Requirements for

THE DEGREE OF BACHELOR OF SCIENCE (INFORMATION AND COMMUNICATION TECHNOLOGY)

Faculty of Information and Communication Technology
Mahidol University

2024

ACKNOWLEDGEMENTS

We would like to express our sincere gratitude to all those who supported and contributed to the successful completion of this senior project.

Foremost, we extend our deepest appreciation to our advisor, Lect. Pagaporn Pengsart, whose expert guidance, constructive feedback, and unwavering encouragement were invaluable throughout the entire process. Her support played a vital role in shaping both the direction and the quality of our work.

We are also truly thankful to the faculty and staff of the Faculty of Information and Communication Technology, Mahidol University, for providing the necessary resources, facilities, and academic environment that made this project possible.

Mr. Purinat Pattanakeaw
Mr. Kitiphum Mueangthongkham
Mrs. Prakaikaew Ratchawongsa

A Distributed Security Information and Event Management (SIEM) Solution using Wazuh

MR PURINAT PATTANAKEAW 6488051 ITCS/B
MR KITIPHUM MUEANGTHONGKHAM 6488123 ITCS/B
MISS PRAKAIKAEW RATCHAWONGSA 6488194 ITCS/B

B.Sc. (INFORMATION AND COMMUNICATION TECHNOLOGY)

PROJECT ADVISOR: LECT. PAGAPORN PENGSART

ABSTRACT

The Wazuh multi-site implementation creates a decentralized solution to unify security monitoring and management across multiple geographical locations. This project focuses on deploying independent Wazuh SIEM clusters including: Wazuh servers, Wazuh indexers, and Wazuh agents at each geographical location called a site. Each site enables log collection and localized processing while providing centralized visibility through a unified Wazuh dashboard which aggregates security alerts from all sites. This provides a coverage view while allowing site-specific monitoring while reducing network congestion through proximity-based agent-server connections.

The solution addresses the needs of an organization with multiple geographical sites, ensuring independent yet synchronized operations. Local Wazuh servers receive logs from their connected agents before they are indexed with unique patterns per site, and replicated across clusters on all sites to ensure data redundancy. The centralized Wazuh dashboard connects to Wazuh indexers and servers across all the sites and uses Wazuh RESTful API to query data and index patterns.

KEYWORDS : CYBERSECURITY / DISTRIBUTED SYSTEM / SIEM / LOG ANALYSIS / LINUX / INDEXING / API / DASHBOARD / WAZUH
71 P.

โซลูชั่นระบบการจัดการความปลอดภัยแบบกระจายศูนย์ด้วย Wazuh

นาย ภูริณัฐ พัฒนแก้ว 6488051 ITCS/B
นาย กิติภูมิ เหมืองทองคำ 6488123 ITCS/B
นางสาว ประกายแก้ว ราชวงษา 6488194 ITCS/B

วิทยาศาสตรบัณฑิต (เทคโนโลยีสารสนเทศและการสื่อสาร)

ที่ปรึกษาโครงการ: อาจารย์ ผกาพร เพ็งศาสตร์

บทคัดย่อ

การใช้งาน Wazuh แบบหลายไซต์ (Multi-site Implementation) เป็นการออกแบบระบบที่ไม่รวมศูนย์ (Decentralized) เพื่อรวมการตรวจสอบและบริหารจัดการด้านความมั่นคงปลอดภัยจากหลายพื้นที่ทางภูมิศาสตร์เข้าไว้ด้วยกัน โครงการนี้มุ่งเน้นที่การติดตั้งระบบ Wazuh SIEM แต่ละคลัสเตอร์ให้สามารถทำงานได้อย่างเป็นอิสระ โดยแต่ละไซต์จะประกอบด้วย: Wazuh Server, Wazuh Indexer และWazuh Agentเพื่อให้สามารถเก็บบันทึกข้อมูล (Logs) และประมวลผลเบื้องต้นภายในพื้นที่นั้น พร้อมทั้งแสดงผลภาพรวมแบบรวมศูนย์ผ่าน Wazuh Dashboard ที่ทำหน้าที่รวบรวมการแจ้งเตือนด้านความปลอดภัยจากทุกไซต์เข้าด้วยกัน ซึ่งช่วยให้สามารถมองเห็นภาพรวมของการรักษาความปลอดภัยในทุกพื้นที่ ขณะเดียวกันก็ยังรองรับการตรวจสอบเฉพาะไซต์ และลดปัญหาความแออัดของเครือข่ายด้วยการเชื่อมต่อระหว่าง Agent และ Server ที่อยู่ในพื้นที่เดียวกัน

โซลูชันนี้ถูกออกแบบมาเพื่อตอบโจทย์องค์กรที่มีหลายไซต์ทางภูมิศาสตร์ โดยให้แต่ละไซต์สามารถทำงานได้อย่างอิสระแต่มีการประสานข้อมูลร่วมกัน Wazuh Server ในแต่ละไซต์จะรับข้อมูลจาก Agent ที่เชื่อมต่ออยู่ จากนั้นจึงดำเนินการจัดทำดัชนี (Indexing) โดยใช้รูปแบบเฉพาะของแต่ละไซต์ และมีการสำรองข้อมูลไปยังคลัสเตอร์ในทุกไซต์เพื่อให้มั่นใจว่ามีการสำรองข้อมูลอย่างเพียงพอ ทั้งนี้ Wazuh Dashboard แบบรวมศูนย์จะเชื่อมต่อกับ Wazuh

Indexer และ Wazuh Server ในทุกไซต์ และใช้ Wazuh RESTful API ในการดึงข้อมูลและแบบแผนของดัชนี (Index Patterns) เพื่อการวิเคราะห์และแสดงผลแบบรวมศูนย์

คำสำคัญ: CYBERSECURITY / DISTRIBUTED SYSTEM / SIEM / LOG ANALYSIS / LINUX / INDEXING / API / DASHBOARD / WAZUH
71 P.

CONTENTS

CHAPTER 1
INTRODUCTION

This chapter provides an overview of the research project, including its objectives and expected benefits.

1.1 Problem Statement

Security Information and Event Management (SIEM) systems aggregate and analyze security logs and events to promptly detect and respond to cyber threats, ensuring regulatory compliance and effective security oversight. Traditional centralized SIEM solutions, however, can become inefficient for organizations operating across multiple geographical locations, leading to high resource usage, network congestion, and management complexity. This project addresses these issues by deploying localized Wazuh SIEM clusters at each site, enabling efficient, site-specific threat detection and analysis, while maintaining centralized visibility through a unified dashboard for comprehensive security monitoring.

1.2 Objectives of the Project

- Design a distributed SIEM solution using Wazuh.
- Implement secure, scalable, and independent monitoring across multiple sites.

1.3 Scope of the Project

Scope of the project focuses on developing and validating a virtual with multi-site Wazuh SIEM environment. The boundaries and limitations are defined as follows:

1. **Geographical Simulation**
    - The “multi-site” setup is simulated using virtual machines and subnetting within a single physical host. Each site is logically isolated, although not physically separated.

2. **Component Coverage**
    - The system includes core Wazuh components: server, indexer, agent, and dashboard. It also incorporates a custom router and load balancer. All components run on Linux-based systems, except for agents, which may operate on Ubuntu.
3. **Use Case Demonstration**
    - The implementation showcases essential use cases, including log collection, agent enrollment, file integrity monitoring, and alert detection (e.g., brute-force attempts). Advanced threat modeling and real-time attack correlation are beyond the scope of this project.
4. **Intended Users**
    - The solution is designed for cybersecurity teams, IT administrators, and analysts in medium to large organizations with distributed infrastructures.
5. **Automation Dependency**
    - To make automation, depending on the project we will use tools such as Ansible and Jinja2.

1.4 Expected Benefits

- A centralized Wazuh dashboard aggregates security information from all sites and provides an overview of each site's data.
- The decentralized architecture enables faster recovery from security incidents, improving overall system resilience.
- Network congestion is reduced by allowing Wazuh agents to connect to local servers instead of a centralized server.
- Site-specific alert patterns lead to quicker and more accurate responses.
- The use of open-source tools such as Wazuh, Ubuntu, and VirtualBox results in significant cost savings.
- System performance and scalability are improved, allowing seamless addition of

new nodes and sites to meet growing security needs.

- A simplified setup script automates the installation, configuration, and connection of all components.
- A centralized template for Wazuh configuration files enables easy editing and deployment across all sites.

CHAPTER 2
BACKGROUND

This chapter introduces SIEM, Wazuh, and development tools like Ansible and Jinja2 used in this project.

2.1 Security Information and Event Management (SIEM)

Security Information and Event Management (SIEM) is a comprehensive cybersecurity solution designed to provide organizations with real-time visibility into their IT environments. By aggregating and analyzing data from various sources—such as firewalls, servers, applications, and cloud services—SIEM systems enable the detection, analysis, and response to security threats.

At its core, SIEM combines two key functionalities: Security Information Management (SIM), which focuses on the collection and long-term storage of log data, and Security Event Management (SEM), which emphasizes real-time monitoring and correlation of events. This integration allows SIEM systems to normalize and analyze vast amounts of data, identifying patterns and anomalies that may indicate potential security incidents. For instance, unusual login attempts or unexpected data transfers can be flagged for further investigation.

2.2 Wazuh SIEM

Wazuh is an open-source SIEM that offers centralized threat detection, compliance enforcement, and incident response for organizations. Its system architecture consists of agents and a central component, both of which dynamically handle threat detection and response functions. The key components include Wazuh Indexer, Wazuh Server, Wazuh Dashboard, and Wazuh Agents.

2.2.1 Wazuh Indexer

The Wazuh indexer is a real-time, full-text search and analytics engine for

security data. Logs ingested by the Wazuh server are analyzed and sent to the indexer for storage and future querying via the Wazuh dashboard. The indexer stores data in JSON format. Each document includes a set of keys, field names, or attributes, along with their corresponding values. These values may be characters, numbers, booleans, dates, arrays, or other data types.

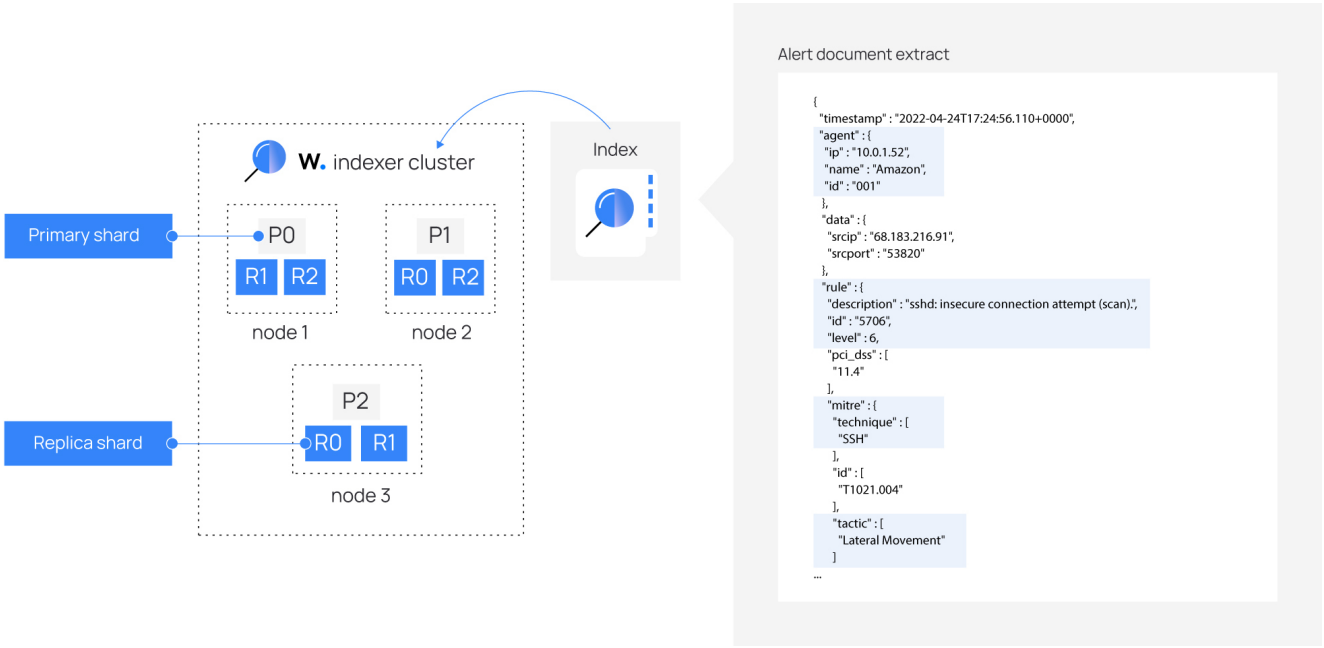

Figure 2.1 Architecture of Wazuh Indexer

An index is a collection of documents that are logically related. In Wazuh, these documents are distributed across containers called shards. By spreading shards across multiple nodes, the Wazuh indexer ensures redundancy, which enhances fault tolerance and increases query capacity as more nodes join the cluster.

2.2.2 Wazuh Server

The Wazuh server component analyzes data received from agents and triggers alerts when threats or anomalies are detected. It also enables administrators to manage agent configurations remotely and monitor their status.

To enhance detection, the Wazuh server integrates threat intelligence sources and enriches alert data using the MITRE ATT&CK framework. It also supports compliance with standards such as PCI DSS, GDPR, HIPAA, CIS, and NIST 800-53, providing context that strengthens security analytics.

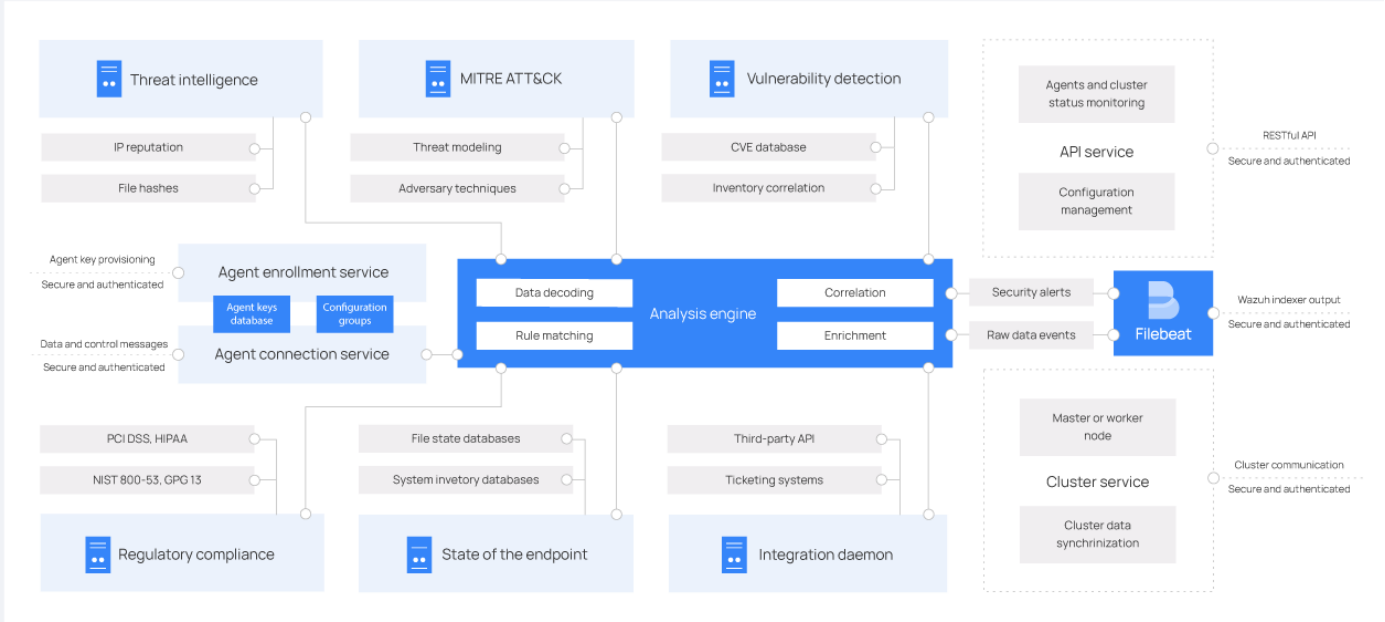

Figure 2.2 Architecture of Wazuh Server

2.2.2.1 Agent Enrollment Service

This service is responsible for enrolling new agents. It generates and distributes unique authentication keys to each agent. The enrollment process runs as a network service and supports authentication using TLS/SSL certificates or a fixed password.

2.2.2.2 Agent Connection Service

The agent connection service receives data from agents and uses the authentication keys from the enrollment service to validate agent identities. It also encrypts communications between each Wazuh agent and the Wazuh server. In addition, it provides centralized configuration management, allowing administrators to push new settings to

agents remotely.

2.2.2.3 Analysis Engine

The analysis engine is the core server component responsible for processing incoming data. It uses decoders to identify the type of logs being analyzed—such as Windows events, SSH logs, or web server logs. These decoders extract relevant fields from each log message, including source IP address, event ID, and username. The extracted data is then evaluated using predefined rules to detect suspicious patterns or security incidents.

2.2.2.4 Wazuh RESTful API

The Wazuh RESTful API provides an interface to interact with the Wazuh infrastructure. It allows users to manage configuration settings for agents and servers, monitor system health, edit Wazuh decoders and rules, and query the state of monitored endpoints. The Wazuh dashboard also relies on this API for functionality.

2.2.2.5 Wazuh cluster daemon

This service enables horizontal scaling of Wazuh servers by deploying them as a synchronized cluster. When used with a network load balancer, this setup supports high availability and balanced workload distribution. The cluster daemon ensures that Wazuh servers remain in sync and can communicate effectively.

2.2.2.6 Filebeat

Filebeat is responsible for sending alerts and events to the Wazuh indexer. It reads the output from the Wazuh analysis engine and forwards events in real time. When operating within a multi-node indexer cluster, Filebeat also supports load balancing.

2.2.3 Wazuh Dashboard

The Wazuh dashboard is a user-friendly web interface for monitoring, analyzing, and visualizing security events and alert data. It is also used for centralized management of the Wazuh platform. The dashboard supports features such as role-based access control

(RBAC) and single sign-on (SSO), enhancing both security and usability.

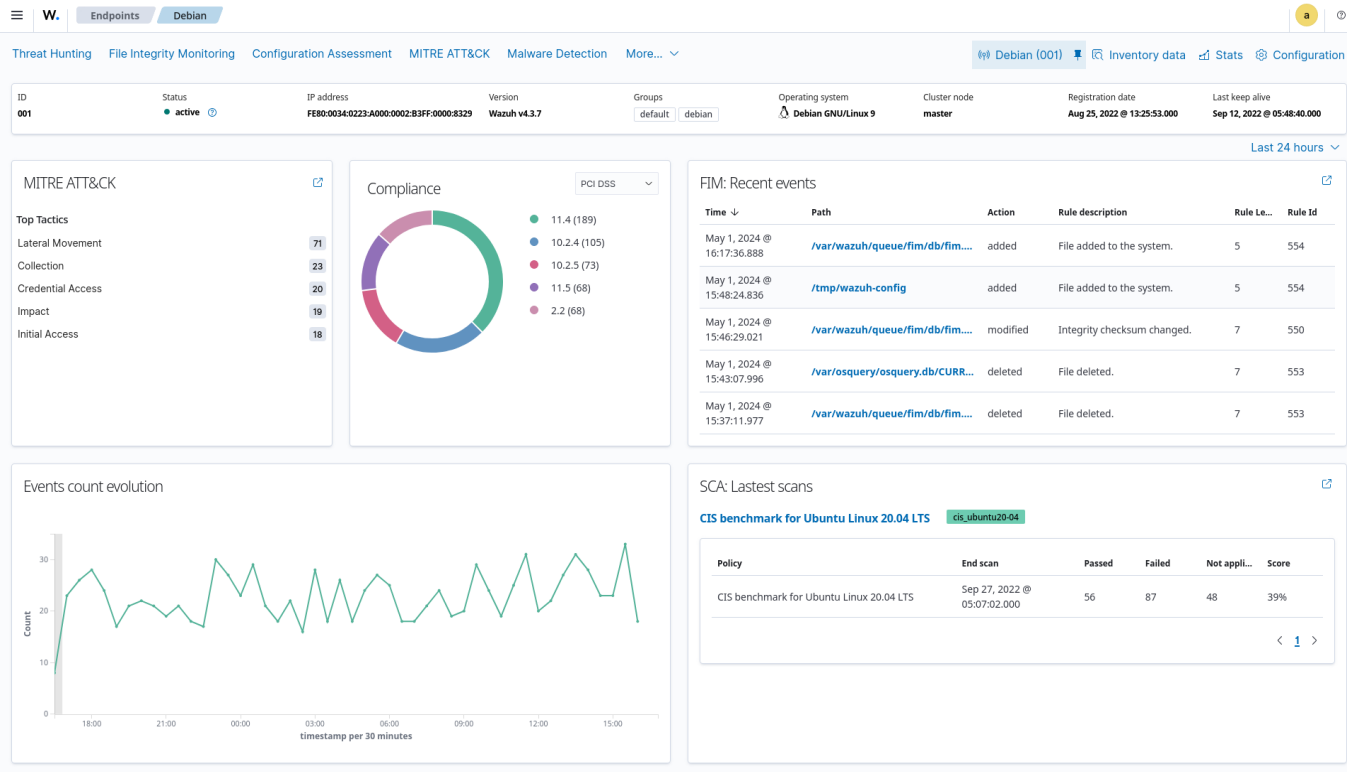

Figure 2.3 Wazuh Dashboard

The web interface helps users navigate through the different types of data collected by the Wazuh agent, as well as the security alerts generated by the Wazuh server. Users can also generate reports and create custom visualizations and dashboards.

The Wazuh dashboard allows users to manage agents configuration and to monitor their status. As an example, for each monitored endpoint, users can define what agent modules will be enabled, what log files will be read, what files will be monitored for integrity changes, or what configuration checks will be performed.

The Wazuh dashboard also provides a user interface dedicated to manage your Wazuh deployment. This includes monitoring the status, logs, and statistics of the different Wazuh components. It also includes configuring the Wazuh server, and creating custom rules and decoders for log analysis and threat detection.

The Wazuh dashboard also includes an API console for users to interact with the Wazuh API. This can be used to manage the Wazuh deployment (e.g., managing server or agent configurations, monitor status and log messages, adding or removing agents, etc.).

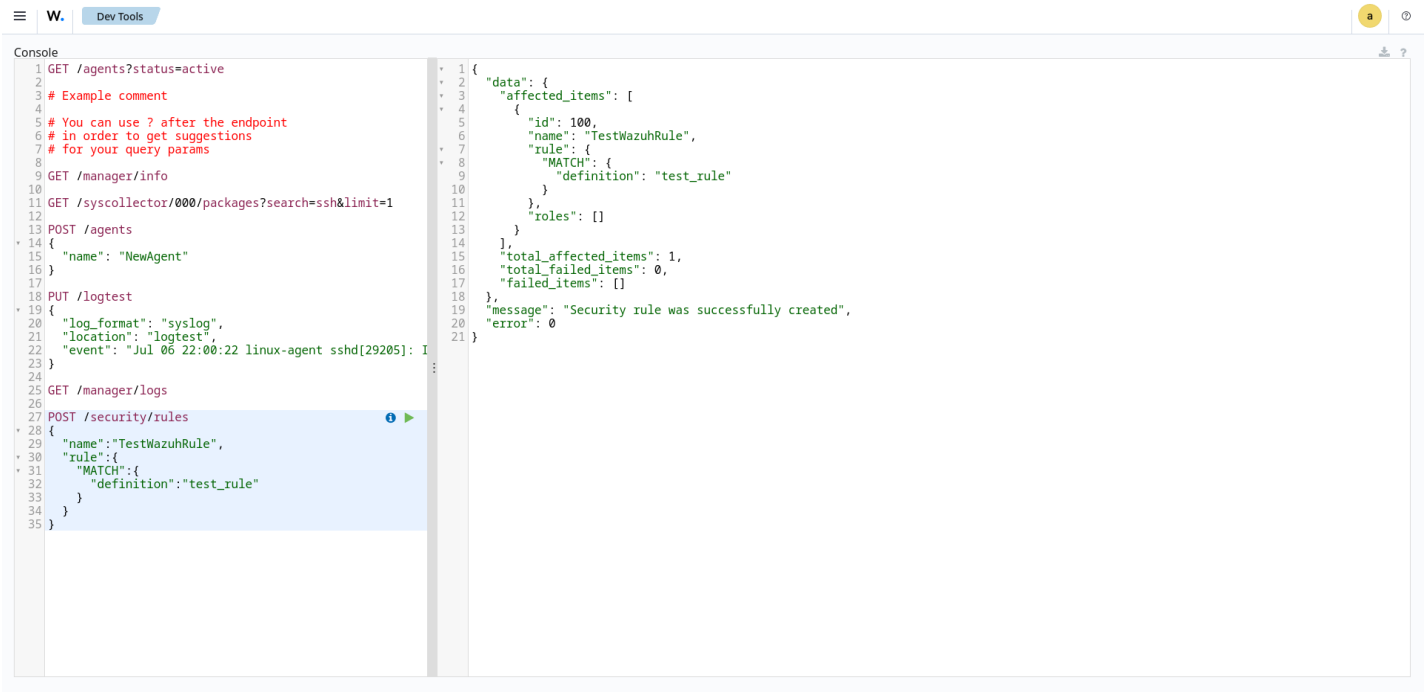

Figure 2.4 Wazuh API Console

2.2.4 Wazuh Agent

The Wazuh agent runs on Linux, Windows, macOS, Solaris, AIX, and other operating systems. It can be deployed to laptops, desktops, servers, cloud instances, containers, or virtual machines. The agent helps to protect your system by providing threat prevention, detection, and response capabilities. It is also used to collect different types of system and application data that it forwards to the Wazuh server through an encrypted and authenticated channel.

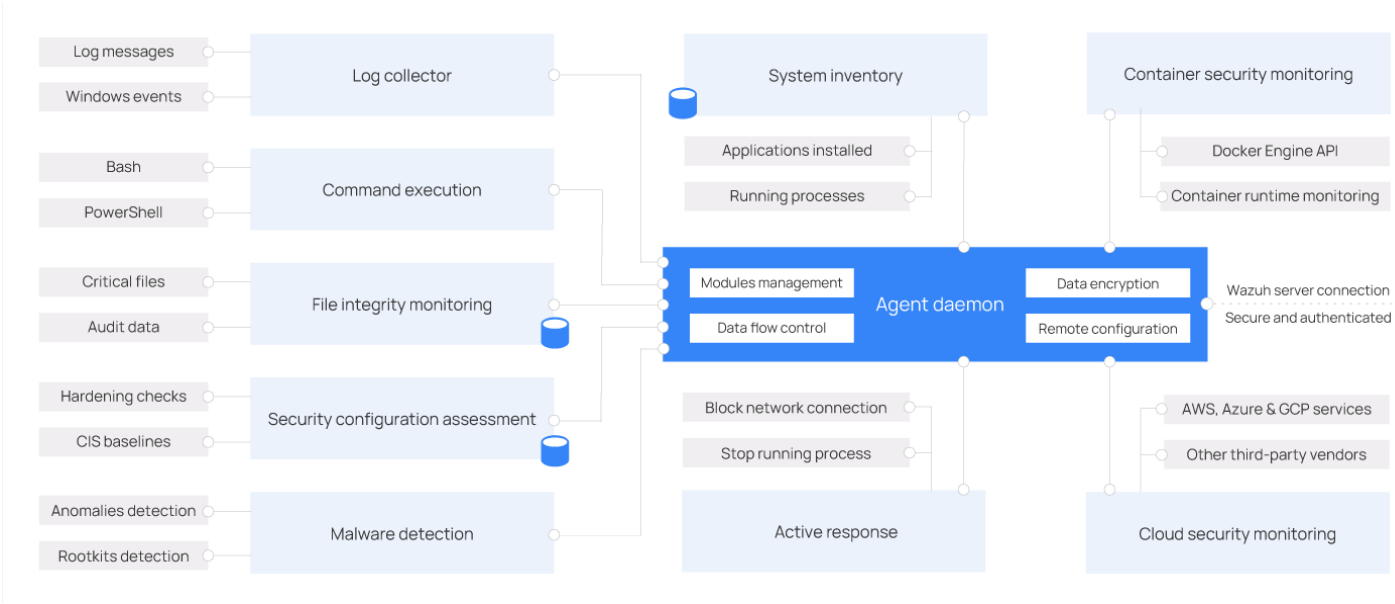

Figure 2.5 Architecture of a Wazuh Agent

The Wazuh agent has a modular architecture. Each component 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 Wazuh agent communicates with the Wazuh server to ship collected data and security-related events. Besides, the agent sends operational data, reporting its configuration and status. Once connected, the agent can be upgraded, monitored, and configured remotely from the Wazuh server.

The communication of the agent with the server takes place through a secure channel (TCP or UDP), providing data encryption and compression in real time. Additionally, it includes flow control mechanisms to avoid flooding, queueing events when necessary, and protecting the network bandwidth.

You need to enroll the agent before connecting it to the server for the first time. This process provides the agent with a unique key used for authentication and data encryption.

2.3 Wazuh System Architecture

The Wazuh architecture is based on agents, running on the monitored endpoints, that forward security data to a central server. The central server decodes and analyzes the incoming information and passes the results, via Filebeat using TLS encryption, along to the Wazuh indexer cluster for indexing and storage.

The diagram as shown in Figure 2.6 represents a Wazuh deployment architecture. It shows the solution components and how the Wazuh server and the Wazuh indexer nodes can be configured as clusters, providing load balancing and high availability.

The Wazuh agent continuously sends events to the Wazuh server for analysis and threat detection. To start shipping this data, the agent establishes a connection with the server service for agent connection. The Wazuh server then decodes and rule-checks the received events, utilizing the analysis engine before using Filebeat to securely transmit alerts and event data to the Wazuh indexer using TLS encryption. Filebeat monitors output data from the Wazuh server and forwards it to the Wazuh indexer, which listens on port TCP/9200 by default. Once indexed, you can analyze and visualize the data through the Wazuh dashboard.

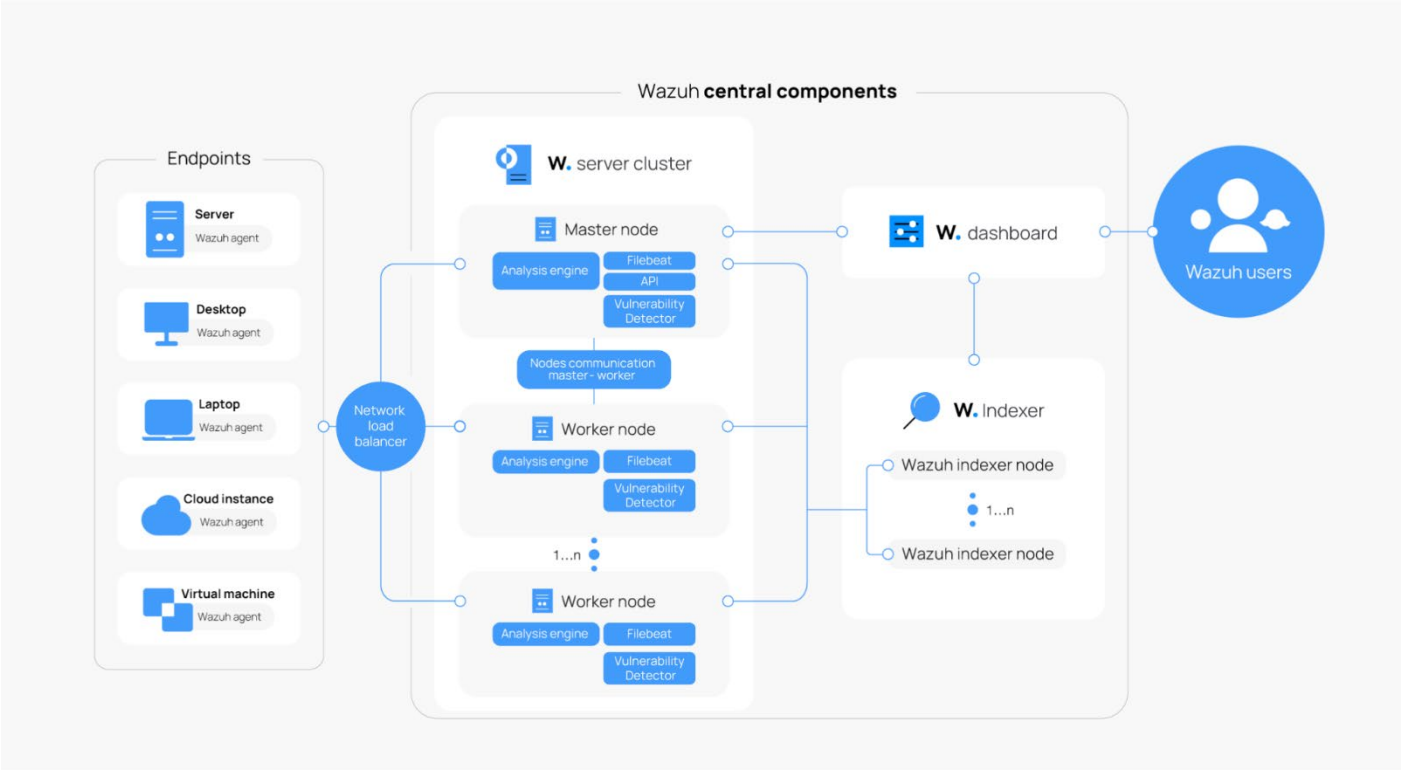

Figure 2.6 Wazuh System Architecture

The Wazuh dashboard queries the Wazuh RESTful API (by default listening on port TCP/55000 on the Wazuh server) to display configuration and status-related information of the Wazuh server and agents. It can also modify agents or server configuration settings through API calls. This communication is encrypted with TLS and authenticated with a username and password.[1]

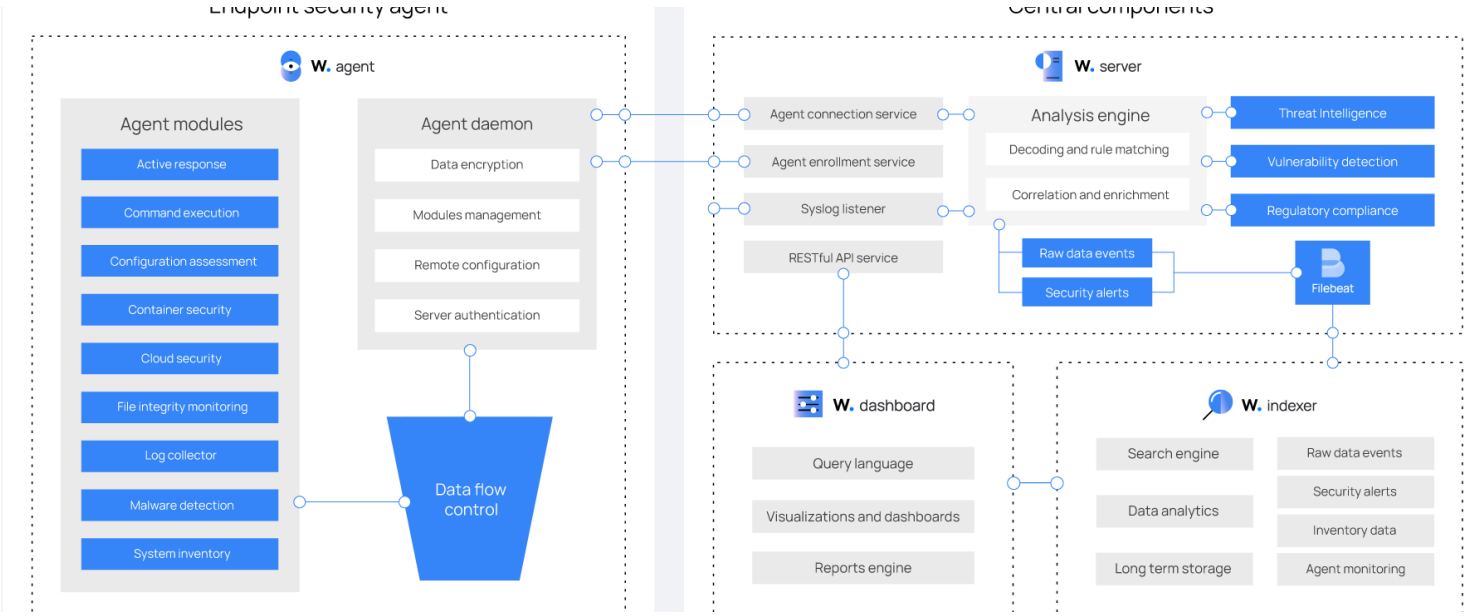

Figure 2.7 Data Flow Diagram of a Wazuh System

2.3.1 Comparison with other SIEM Architecture

From section 2.3.1.1 to 2.3.1.3, comparisons between Wazuh and multiple well-known SIEM tools, including Splunk, Elastic SIEM, IBM QRadar, Microsoft Sentinel, LogRhythm, and ArcSight in terms of scalability, distributed design, and cost are analyzed.

2.3.1.1 Scalability

Wazuh achieves horizontal scalability by clustering both its server component and its indexing component, allowing organisations to increase capacity simply by adding more nodes [7].

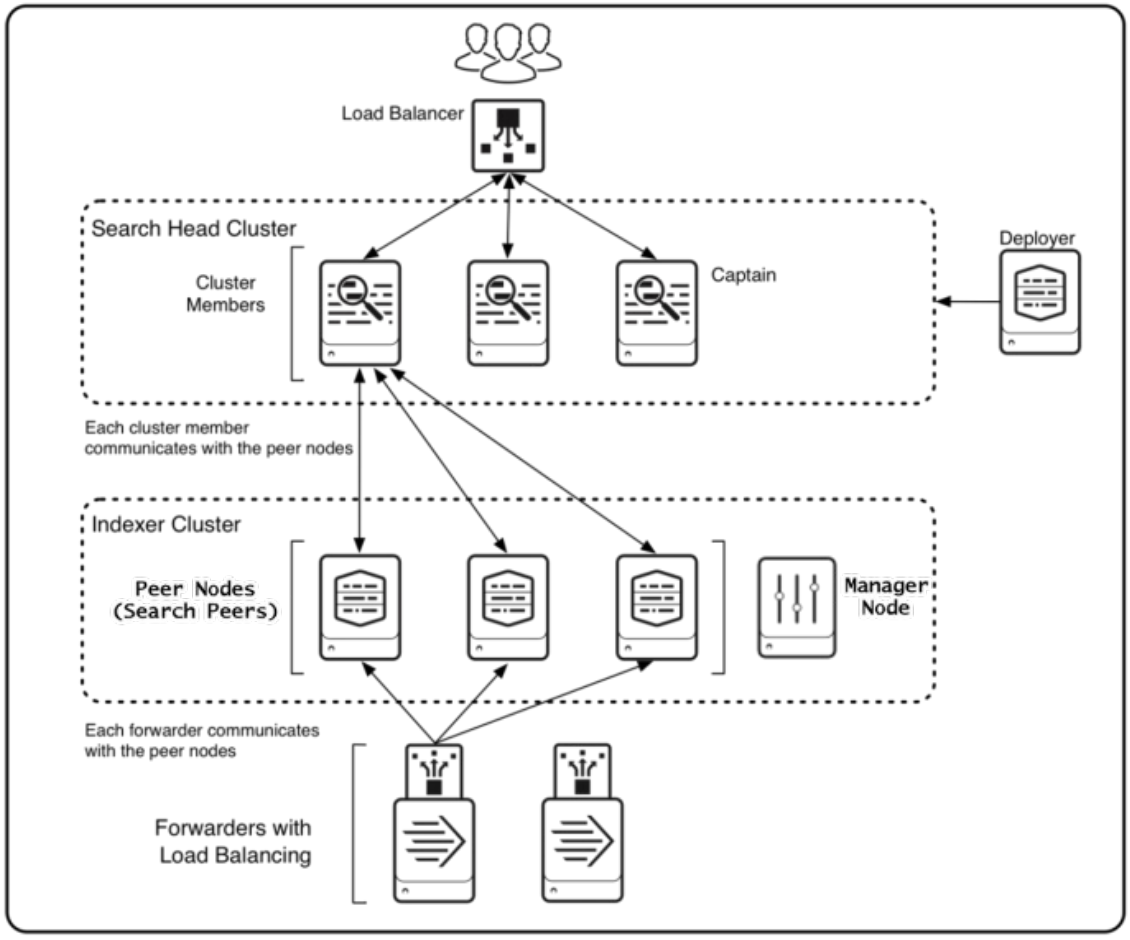

Figure 2.8 Splunk Architecture

Splunk follows a similar scale-out model, distributing data across indexer clusters that can be searched in parallel by search-head clusters [8].

Figure 2.9 Elastic SIEM Architecture[9]

Elastic SIEM inherits the native sharding and replica mechanisms of Elasticsearch, automatically balancing data and queries across the cluster [10]. Microsoft Sentinel, being cloud-native, elastically expands ingestion and analytics resources on Azure without customer-managed infrastructure [11].

In contrast, IBM QRadar, Micro Focus ArcSight and LogRhythm began as appliance-centric platforms: they scale by adding dedicated collector or processor modules, and only newer releases introduce distributed correlation engines or unlimited-data options [12–14]

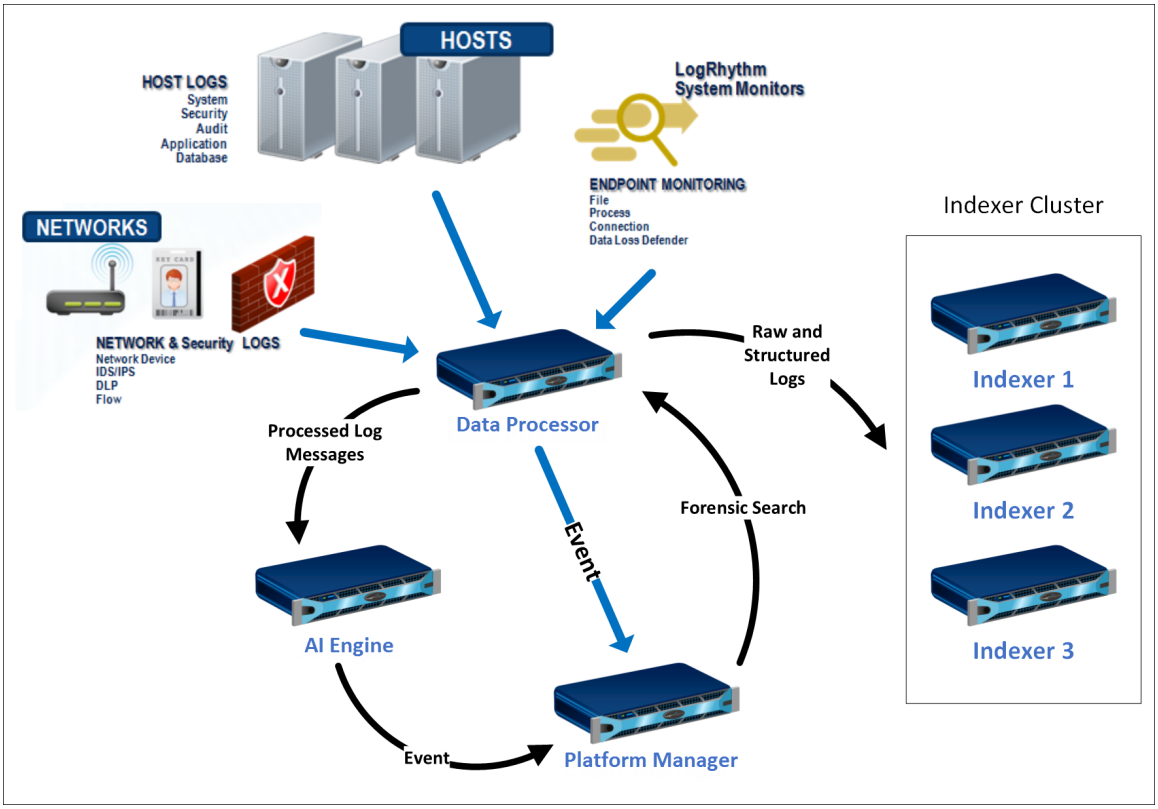

Figure 2.10 LogRhythm SIEM Architecture[19]

2.3.1.2 Distributed Design

Modern platforms decouple ingestion, processing and storage for resilience. Wazuh agents forward events to any node in a multi-master server pool, which then forwards normalised alerts to an OpenSearch/Elasticsearch cluster, eliminating single points of failure [7]. Splunk's distinct search-head and indexer tiers implement a true peer-to-peer federation so that searches run concurrently across sites [8]. Elastic uses primary-and-replica shards to keep copies of every index on different machines, ensuring both high availability and locality-aware search [9]. Sentinel's micro-service architecture spans multiple Azure regions, automatically provisioning data-connector, analytics-rule and hunting components as load grows [11]. QRadar and ArcSight deploy distributed collectors but still rely on a central console or ESM for correlation and GUI access [12,13], while LogRhythm distributes collection and indexing yet keeps coordination in a Platform Manager service [14].

2.3.1.3 Cost

Wazuh is open-source under GPL v2/Apache 2.0 licences, so there are no software fees; organisations pay only for the hardware or cloud resources they provision [18]. Elastic SIEM offers a free Basic tier with core security features and optional paid subscriptions, keeping the entry cost low and predictable [17]. Commercial tools are significantly more expensive: industry analyses place Splunk Enterprise Security licences as high as $1,800 to $18,000 per year for just 1–10 GB/day of data, with costs scaling linearly after that [15]. IBM QRadar and ArcSight continue to license by events-per-second or appliance capacity, which can push large deployments into six-figure annual spend [12,13]. LogRhythm now markets a “True Unlimited Data Plan”, but it is still subscription-based and price-on-request, leaving total cost ambiguous [14]. Microsoft Sentinel adopts consumption pricing—about $5.22 per GB ingested on a pay-as-you-go basis—which offers flexibility but can become costly for multi-terabyte environments [16]. Consequently, from a purely financial perspective, Wazuh (and to a lesser extent Elastic) provides the lowest barrier to scale, whereas proprietary and cloud-metered SIEMs demand careful budgeting as data volumes grow.

2.4 Wazuh Related Work

Wazuh is well known as an open-source SIEM tool, which has enabled proactive detection and response to cyber threats with its powerful log analysis capabilities. Stanković et al. [2] regulate the architecture of Wazuh by combining agents, servers, and the Elastic Stack, enabling real-time detection of security alerts with actionable information. It offers features such as intrusion detection, file integrity monitoring, and compliance with PCI DSS and GDPR. The integration with frameworks like the MITRE ATT&CK framework improves its capabilities in threat detection, while the scalability of log management makes it ideal for handling representations of distributed systems like university networks. Such features position Wazuh as a powerful tool for centralizing the monitoring and security of heterogeneous environment infrastructures.

Another research by Jumiaty and Soewito [3] showed that the Wazuh package was

able to detect SQL injection and brute-force attacks based on real-time log monitoring, scanning in multiple stages, and adding a comprehensive ruleset. They combined Wazuh with TheHive, which assists incident response by automatically creating cases for alerts and assigning them to the appropriate team members. This, in turn, enhanced the efficiency of threat detection and response, with admins receiving rapid notifications via Telegram through integration possibilities offered by ZeroTier. The objective of this research is to demonstrate Wazuh as an open-source, multi-platform tool capable of providing security visibility for applications in a secure and scalable manner across different environments.

In his study [4], Hafiz Javid analyzed how Wazuh operates as both a Security Information and Event Management (SIEM) and an Extended Detection and Response (XDR) solution for on-premises environments. The findings revealed that Wazuh excels in identifying and mitigating vulnerabilities, using tools like VirusTotal to efficiently detect risks and File Integrity Monitoring (FIM) for real-time protection. Practical experiments, including simulated attacks with Atomic Red Team and vulnerability scans, highlighted Wazuh's capability to detect threats automatically. The study demonstrated that Wazuh is effective across various operating systems, solidifying its role as a valuable tool in modern cybersecurity practices.

Mansoor Ebrahim et al. [5] explored Wazuh Cloud integration for small business security. In their research, they noted the performance of the Host-Based Intrusion Detection System (HIDS) in Wazuh against port scans, Denial of Service (DoS), and Metasploit intrusions. Through Cloud Wazuh, they demonstrated operations with tailored security regulations as well as lowered costs by eliminating the need for an internal IT security team. This research highlighted the scalability and user-friendly aspects of Wazuh as an effective security solution for protecting digital assets in small enterprises with minimal interruptions.

Overall, these solutions highlight Wazuh’s versatility in its ability to fit in multiple cybersecurity needs. Stanković [2] demonstrated the use of Wazuh agents, servers, and Elastic stack for real-time threat detection, while Jumaity and Soewito [3] displayed Wazuh’s ability to detect SQL injection and different types of brute-force attacks. Hafiz

Javid [4] emphasized Wazuh's role using SIEM and XDR solutions to detect and monitor vulnerability and file integrity in setups. Similarly, Mansoor Ebrahim [5] uses Wazuh as cloud for small businesses, allowing Wazuh users to have a quick and easy scalable, cost-effective solution for security management. These studies highlight multiple aspects of Wazuh' flexibility and its ability to adapt to various work environments and user styles ranging from small enterprises to university environments, while maintaining its core functionalities of intrusion detection, compliance, and real-time monitoring.

2.5 Tools

2.5.1 Ansible

Ansible is an open-source automation tool used for IT configuration management, application deployment, and task automation. It helps system administrators and DevOps teams automate repetitive tasks and manage complex environments easily and reliably.

Key Features:

- Agentless: No need to install any software on the target systems. Ansible connects via SSH or WinRM.
- Simple Language: Uses YAML (in files called playbooks), which is human-readable and easy to write.
- Idempotent: Ensures tasks are only done when necessary running the same playbook multiple times won't cause issues.
- Cross-Platform: Works across Linux, Unix, Windows, cloud platforms, containers, and more.
- Extensible: Can be integrated with Jinja2 templates for dynamic configuration and supports modules to extend functionality.

How Ansible Works:

1. Control Node: The machine where Ansible is installed and from which commands are run.
2. Managed Nodes: The machines being configured, which don't require an agent.

3. Inventory File: Defines the list of managed nodes (hosts).
4. Modules + Playbooks: Define what to do and how to do it using YAML.

The architecture of Ansible separated devices into two categories: control node and managed node. A control node can manage every node, including itself, with minimal amount of human interactions required on any managed node.

Components of an Ansible environment include an inventory, a list of hosts that will be managed through the control node, and playbooks, which list the actions, executions and hosts that will be affected.

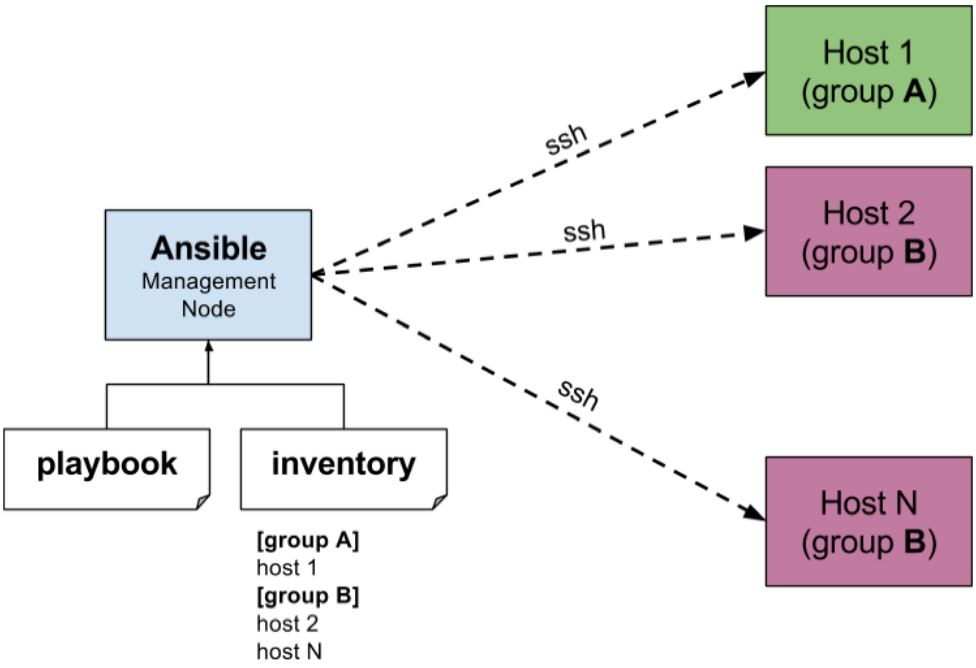

Figure 2.11 Ansible Architecture

2.5.1.1 Integration with Jinja2

Jinja2 is a template engine that enables dynamic file modification and population at runtime. When used with Ansible, it can automate the editing of Wazuh configuration files, rules, and scripts.

For example, the opensearch.yml configuration file on each Wazuh Indexer must include the list of all other Indexers in the cluster. Without automation, adding a new Indexer requires manually updating both the new and all existing opensearch.yml files, an effort that grows multiplicatively with the number of nodes.

Using Jinja2 templating with Ansible, however, this process is automated. Ansible reads the full inventory of Indexers, dynamically renders the appropriate opensearch.yml file for each node, and deploys it accordingly. This eliminates the need for manual updates and ensures consistency across the cluster.

```
{% for host in groups['indexer'] %}
  - "{{ host }}"
{% endfor %}
cluster.name: "wazuh-cluster"
discovery.seed_hosts:
{% for host in groups['indexer'] %}
  - "{{ hostvars[host]['ansible_host'] }}"
{% endfor %}
```

Figure 2.12 Jinja2 template for Indexer's configuration file

CHAPTER 3
METHODOLOGY

This chapter describes the architecture and design of Wazuh and its multi-site design of the project.

3.1 Requirement Gathering

An organization with operations spanning across two geographically dispersed locations wants to implement a SIEM solution, the requirements are as followed:

- Security-related operations should perform independently from each site.
- Logs data collection for each site is independent of the other.
- Each site's logs are backed up across all sites.
- The cost of the solution should be minimal when possible.
- Ensure that each endpoint connects to the nearest geographical manager to minimize network congestion.
- A centralized dashboard can be used to view security events and alerts from all sites.

3.2 Requirement Analysis

In accordance to the gathered requirements, Wazuh SIEM was chosen for a variety of reasons:

- Wazuh is an open-source and free SIEM solution.
- The architecture of Wazuh SIEM, which consists of servers, indexers, and agents, can be deployed separately on each site, creating independence from other sites.
- Indexes and alerts from each site's indexer can be customized to distinguish the site of origins with unique patterns.
- Despite being in different sites, Wazuh indexers on each site can communicate with each other to create backup and redundant storage.
- An endpoint can be chosen to connect to any server. For this project the closest geographical server is chosen for each agent.

- The dashboard component, which uses Wazuh RESTful API, can be configured to accept data from each site's servers thus creating unified monitoring.

3.3 System Design

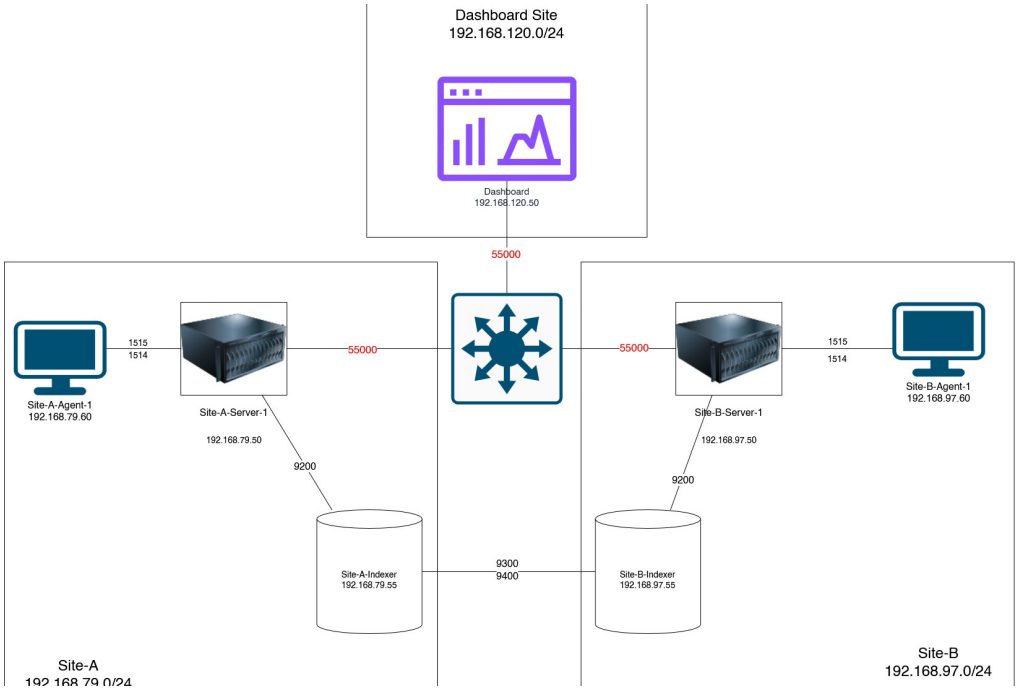

Figure 3.1 Wazuh Multi-Site Architecture

The sites, despite not being geographically separated, are logically separated using a router, which divides the project into three subnets while having the same functionalities:

- Site A:
  - Has an IP address space of 192.168.79.0/24
  - Wazuh Indexer (indexer_a_1) has an address 192.168.79.55.
  - Wazuh Server (server_a_1) has an address 192.168.79.50
  - Wazuh Agent (agent_a_1) has an address 192.168.79.60
  - Each Wazuh component is installed on Ubuntu 20.04

- Site B:
  - Has an IP address space of 192.168.97.0/24

    - Wazuh Indexer (indexer_b_1) has an address 192.168.79.55.
    - Wazuh Server (server_b_1) has an address 192.168.97.50
    - Wazuh Agent (agent_b_1) has an address 192.168.97.60
    - Each Wazuh component is installed on Ubuntu 20.04
- Site C:
    - Has an IP address space of 192.168.120.0/24

The dashboard is accessible using a web browser using the URL of 192.168.120.50 and inputting username and password of both “admin”.

CHAPTER 4
EVALUATION AND RESULTS

This chapter details the implementation, experimentation, analysis, and evaluation of the multi-site implementation:

4.1 Virtualized Implementation

The multi-site set up is performed within a single computer through multiple virtual machines. The setup acted as a basis for the physical implementation later on.

4.1.1 Device Specification

- Host Device
    - Operating Systems: Windows 10
    - Processor: 6
    - Memory: 32GB
    - Virtualization Software: Oracle VirtualBox
- Virtual Machine
    - Wazuh Server, Indexer, and Dashboard
        - Operating Systems: Ubuntu 20.04
        - Processor: 1
        - Memory: 3GB
    - Wazuh Agent
        - Operating System: Lubuntu 24.04
        - Processor: 1
        - Memory: 2GB
    - Router
        - Operating: Ubuntu Server 20.04
        - Processor: 1
        - Memory: 1GB

4.1.2 Pre Implementation

- Inside Oracle VirtualBox application, we created 3 VirtualBox Host-Only Ethernet

Adapter networks:

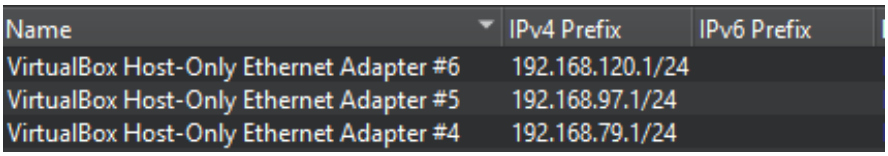

Figure 4.1. Virtual Machine network setting

- We installed 8 virtual machines classified as followed:
    - Site A (192.168.79.0/24):
        - Server (192.168.79.50) designated as server-a-1
        - Indexer (192.168.79.55) designated as indexer-a-1
        - Agent (192.168.79.60) designated as agent-a-1
    - Site B (192.168.97.0/24)
        - Server (192.168.97.50) designated server-b-1
        - Indexer (192.168.97.55) designated indexer-b-1
        - Agent (192.168.97.60) designated agent-b-1
    - Dashboard (192.168.120.0/24)
        - Dashboard (192.168.120.50) designated dashboard-1
    - Router
        - Site A interface: 192.168.79.25
        - Site B interface: 192.168.97.25
        - Dashboard interface: 192.168.120.25
        - Bridged Interface: 192.168.1.128
- Next, we edited /etc/netplan/01-netcfg.yaml file to allow routing between each network along with internet access

```
---
network:
  version: 2
  renderer: networkd
  ethernets:
    enp0s3:
      dhcp4: true
    enp0s8:
      dhcp4: false
      addresses:
        - 192.168.120.25/24
    enp0s9:
      dhcp4: false
      addresses:
        - 192.168.79.25/24
    enp0s10:
      dhcp4: false
      addresses:
        - 192.168.97.25/24
```

Figure 4.2: Router configuration

- On the dashboard node, we installed Ansible, an open-source automation tool that simplifies repetitive tasks in managing other components remotely and automatically. Its capabilities include:
    - Installing packages
    - Running shell commands
    - Copying files
    - Managing specific groups of computers at once.
- Requirements for Ansible are as followed:
    - The managed device must have Python installed.
    - The managed device must have the managing device's SSH key as known.
- Ansible manages other devices through inventory and playbooks.
    - Playbooks list specific instructions to operate on hosts listed in the inventory.
- The playbooks are organized into different categories, each affecting different hosts

depending on the type of the host.

```yaml
- name: Step 1 Install Wazuh server and Filebeat
  hosts: server
  become: yes
  vars_files:
    - wazuh.yml
  tasks:
    - name: Install wazuh-manager and filebeat
      apt:
        name:
          - wazuh-manager
          - filebeat
        state: present
        update_cache: yes

    - name: Download wazuh-certs-tool.sh
      get_url:
        url: https://packages.wazuh.com/4.11/wazuh-certs-tool.sh
        dest: /home/{{ ansible_user }}/wazuh-certs-tool.sh
        mode: '0755'
```

Figure 4.3 Example playbook for Wazuh Server

```
# === DASHBOARD ===
[site_dashboard]
dashboard_1 ansible_host=192.168.120.50 ansible_user=wazuh ansible_connection=local

# === SITE A ===
[site_a_indexer]
indexer_a_1 ansible_host=192.168.79.55 ansible_user=wazuh

[site_a_server]
server_a_1 ansible_host=192.168.79.50 master=true cluster_key=53a5e452e4d96a1f48ab2e596
#server_a_2 ansible_host=192.168.133.26 master=false ansible_user=wazuh

[site_a_agent]
agent_a_1 ansible_host=192.168.79.60 ansible_user=wazuh


# This is just test LB
#[site_a_loadbalancer]
#lb_a_1 ansible_host=192.168.133.25 ansible_user=nginx ansible_sudo_pass=wazuh

# === SITE B ===
[site_b_indexer]
indexer_b_1 ansible_host=192.168.97.55 ansible_user=wazuh
```

Figure 4.4 Ansible inventory classifying hosts into different category

- Utilizing Jinja2 for file templating, we can create j2 files that will be distributed during installation and automatically configured to match each host's requirements.

```
# Wazuh - Filebeat configuration file

{% set site_server_group = group_names | select('match', 'site_.*_server') | list | first %}
{% set site_id = site_server_group | regex_replace('site_(.*)_server', 'site_\\1') %}
{% set indexer_group = site_id + '_indexer' %}
{% set indexer_hosts = groups[indexer_group] | map('extract', hostvars, 'ansible_host') | list %}

# DEBUG: Rendering for {{ inventory_hostname }}
# DEBUG: This server is in groups: {{ group_names }}
# DEBUG: Using indexer group: {{ indexer_group }}
# DEBUG: Indexer hosts: {{ indexer_hosts }}


output.elasticsearch:
  hosts: [{% for ip in indexer_hosts %}"{{ ip }}:9200"{% if not loop.last %}, {% endif %}{% endfor %}]
  protocol: https
  username: "admin"
  password: "admin"
  ssl.certificate_authorities:
    - /etc/filebeat/certs/root-ca.pem
  ssl.certificate: "/etc/filebeat/certs/filebeat.pem"
  ssl.key: "/etc/filebeat/certs/filebeat-key.pem"

setup.template.json.enabled: true
setup.template.json.path: '/etc/filebeat/wazuh-template.json'
setup.template.json.name: '{{ indexer_group }}'
setup.ilm.overwrite: true
setup.ilm.enabled: false
```

Figure 4.5 Example Filebeat configuration file template using Jinja2

4.1.3 Implementation

- The test playbook is run using ansible-playbook command to verify requirements.

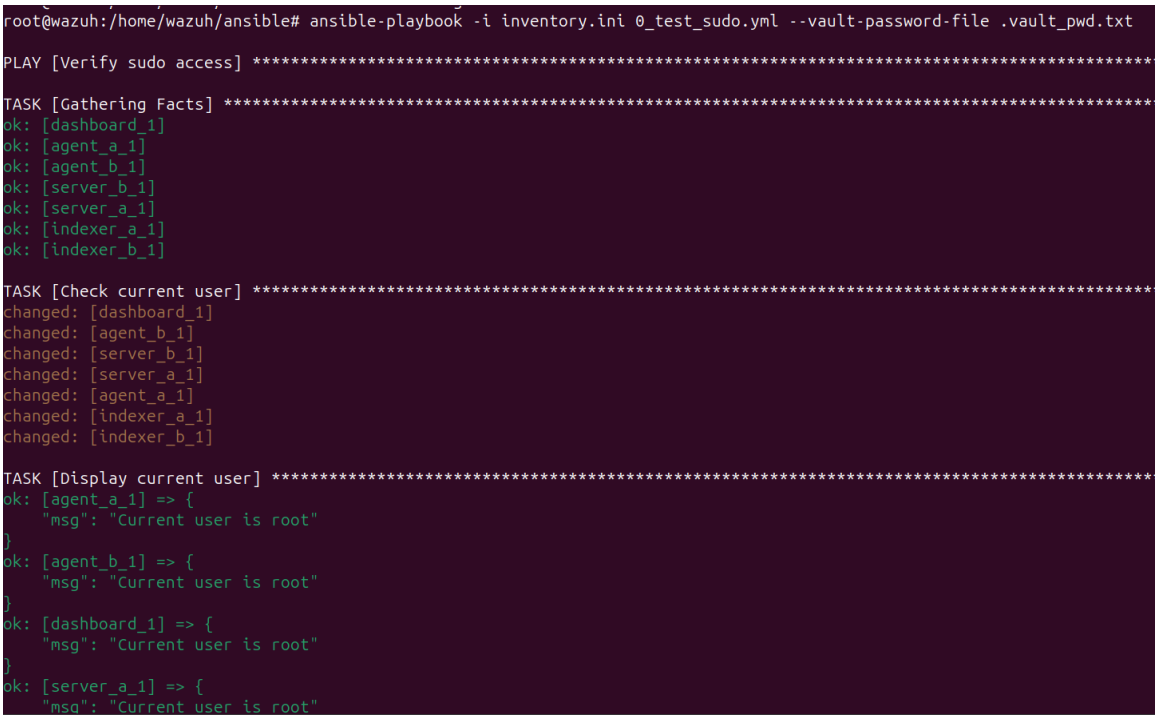

Figure 4.6 Running a test playbook

- The implementation detailed in the playbooks are as followed:
  - All nodes:
    - Configuring package manager proxy server
    - Install additional dependencies
    - Adding Wazuh repository to the package manager
    - Download Wazuh official certification tool.
    - A certificate configuration file (config.yml) template is copied onto every device, with the inventory name and IP being filled dynamically.
  - Dashboard node:
    - On the dashboard node, the certificate tool is run to generate the dashboard certificate along with the root certificates that will be used to create certificates for every Wazuh component later.

- The root certificates will be kept for future component addition.
    - The root certificates are copied to every Wazuh component.
    - The dashboard configuration file's template (opensearch_dashboard.yml) is populated dynamically using the IP of the indexers from the inventory.
    - Server API selector file (wazuh.yml) is also copied and filled using each site's master server node.
- Server nodes:
    - Filebeat configuration file (filebeat.yml) and alert template file (wazuh-template.json) are copied to each server node after their certificate creations. The alert template will be unique based on the specified site name in the inventory while filebeat will only include indexers from the same site.
- Indexer nodes:
    - The Wazuh indexer (indexer_a_1 and indexer_b_2) generate their certificates using the received root certificates.
    - Indexer's configuration files (opensearch.yml) are copied to each indexer node and populated with every other indexer IP address.
- Agent:
    - Only the agent's configuration file needs to be modified to point to the master server on the same site.

4.1.4 Post Implementation

By going to the dashboard IP address in a web browser (192.168.120.50), the browser will attempt to warn us that the dashboard URL is not secured, this can be ignored since our dashboard certificates are self-signed by us using root certificates as trusted authorities.

Accessing the dashboard allows us to choose which site to view the information through a selector at the top of the web page, with each site being completely independent from one another.

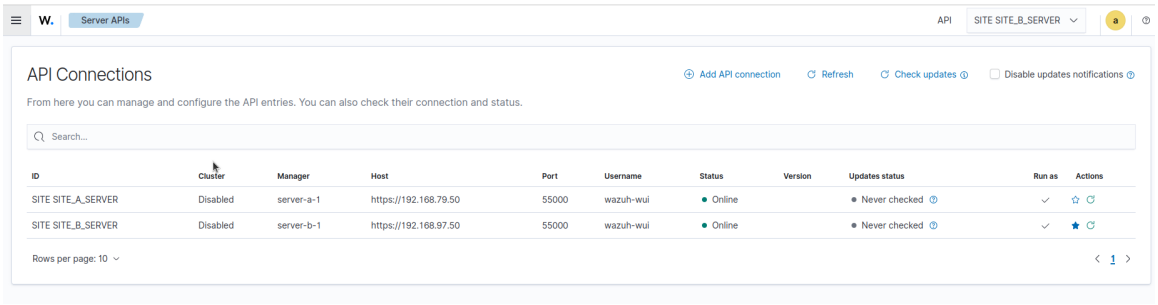

Figure 4.7 Wazuh Dashboard with Site Selector enable

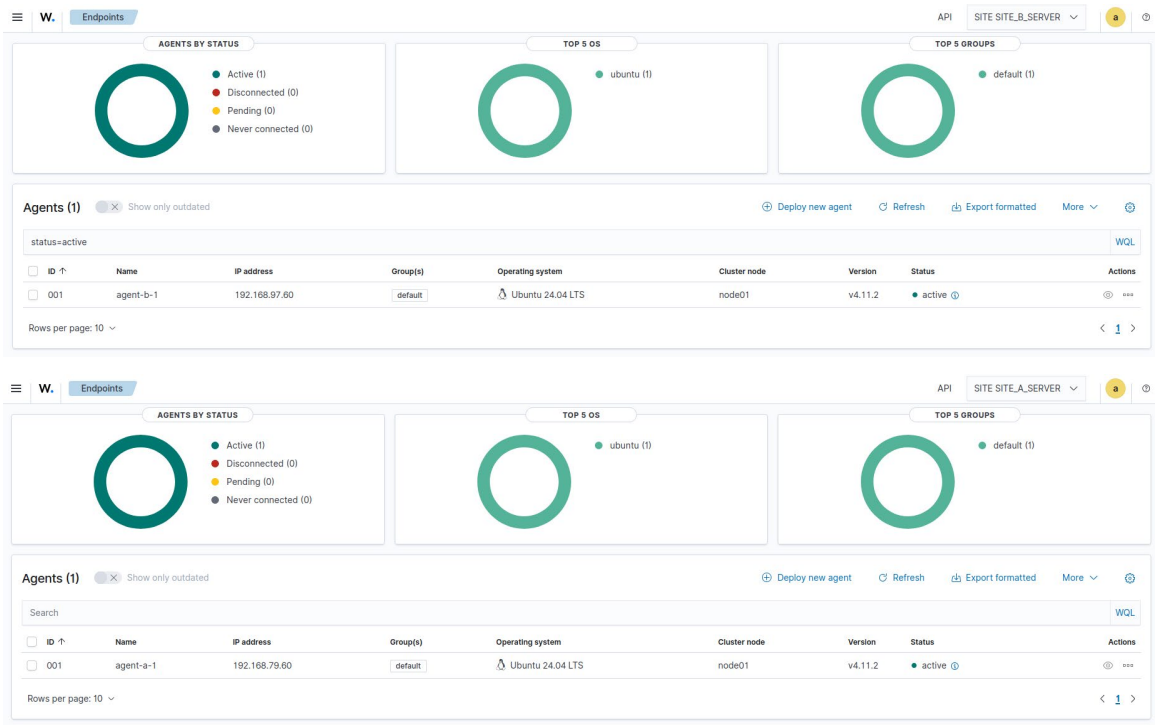

Figure 4.8 and 4.9 Agents from both sites are counted and viewed independently

4.1.5 Verification

To confirm that we're implementing the system correctly based on our requirement, we need to verify whether the Jinja2 automated configuration files were written correctly during ansible software automation process:

- config.yml
    - Used by every Wazuh component during certificate creation.
    - E.g. type = indexer, name = server_a_1, IP = 192.168.79.50

```
nodes:
  server:
    - name: server_a_1
      ip: 192.168.79.50

# DEBUG: Rendering for server_a_1
# DEBUG: group_names = ['server', 'site_a_server', 'wazuh']
# DEBUG: role = server# DEBUG: IP = 192.168.79.50
```

Figure 4.10 Certificate configuration file for server_a_1

4.1.6 Experimentation

4.1.6.1 Event Isolation Testing

- Objective
    - To verify that events and alerts from one site do not show up on the other site.
    - To ensure that each site's server nodes handle their own site's alerts.
- Procedures
    - We plan to introduce a malicious actor (192.168.79.225) to site A who will attempt to brute force SSH access to agent_a_1 using hydra.
    - Afterwards, the dashboard for site A and site B will be inspected for alerts from agent_a_1.

- Testing

```
root@wazuh:/home/wazuh# hydra -t 4 -l root -P password.txt 192.168.79.60 ssh
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or se
cret service organizations, or for illegal purposes (this is non-binding, these *** ignore l
aws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-04-18 11:42:03
[DATA] max 4 tasks per 1 server, overall 4 tasks, 7 login tries (l:1/p:7), ~2 tries per task
[DATA] attacking ssh://192.168.79.60:22/
```

Figure 4.11 Actor attempting to brute force SSH into agent_a_1

- Result
    - The alert showed up only when inspecting site A API in the dashboard.
    - Site B displayed no alert related to the event.
    - Site A alerts also displayed a unique pattern to distinguish the origin of the alert.
- Conclusion
    - By separating the set up modularly (sites), we can isolate problems and alerts to a specific site. This will be crucial especially when the set up is scaled to accommodate more Wazuh components and sites.

4.1.6.2 Site Isolation Testing

- Objective
    - To verify that the remaining sites, including the dashboard, can remain operational even when a site is considered down.
    - To verify that each site performs independently of each other.
- Procedures
    - Starting by shutting down each individual Wazuh component on Site B, we can observe the changes they affected from the dashboard.

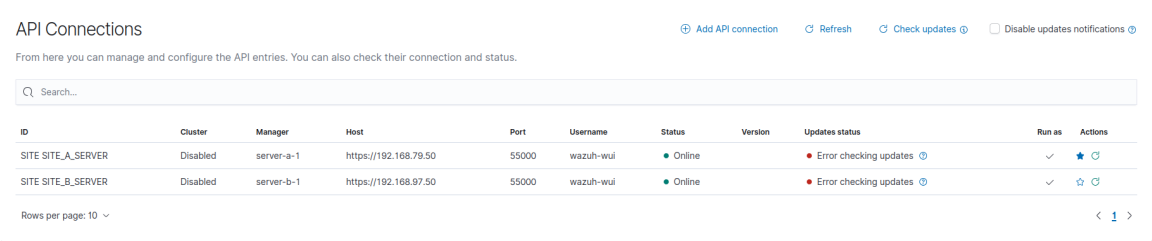

Figure 4.12 Current dashboard status showing both sites as online.

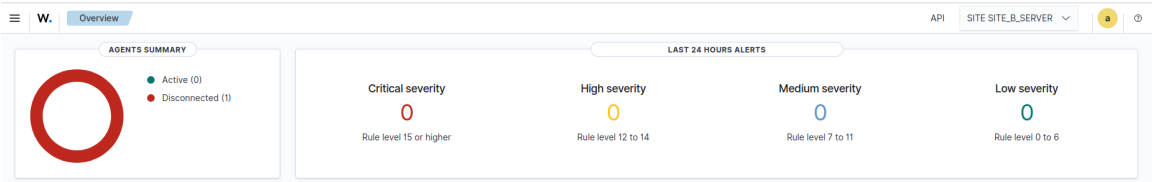

Figure 4.13 Dashboard for Site B API after agent_b_1 was brought offline

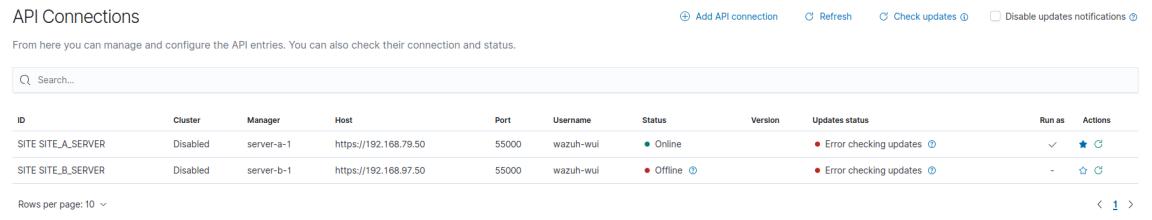

Figure 4.14 Dashboard status after server_b_1 was brought offline.

- Result
  - We are unable to select or view Site B alerts and agents status while they are powered off.
  - Site A servers and agents remained operational and can be viewed on the dashboard as normal.

- Conclusion

- The result of site isolation testing reinforces the design of multi-site implementation. By giving Wazuh components independence to operate in their own sites, we can ensure overall operability even during downtimes.

4.1.6.3 Component Scalability Testing

- Objective
  - To verify that additional Wazuh components and new sites can be added without compromising existing implementation.
  - To create a cluster of Wazuh components that will enhance existing capabilities.
  - To integrate system design components such as load balancer to reroute traffic between agents and server clusters.
  - To verify that cluster installation on one site does not affect the others, and that we can independently scale one site without affecting the others.
- Procedures
  - We added a second Wazuh server (192.168.79.51) to function as a worker node while the existing server operates as a master node.
  - In addition to a second server node, an NGINX load balancer (192.168.79.88) is deployed to help facilitate traffic between both servers and agents. A second Site A agent (192.168.79.70) is also deployed.
  - Instead of connecting to either server, the agents are connected to the load balancer, which will handle the routing itself.
  - We can configure the existing Ansible inventory to include the new Wazuh components and rerun the playbooks to deploy them.
- Testing
  - After installation, we can verify that agent_a_2 has been enrolled and displayed in the dashboard and that both agent_a_1 and agent_a_2 are connected to different servers in the cluster.

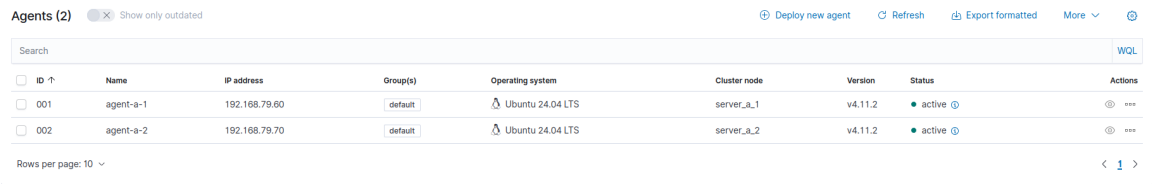

Figure 4.15 Two Site A agents connecting to two different servers

- Running the cluster command also displayed that the servers are now working in a cluster.

```
root@server-a-1:/home/wazuh# /var/ossec/bin/cluster_control -l
NAME        TYPE    VERSION  ADDRESS
server_a_1  master  4.11.2   192.168.79.50
server_a_2  worker  4.11.2   192.168.79.51
```

Figure 4.16 Server cluster status

- Result
  - By powering off a worker server of a cluster, the load balancer will automatically reroute its connected agent to the next available server.

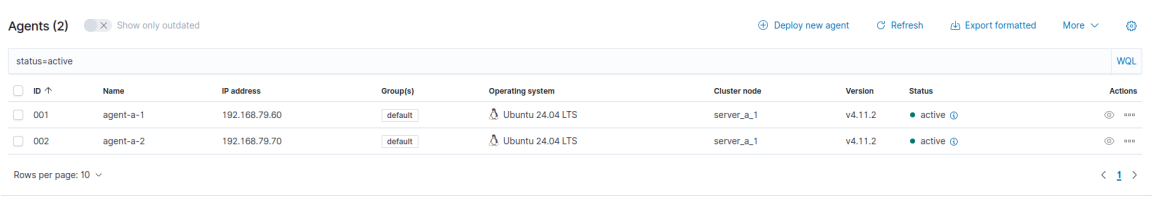

Figure 4.17 Wazuh agent is automatically rerouted to another server

  - The rerouting does not work with the master server (server_a_1) being offline because the multi-site dashboard uses each site's master server address for its API usage and display.
- Conclusion
  - The cluster architecture of Wazuh servers ensures availability for a site's agents in emergency events that cause server downtimes.

- The independent nature of the multi-site implementation allows for freedom of design choices. Sites can have different amounts of servers and agents without affecting the others.
- Incorporating load balancer makes configuration of agents easier, since instead of providing them a list of fallback servers, the load balancer will handle the list of servers and route each agent based on criteria.
- The master/server architecture of the cluster is static, meaning that when the master server of a cluster is down, a worker server cannot become a master server, thus rendering the entire cluster as offline.
- For the dashboard to detect and display a site API as online, the site's master node must be online.
- The reason why site A and B's servers are not joined together into one cluster such that Site A's agents can connect to site B's servers and vice versa is because if a master server is offline, either if it's a site A's server or site B's, then no alerts will be sent to the dashboard.

4.2 Physical Implementation

Despite the success of the virtual implementation and testing, the virtual environment has a limited amount of a host computer's resources which means that the amount of virtual machines that can be run simultaneously are also limited which also leads to impossibility of further scalability testing.

4.2.1 Device Specification

Each device in the implementation is installed on separate computers.

- Dashboard and Ansible Control Node
  - Operating Systems: Ubuntu 20.04
  - Processor: 3
  - Memory: 16GB
- Other Components
  - Operating Systems: Ubuntu 20.04
  - Processor: 2

- Memory: 4GB

The network simulation is achieved using VLAN to separate the network into four distinct networks, with one additional network being used for future testing.

4.2.2 Implementation

The same playbook and inventory used during virtual implementation are also used for physical implementation. The dashboard node is selected to be used as the control node for Ansible automation.

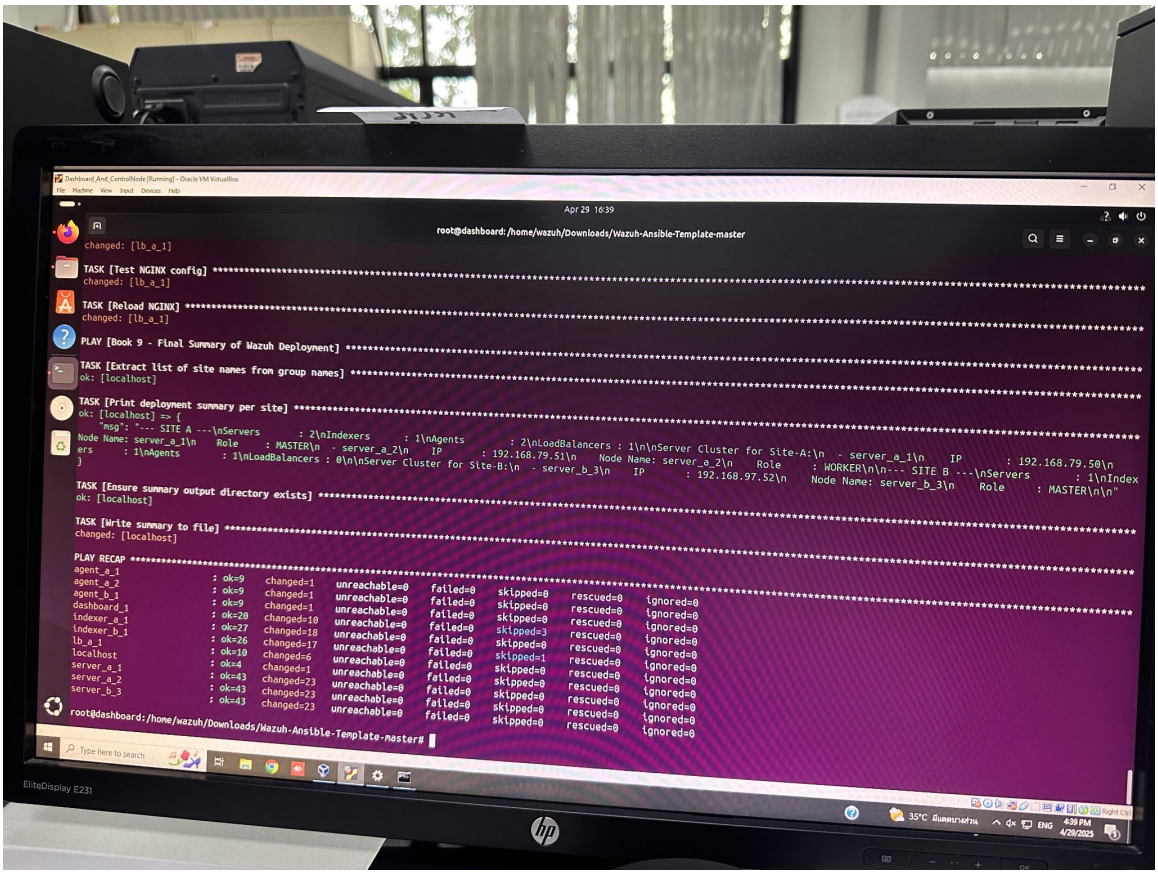

Figure 4.18 The playbook being executed on the dashboard node

4.2.3 Experimentation

4.2.3.1 Single-Site Component Scalability Testing

- Objective

- To verify that any site can scale up and includes more components to enhance local functionalities without compromising other sites in the set up.

- Procedures
  - Using Ansible, new components are specified in the inventory following the existing naming convention to ensure that Jinja2 template engine can extract the hostname and IP into new configuration files.
  - Rerunning the existing playbooks, new components will be installed and configured and existing components reconfigured to accept and connect to the newly added items.

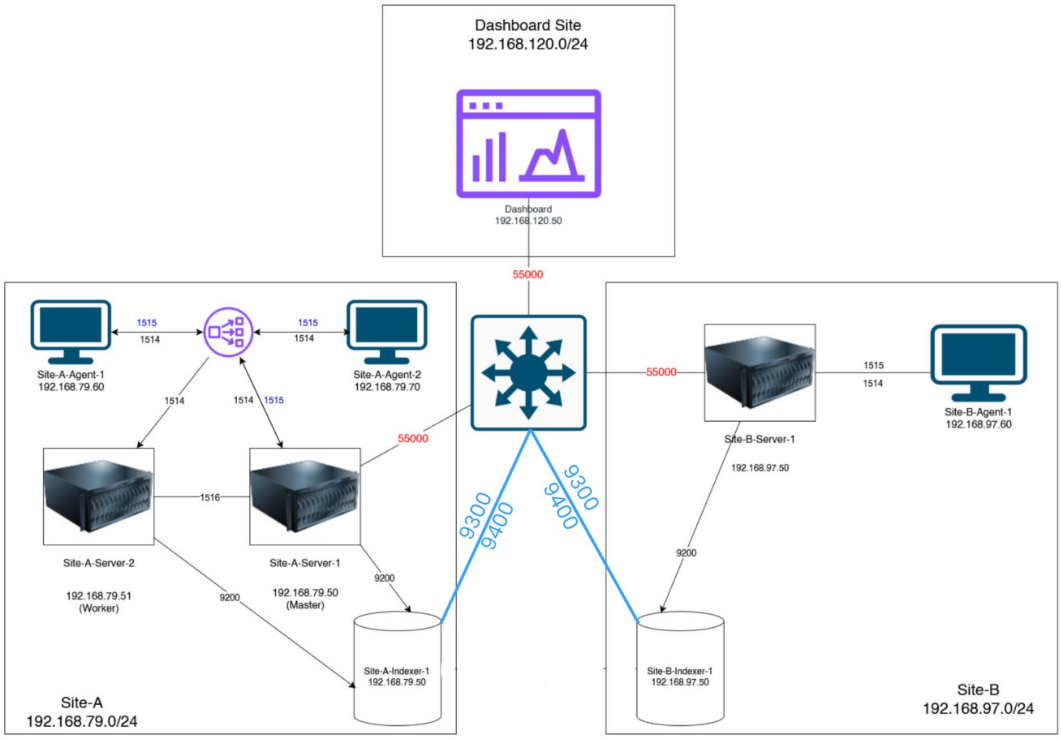

Figure 4.19 Testing Architecture

- Testing

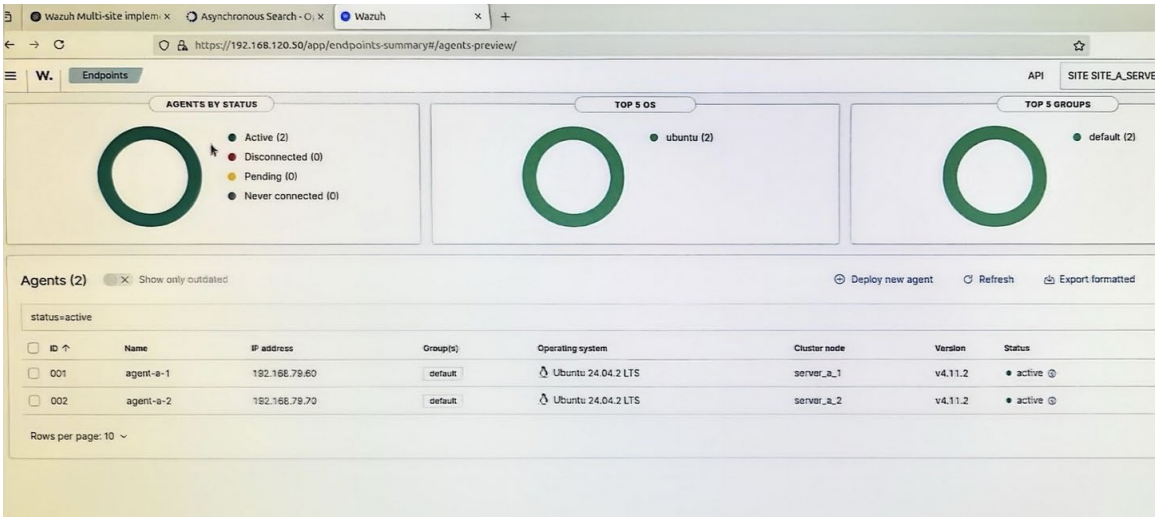

Figure 4.20 Agents of site A connecting to different servers of site A's cluster

- Result
  - Site A's agents are displayed on the dashboard as connected to different servers in site A's cluster.
- Conclusion
  - This test validated the ability to horizontally scale a single site by adding new Wazuh components such as servers, agents, and load balancers without disrupting other sites. It demonstrates how our automation tools dynamically accommodate expansion at a site level and that Wazuh components can be configured to accept both new and existing components.

4.2.3.2 Cross-Site Component Scalability Testing

- Objective
  - To verify that multiple sites can be scaled in parallel without having to disrupt or affect other site's operations.
- Procedures
  - Similar to the previous experiment, new components related to site B are specified in the inventory to be installed.

- Rerunning the existing playbooks, new components will be installed and configured and existing components reconfigured to accept and connect to the newly added items.

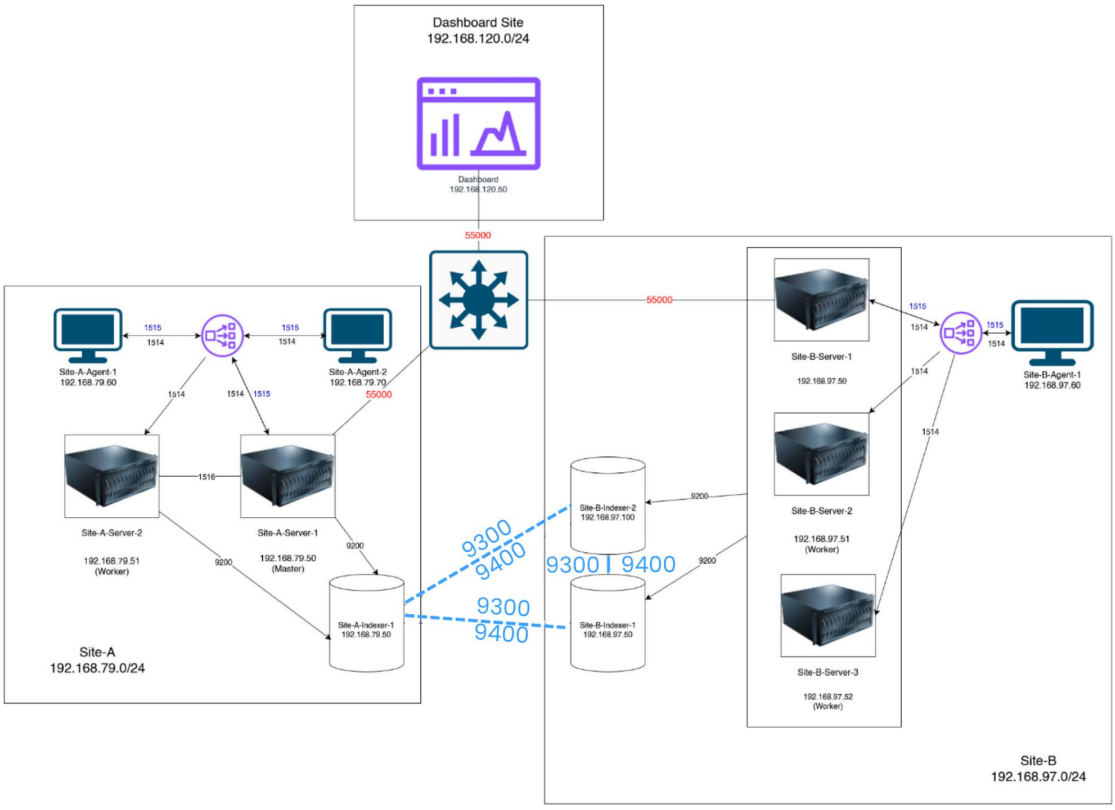

Figure 4.21 Testing Architecture

- Testing

```
root@indexer-b-1:/home/wazuh# curl -k -u admin:admin https://192.168.97.55:9200/_cat/nodes?v
ip            heap.percent ram.percent cpu load_1m load_5m load_15m node.role node.roles
        cluster_manager name
192.168.97.55           36          94   9    0.32    0.43     0.35 dimr      cluster_manager,data,ingest,remote_cluste
r_client *              indexer_b_1
192.168.97.100          41          82  12    0.58    0.60     0.54 dimr      cluster_manager,data,ingest,remote_cluste
r_client -              indexer_b_2
192.168.79.55           27          89  13    0.97    0.70     0.46 dimr      cluster_manager,data,ingest,remote_cluste
r_client -              indexer_a_1
```

Figure 4.22 New indexer node is added to the global cluster

- Result
  - Site A and B's clusters are displayed and viewed independently.

- When site A's servers are down, the agents would not be able to connect to site B's servers.

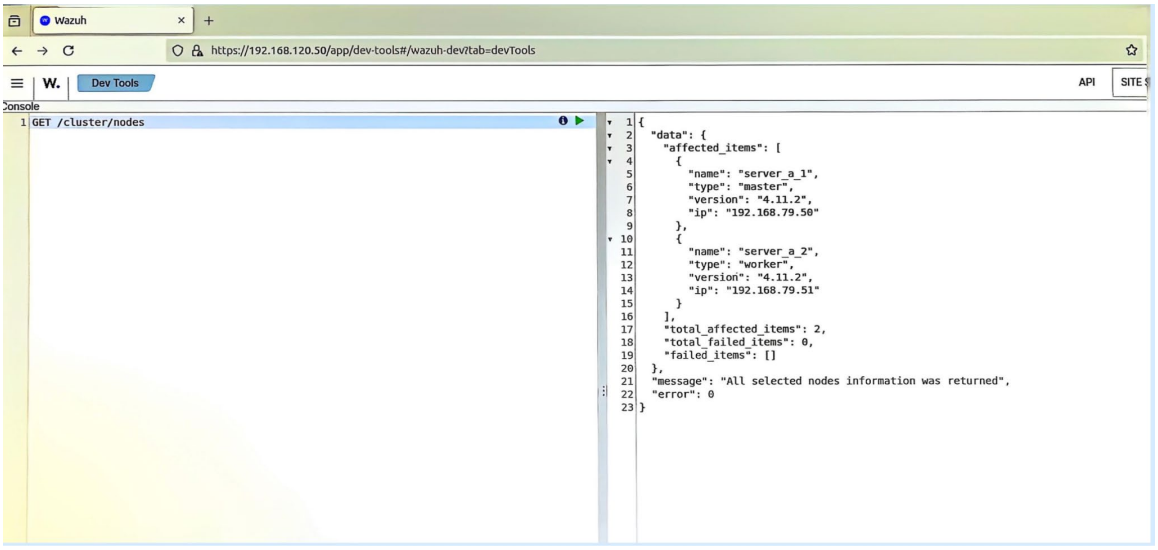

Figure 4.23 Site A's cluster

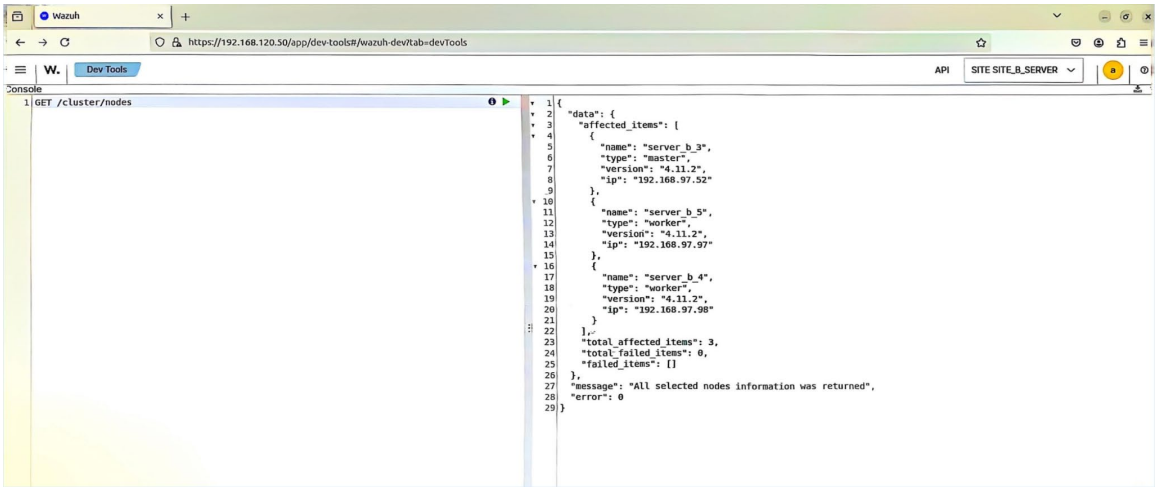

Figure 4.24 Site B's cluster

- Conclusion
  - This test validated the ability to horizontally scale multiple sites in parallel by adding new Wazuh components such as servers, agents, and load balancers without disrupting other sites.

4.2.3.3 New-Site Scalability Testing

- Objective
    - To verify that a new site or multiple sites can be added semi-automatically by modifying the existing Ansible inventory to accommodate the new site.
    - To verify that new sites would not compromise existing components and sites.
- Procedures
    - Similar to the previous steps, new components will be specified in the inventory and the playbook is executed again.

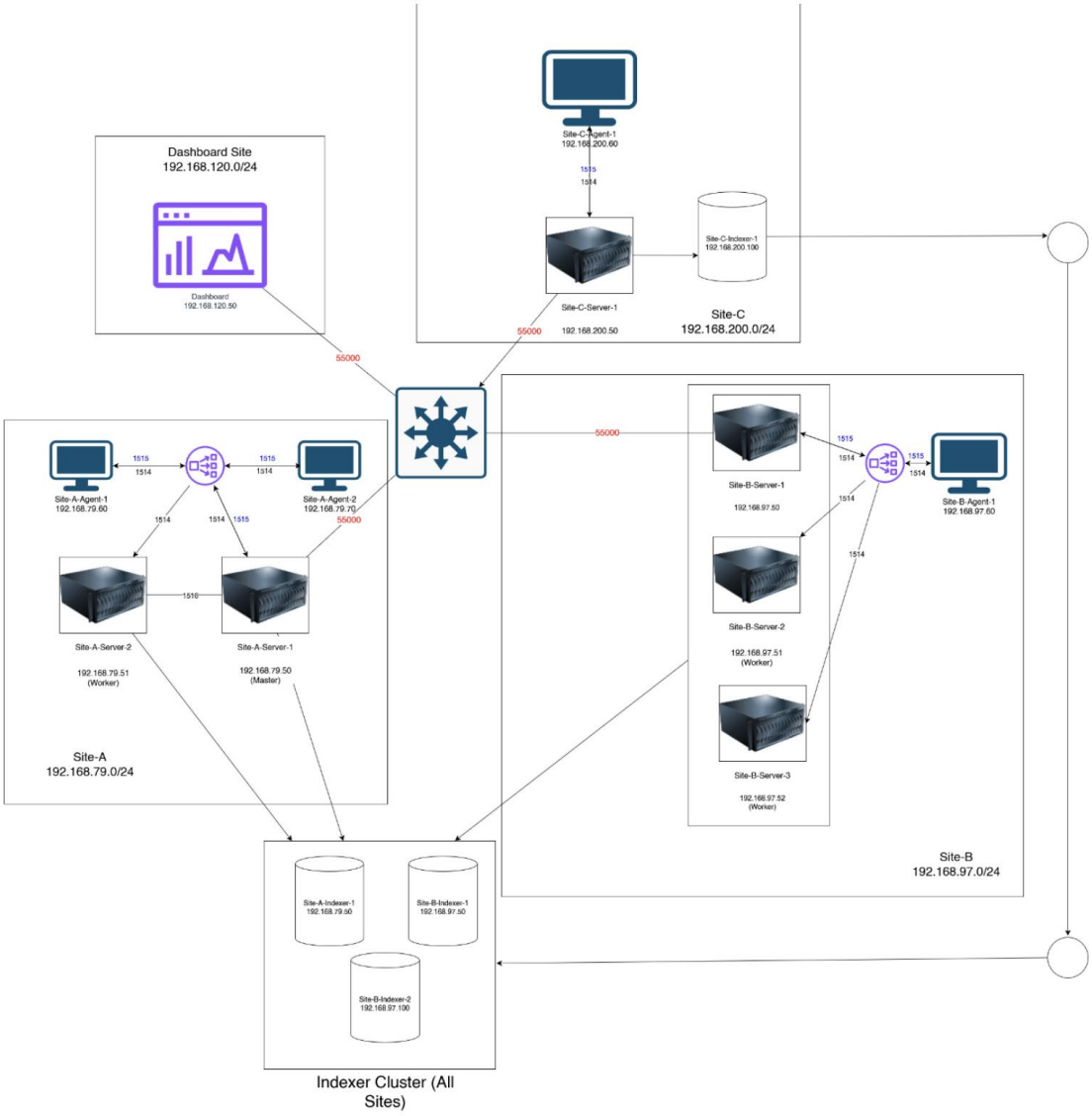

Figure 4.25 Additional site's architecture

- Testing

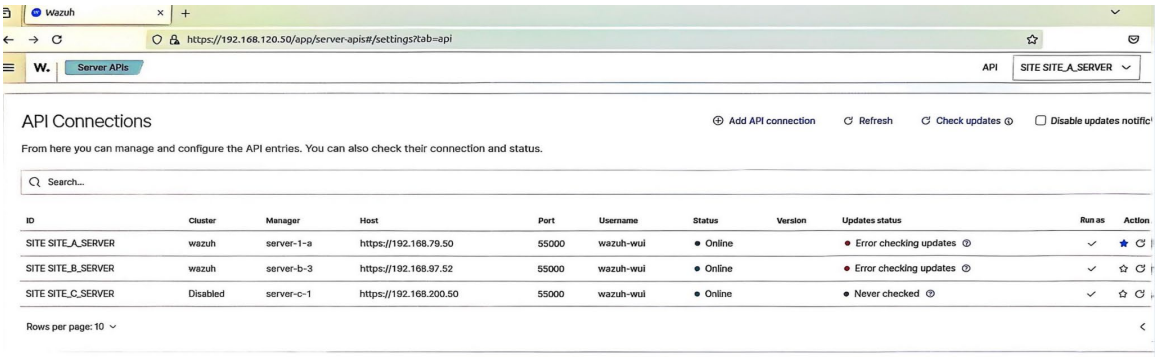

Figure 4.26 New site showing up as an option in the dashboard

- Result
  - An entirely new site is displayed on the dashboard.
  - The existing sites and components are not affected.
- Conclusion
  - The new site was added successfully, and all components (Indexer, Server, Agent) are functioning as intended. Alerts from site C are visible in the central Dashboard, and the site operates in isolation for processing but contributes to the global visibility. The deployment process is reproducible and can be reused for future site additions.

4.2.3.4 Offline Server Alert Testing

- Objective
  - To verify that Wazuh agents can continue monitoring and forwarding logs during a server outage and that once the server is back online, the relevant alerts (including those from the time it was offline) are correctly displayed on the Wazuh Dashboard.
- Procedures
  - Both site A's servers are shut down, not allowing the agents to send alerts to any of the servers. We can verify that the alerts are not displaying by attempting to select site A in the dashboard.

- Using Hydra from an attacker node, an SSH brute force attack is attempted on an agent of site A. During the attack, we can open the dashboard to see that the alerts are not appearing.
- After the attack, opening the dashboard, we can see that the alerts, including during the server's downtime, are now displayed on the dashboard.

- Testing

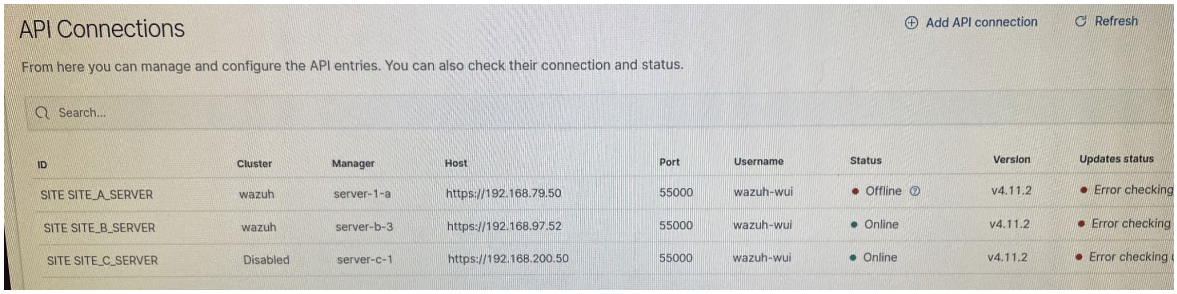

Figure 4.27 Site A's servers are brought offline

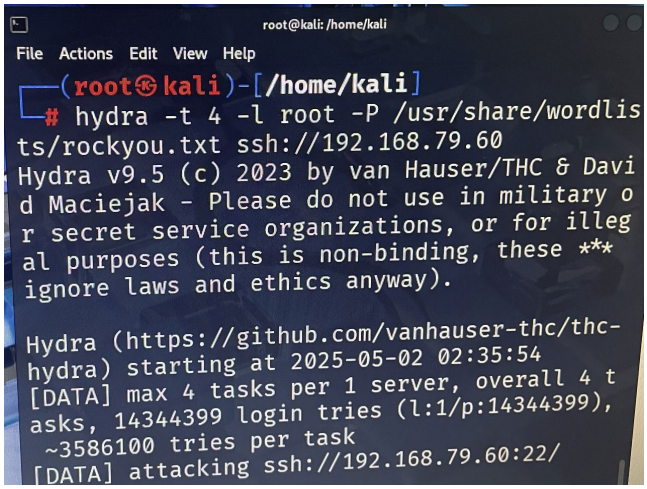

Figure 4.28 An attacker performing SSH on Site A's agent

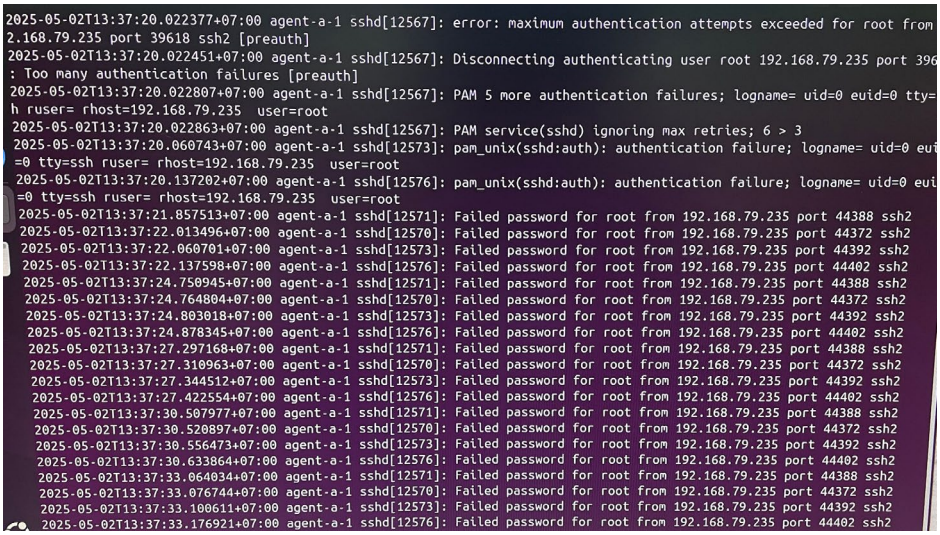

Figure 4.29 Authentication log on agent_a_1 showing repeated SSH failed attempts

- Result
    - As soon as the servers for site A are brought back online, the alerts during the attack are now displayed on the dashboard.

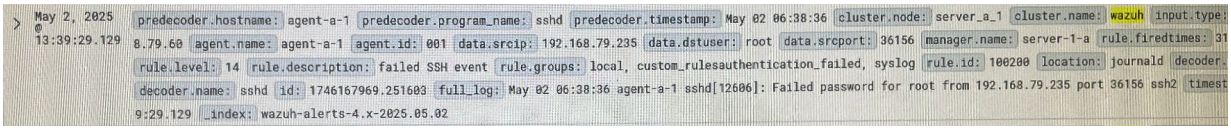

Figure 4.30 The alerts for the attacks happening during the server's downtime are shown.

- Conclusion
    - Despite Wazuh server being required for an alert and events of the agent to be displayed on the dashboard, even if the server is offline during a security event, the agent’s security logs during the server’s downtime will be collected as soon as the server becomes online, which would then be processed and the alerts generated.

- This capability ensures that events happening during server's downtime will not go undetected later, but that real-time monitoring might be temporarily compromised.

CHAPTER 5
CONCLUSION AND FUTURE WORK

5.1 Conclusion

The implemented Wazuh multi-site SIEM solution successfully created a unified security monitoring experience across multiple geographical locations. The feasibility of independent Wazuh SIEM clusters connected under a centralized master dashboard enabled efficient log collection and processing while significantly reducing network congestion. Leveraging open-source tools resulted in substantial cost savings, requiring mainly time and hardware resources for deployment. In summary, this project achieved the following:

- **Decentralize Security Monitoring**
  Design a distributed architecture where each site operates its own Wazuh server, indexer, and agents. This structure enables local log processing and reduces reliance on a central server for real-time analysis.
- **Centralized Visibility Across Sites**
  Integrate a unified Wazuh dashboard that connects to all site-specific servers and indexers through the Wazuh RESTful API. This enables security teams to monitor alerts and events from all locations within a single interface.
- **Minimize Network Congestion**
  Ensure each Wazuh agent connects only to its nearest local server. This setup reduces cross-site data transmission and prevents network bottlenecks.
- **Implement Site-Specific Log and Alert Patterns**
  Configure each Wazuh indexer with a unique log indexing pattern. This helps identify the source of alerts and enhances response accuracy.
- **Ensure Data Redundancy Across Sites**
  Replicate logs and indexed data across all sites to support high availability and effective disaster recovery.

- **Automate Configuration and Deployment**

  Use Ansible playbooks and Jinja2 templates to automate the installation, configuration, and synchronization of Wazuh components across multiple sites.
- **Enhance Cost-Effectiveness and Maintainability**

  Build the system using free and open-source tools such as Wazuh, Ubuntu, and VirtualBox to reduce costs and avoid vendor lock-in. This approach ensures the system remains scalable and easy to maintain.
- **Support System Scalability**

  Apply load balancing and horizontal scaling techniques such as clustering Wazuh servers or adding new agents and indexers to accommodate growth and increased system demands.

However, certain constraints and limitations emerged during implementation, notably in the areas of performance measurement and scalability.

5.2 Discussion

This section summarizes the advantages and limitations identified during the implementation of the multi-site Wazuh SIEM solution.

5.2.1 Advantage

- A centralized Wazuh dashboard aggregates security information from all sites and provides an overview of each site's data.
- The decentralized architecture enables faster recovery from security incidents, improving overall system resilience.
- Network congestion is reduced by allowing Wazuh agents to connect to local servers instead of a centralized server.
- Site-specific alert patterns lead to quicker and more accurate responses.
- The use of open-source tools such as Wazuh, Ubuntu, and VirtualBox results in significant cost savings.
- System performance and scalability are improved, allowing seamless addition of new nodes and sites to meet growing security needs.
- A simplified setup script automates the installation, configuration, and connection of all components.

- A centralized template for Wazuh configuration files enables easy editing and deployment across all sites.

5.2.2 Limitation

- **Manual Recovery for Master Node:** If a site's master server utilized by the dashboard API becomes unavailable, the site temporarily loses dashboard accessibility. The current architecture lacks automatic failover capabilities for master nodes.
- **Scalability Limitations:** Although new components can be integrated through updated playbooks, introducing entirely new sites still necessitates manual edits to the inventory file and re-execution of setup scripts.

Furthermore, challenges arose in accurately measuring critical performance metrics, such as Total Ingestion Time, Network Bandwidth Utilization, Fault Tolerance, and Reliability. Consequently, these metrics were excluded from the evaluation process. Additionally, the manual configuration of rules and alerts, although effective, proved time-consuming and static, limiting proactive response to unknown threats.

5.3 Future Work

Future developments will concentrate on integrating artificial intelligence (AI) and machine learning techniques to automate rule creation and enhance alerting patterns. Employing AI-driven log analysis can detect evolving threat patterns and dynamically generate detection rules without manual intervention, significantly improving response efficiency and accuracy. Moreover, automated prioritization of alerts via machine learning can drastically reduce false positives, allowing security professionals to focus on genuine threats. By evolving into an adaptive, intelligent security system, the Wazuh multi-site SIEM solution has the potential to become a truly next-generation cybersecurity platform, improving scalability, efficacy, and responsiveness to emerging threats.

Appendix A: Virtual Machine and Network Configuration

A.1 Host System Specifications

- Operating System: Windows 10
- Processor: 6 cores
- RAM: 32 GB
- Virtualization Software: Oracle VirtualBox

A.2 Virtual Machines Overview

Site	Component	IP Address	OS Version	RAM	CPU
A	Wazuh server	192.168.79.50	Ubuntu 20.04	3 GB	1
A	Wazuh indexer	192.168.79.55	Ubuntu 20.04	3 GB	1
A	Wazuh Agent	192.168.79.60	Ubuntu 24.04	2 GB	1
B	Wazuh server	192.168.97.50	Ubuntu 20.04	3 GB	1
B	Wazuh indexer	192.168.97.55	Ubuntu 20.04	3 GB	1
B	Wazuh Agent	192.168.97.60	Ubuntu 24.04	2 GB	1
C	Wazuh Dashboard	192.168.120.50	Ubuntu 20.04	3 GB	1
Router	-	Multiple	Ubuntu	1 GB	1

			Server 20.04		

A.3 Router Interfaces

- Site A: 192.168.79.25
- Site B: 192.168.97.25
- Dashboard: 192.168.120.25
- Bridged: 192.168.1.128

Appendix B: Ansible and Jinja2

B.1 Ansible Requirements

• Python must be installed on all devices.

• SSH key-based authentication must be configured between dashboard and other nodes using ssh-copy-id command

B.2 Ansible Playbook Syntax

- The Ansible Syntax uses YAML language for human readability
- Target devices and tasks to be performed are specified in the playbook
- A become flag can be set to run commands as sudoer

```yaml
- name: Install system dependencies on all nodes
  hosts: all
  become: yes
  vars_files:
    - wazuh.yml
  tasks:
    - name: Ensure /etc/resolv.conf exists
      ansible.builtin.file:
        path: /etc/resolv.conf
        state: touch
        owner: root
        group: root
        mode: '0644'
```

Figure B.1 Ansible playbook syntax example

B.3 Ansible Inventory Syntax

- The hosts are listed sequentially and separated into groups.
- Each host has host-specific variables that can be accessed through the playbook or through Jinja2 templates.
    - Some built-in variables include “ansible_host” for the host’s IP address and “ansible_user” for the host’s username.
    - Allows for groups containing subgroups. (Wazuh group containing the Indexers, Servers, and Dashboards groups)

```
# === DASHBOARD ===
[site_dashboard]
dashboard_1 ansible_host=192.168.120.50 ansible_user=wazuh ansible_connection=local ansible_become_password=wazuh

# === SITE A ===
[site_a_indexer]
indexer_a_1 ansible_host=192.168.79.55 ansible_user=wazuh ansible_become_password=wazuh


[site_a_server]
server_a_1 ansible_host=192.168.79.50 master=true cluster_key=53a5e452e4d96a1f48ab2e596e934577 ansible_user=wazuh ansible_become_password=wazuh
server_a_2 ansible_host=192.168.79.51 master=false ansible_user=wazuh ansible_become_password=wazuh
```

Figure B.2 Ansible inventory used

- Allows the use of variables in the template files before those template files are copied to the target devices during Ansible playbook execution.

```
nodes:
{% if 'indexer' in group_names %}
  indexer:
    - name: {{ inventory_hostname }}
      ip: {{ ansible_host }}
{% elif 'server' in group_names %}
  server:
    - name: {{ inventory_hostname }}
      ip: {{ ansible_host }}
{% elif 'dashboard' in group_names %}
  dashboard:
    - name: {{ inventory_hostname }}
      ip: {{ ansible_host }}
{% endif %}
```

Figure B.3 Jinja2 Syntax

- Includes loops, conditional statements, and variables from the inventory.

GitHub Repository of the project: https://github.com/Purinat33/ICTSP-18-Ansible/

Appendix C: List of Packages Installed

Name	Target	Description
openssh-server	All	Provides SSH server daemon (sshd) for secure remote logins and encrypted command execution.
openssh-client	All	Includes command-line tools (ssh, scp, sftp) for secure remote connections, file transfers, and SSH key management.
build-essential	All	Meta-package installing essential compilation tools like GCC, make, and related libraries.
dkms	All	Dynamic Kernel Module Support framework for automatic kernel module rebuilding on kernel updates.
debhelper	All	Collection of programs automating Debian package building tasks, simplifying packaging.
tar	All	Utility for archiving multiple files into a single tarball for backups and software distribution.

curl	All	Command-line tool for data transfer using protocols (HTTP, FTP, etc.), facilitating downloads and uploads.
libcap2-bin	All	Provides utilities for managing POSIX capabilities, enabling fine-grained process privilege control.
Debconf	Wazuh	Debian's configuration management system prompting configuration questions during package installation.
adduser	Wazuh	Provides commands (adduser, deluser) for simplified user and group management with automated directory setup.
procps	Wazuh	Includes utilities (ps, top, vmstat, free) for monitoring and managing system processes and memory usage.
gnupg	Wazuh	Tool for secure communication, data encryption, digital signatures, and encryption key management (OpenPGP standard).

apt-transport-https	Wazuh	Enables APT package manager access to repositories via secure HTTPS connections.
wazuh-indexer	Indexer	Indexes and stores Wazuh server-generated alerts, providing real-time data search and analytics.
wazuh-manager	Server	Collects, analyzes, and responds to security data from monitored endpoints.
filebeat	Server	Lightweight open-source log shipper (Filebeat) for efficient log data collection and forwarding to centralized systems.
wazuh-dashboard	Dashboard	Web-based interface for visualizing, managing, and analyzing Wazuh security data.
lsb-release	Agent	Provides lsb_release utility displaying standardized Linux distribution-specific information.
wazuh-agent	Agent	Lightweight, cross-platform Wazuh endpoint component monitoring systems and forwarding data for analysis.
nginx	Load Balancer	Provides NGINX web server for web content serving, reverse

		proxying, load balancing, and caching.
libnginx-mod-stream	Load Balancer	Provides NGINX Stream module for handling and proxying low-level TCP and UDP traffic.

REFERENCES

[1] Gailly J, Adler M. Components. Wazuh Documentation [Internet]. 2024 Nov 4 [cited 2024 Nov 22]; Available from: https://documentation.wazuh.com/current/getting-started/components/index.html

[2] Stanković S, Gajin S, Petrović R. A Review of Wazuh Tool Capabilities for Detecting Attacks Based on Log Analysis. Proceedings of IX International Conference IcETRAN [Internet]. 2022 Jun 9 [cited 2024 Nov 22]. Available from: https://www.etran.rs/2022/zbornik/ICETRAN-22_radovi/068-RTI2.6.pdf

[3] Jumiaty, Soewito B. SIEM Threat Intelligence Protecting Applications. Proceedings of the International Conference on Computer Applications and Information Security [Internet]. 2024 Sep [cited 2024 Nov 22]; Volume 15, No. 9. Available from: https://ftp.saiconference.com/Downloads/Volume15No9/Paper_23-SIEM_Threat_Intelligence_Protecting_Applications.pdf

[4] Javid H. Wazuh as a SIEM and XDR Solution [Internet]. Spring 2024 [cited 2024 Nov 22]. Available from: https://www.theseus.fi/bitstream/handle/10024/862355/Javid_Hafiz.pdf?sequence=2&isAllowed=y

[5] Majid SMA, Basit AB. Security and Threat Detection through Cloud-Based Wazuh Deployment [Internet]. January 2024 [cited 2024 Nov 22]. Available from: https://www.researchgate.net/publication/379529016_Security_and_Threat_Detection_through_Cloud-Based_Wazuh_Deployment

[6] Kumar R. Understanding Ansible Architecture using diagram [Internet]. 2019 Jul [cited 2025 Jun 24]. Available from: https://www.devopsschool.com/blog/understanding-ansible-architecture-using-diagram/

[7] Wazuh Documentation Team. Wazuh server cluster – architecture overview. Wazuh Documentation [Internet]. 2025 [cited 2025 Jun 7]. Available from: https://documentation.wazuh.com/current/user-manual/wazuh-server-cluster/architecture-

overview.html

[8] Splunk Documentation. Integrate the search head cluster with an indexer cluster. Splunk Docs [Internet]. 2024 [cited 2025 Jun 7]. Available from: https://docs.splunk.com/Documentation/Splunk/9.4.2/DistSearch/SHCandindexercluster

[9] Hannachi H. Elastic — Elastic SIEM fundamentals. Medium [Internet]. 2024 Apr 12 [cited 2025 Jun 7]; Available from: https://hassen-hannachi.medium.com/elastic-elastic-siem-fundamentals-3337d580fafe

[10] Elastic. Clusters, nodes and shards. Elastic Docs [Internet]. 2024 [cited 2025 Jun 7]. Available from: https://www.elastic.co/docs/deploy-manage/distributed-architecture/clusters-nodes-shards

[11] Microsoft. What is Microsoft Sentinel? Microsoft Learn [Internet]. 2024 [cited 2025 Jun 7]. Available from: https://learn.microsoft.com/azure/sentinel/overview

[12] IBM Corporation. QRadar architecture overview. IBM Docs [Internet]. 2023 [cited 2025 Jun 7]. Available from: https://www.ibm.com/docs/en/qsip/7.4?topic=deployment-qradar-architecture-overview

[13] OpenText. Micro Focus ArcSight ESM technical overview. OpenText Community [Internet]. 2021 [cited 2025 Jun 7]. Available from: https://community.microfocus.com/cfs-file/__key/telligent-evolution-components-attachments/00-224-00-00-00-19-72-54/ESM_7.3_Binder.pdf

[14] LogRhythm. Understand the LogRhythm architecture. LogRhythm Docs [Internet]. 2024 [cited 2025 Jun 7]. Available from: https://docs.logrhythm.com/lrsiem/docs/understand-the-logrhythm-architecture

[15] UnderDefense. Splunk SIEM pricing guide 2025. UnderDefense Blog [Internet]. 2025 [cited 2025 Jun 7]. Available from: https://underdefense.com/industry-pricings/splunk-siem-pricing

[16] Microsoft Azure. Microsoft Sentinel pricing. Azure Pricing [Internet]. 2025 [cited 2025 Jun 7]. Available from: https://azure.microsoft.com/pricing/details/microsoft-sentinel/

[17] Elastic. Pricing FAQ. Elastic.co [Internet]. 2025 [cited 2025 Jun 7]. Available

from: https://www.elastic.co/pricing/faq

[18] Wazuh. Open-source XDR and SIEM platform overview. Wazuh.com [Internet]. 2025 [cited 2025 Jun 7]. Available from: https://wazuh.com/

[19] LogRhythm. Understand the LogRhythm Architecture. docs.logrhythm.com [Internet]. 2025 [cited 2025 Jul 4]. Available from: https://docs.logrhythm.com/lrsiem/docs/understand-the-logrhythm-architecture