Journal of Applied Informatics and Computing (JAIC)
Vol.9, No.5, October 2025, pp. 2764~2774
e-ISSN: 2548-6861

Adaptive File Integrity Monitoring for Container Virtualization Environments using OSSEC with Real-Time Alerting

Gerry Italiano Wowiling [1]*, Eka Stephani Sinambela [2]*, Frengki Simatupang [3]*, Fabert Jody Manuel Siagian [4]*, Aisyah Ayu Sibarani [5]*, Indah Sari Batubara [6]*
* DIII Teknologi Komputer, Institut Teknologi Del
gerry@del.ac.id[1] , eka@del.ac.id[2] , frengki.simatupang@del.ac.id[3] , ce322004@students.del.ac.id [4], ce322046@students.del.ac.id [5], ce322062@students.del.ac.id [6]

Article Info

Article history:

Received 2025-06-30
Revised 2025-07-30
Accepted 2025-08-10

Keyword:

Container,
Docker,
File Integrity Monitoring,
OSSEC,
Virtualization.

ABSTRACT

In this ever-evolving digital age, container technology has become one of the main solutions in cloud computing due to its efficiency and flexibility. However, the dynamic and ephemeral nature of containers poses new challenges in terms of security, especially regarding data integrity. The implementation of OSSEC in container environments requires a tailored approach, as it lacks native support for automatically detecting new containers. Agents must be embedded within container images or installed at the host level. These agents activate each time a container runs and send monitoring data to the OSSEC server. With orchestration and automated configuration, monitoring results are stored externally, and real-time email alerts can be triggered upon detecting suspicious file changes. Container environments are increasingly targeted by cyber threats such as malware and ransomware, which pose risks of unauthorized data access or encryption. Limited file integrity monitoring within containers creates a security gap that can be exploited undetected. This research addresses the issue by implementing a File Integrity Monitoring (FIM) mechanism using OSSEC, an open-source Host Intrusion Detection System (HIDS) capable of real-time file and log monitoring, malware detection, and automated threat response. OSSEC is deployed within a Docker-based setup and integrated with a Web User Interface for visualizing logs and monitoring activity. The system includes real-time email notifications for immediate alerts. Testing through file modification scenarios confirmed OSSEC's accuracy in detecting changes and notifying administrators. This implementation effectively strengthens data security and provides timely threat detection in containerized environments.

I. INTRODUCTION

The use of containerization technologies such as Docker and Kubernetes has revolutionized today's software development process[1][2][3]. These technologies, by packaging applications and their dependencies into portable and isolated units, have significantly enhanced efficiency and scalability[4]. However, container environments' ephemeral and dynamic nature also presents security challenges[5][6]. Conventional security approaches often struggle to keep up with containerized systems' rapid creation and removal, leading to potential security gaps. These gaps, such as unauthorized configuration changes or the insertion of undetected malicious code, can be difficult to identify[7].

This research is dedicated to implementing a robust File Integrity Monitoring (FIM) system using OSSEC, an open-source Host-based Intrusion Detection System (HIDS)[8][9]. OSSEC is one of the best open-source tools because it can detect attacks in real time. So Snort's performance as an HIDS has many shortcomings, and Tripware is too simple. Tripware only detects file changes on the host, or in other words, the rules provided by Tripware are limited. Additionally, another open-source solution, Fail2Ban, has limitations because it does not support centralized log management and lacks important features such as file integrity checks, registry monitoring, rootkit detection, and event correlation. Unlike Fail2Ban, OSSEC offers a broader range of features, including the ability to detect attacks in real-time, making it one of the best open-source solutions for system security. Furthermore, recent studies have implemented Wazuh—an advanced fork of OSSEC—for intrusion prevention and malware containment in modern server environments, reinforcing its role in proactive threat management[10]. OSSEC is one of the best open source tools because it can detect attacks in real time. So Snort's performance as an HIDS has many shortcomings, and Tripware is too simple. Tripware only detects file changes on the host, or in other words, the rules provided by Tripware are limited[11]. Additionally, another open-source solution, Fail2Ban, has limitations because it does not support centralized log management and lacks important features such as file integrity checks, registry monitoring, rootkit detection, and event correlation. Unlike Fail2Ban, OSSEC offers a broader range of features, including the ability to detect attacks in real-time, making it one of the best open-source solutions for system security[12]. FIM, a proactive approach, plays a pivotal role in ensuring the integrity of files in operating systems and applications by comparing them against a previously verified baseline, a fundamental method in the security of virtualization environments[13][14][15]. As noted by Mane et al., File Integrity Monitoring is a crucial pillar of modern cybersecurity frameworks, enabling early detection of unauthorized file manipulations across various platforms[13]. OSSEC, chosen for its ability to perform real-time monitoring and provide centralized log analysis, is essential for detecting suspicious activity in distributed environments[16]. The file integrity monitoring system using OSSEC can maintain its monitoring function even if the container is temporary, stopped, or restarted, as long as the deployment architecture is designed appropriately[14]. This is possible because OSSEC centrally stores monitoring results, file baselines, and logs on the OSSEC server running outside the container. In this way, monitoring data is not dependent on containers that can be deleted at any time. Additionally, the OSSEC agent embedded in the container image will automatically restart when the container is restarted and immediately connect to the OSSEC server to continue the monitoring process[17]. Alternatively, the agent can be run at the host level and utilize volume mounting so that container system files can still be monitored even if the container is stopped or restarted. With this approach, data integrity can be maintained, and any suspicious file changes can still be detected and reported via real-time email notifications. This proactive monitoring approach is conceptually in line with various stochastic models commonly used to manage complex and dynamic systems, such as in modeling congestion control in the TCP protocol[18]. With the FIM system, any unauthorized changes to binary files, configuration files, or critical directories can be immediately detected and reported, enhancing the security of containerized environments.

Conventional security approaches often struggle to keep up with the speed of container creation and deletion, opening up vulnerabilities such as unauthorized configuration changes or the insertion of malicious code that is difficult to detect. Previous studies have addressed container security issues in general, but most have focused on network security, authentication, or log auditing, while research specifically exploring the implementation of File Integrity Monitoring (FIM) based on OSSEC in container environments remains limited. This research gap is the focus of this study. This study aims to design, implement, and evaluate the performance of OSSEC as an FIM solution in a dynamic container environment, with a focus on agent configuration, the effectiveness of directory monitoring within containers, and the ability to detect unauthorized file changes. With this approach, it is hoped that a practical framework can be developed that can be reapplied by system administrators to strengthen the protection of container-based infrastructure and minimize security risks.

One of the main challenges in implementing File Integrity Monitoring (FIM) in a containerized environment is the potential overhead and complexity in managing the monitoring agents[11]. Ideally, each container requires monitoring, but without proper management, this can lead to excessive consumption of host resources. This situation underscores the critical importance of designing an efficient monitoring architecture, where agent configuration can be done centrally while running optimally within the container. Without a structured and automated approach, security teams risk falling back on manual methods that are not only time-consuming but also prone to errors, especially in large-scale environments with hundreds to thousands of concurrently active containers. Therefore, a structured and automated approach to FIM is crucial for maintaining the security of containerized environments.

Previous studies have discussed security measures in container environments, but they generally focus on vulnerability scanning, log auditing, or network security. Not many studies have specifically applied File Integrity Monitoring (FIM) to detect file changes within containers. Additionally, most FIM approaches that have been studied have not utilized OSSEC as an intrusion detection system, despite OSSEC's advantages in real-time file monitoring. Previous research has also generally not been equipped with automatic notification mechanisms via email when file changes or suspicious attacks are detected. This often results in potential attacks being detected too late by users. Recent enhancements to OSSEC also include advanced log correlation capabilities and mechanisms to override false positives or negatives, significantly improving its detection accuracy in high-noise environments[16]

OSSEC emerges as a potent solution to the challenge of file integrity monitoring in containerized environments[19]. As open-source software, OSSEC offers unparalleled flexibility in customization, a feature often lacking in commercial solutions. Its key features, including real-time file change monitoring, checksum checking, and validation of file access rights, make it a robust tool for maintaining system integrity[12]. Its reliability can be likened to that of a high-resolution distributed measurement system capable of accurately collecting data from multiple points simultaneously. With its agent-based architecture, OSSEC enables detailed monitoring at both the host and container levels while its central server aggregates all alerts for further analysis[20]. This approach aligns with best practices outlined in various technical reports and industry standards, making OSSEC a compelling choice for container security.

This research aims to design, implement, and assess the performance of OSSEC as a File Integrity Monitoring (FIM) solution in a dynamic container environment. The primary focus of the research is on the process of configuring the OSSEC agent to monitor directories in containers, as well as testing its ability to identify various types of changes that occur. The expected outcome of this research is the development of a practical framework that can be re-implemented by system administrators and security professionals to strengthen the protection of container-based infrastructure and minimize the risk of unauthorized or undetected file changes.

II. Proposed Method

The method used in this final project includes the stages of system requirements analysis, architecture design, implementation, and testing. The system design process is described in detail in the following sub-chapters to provide a thorough understanding of the developed system's working mechanism.

A. *System Architecture Design*

Using a client-server architecture, this system is designed to centrally monitor file integrity in a Docker container environment. The OSSEC Agent acts as a client embedded in each container, while the OSSEC Server serves as the central data collector and analyzer. In its implementation, the OSSEC Agent is run directly inside each target container it wants to monitor, rather than as a separate container or sidecar. This approach allows the monitoring process to be performed locally by the agent within the container itself, eliminating the need for volume sharing mechanisms or external access to the container file system. The system architecture design of 'File Integrity Monitoring in Container Environments using OSSEC' can be seen in Figure 1.1

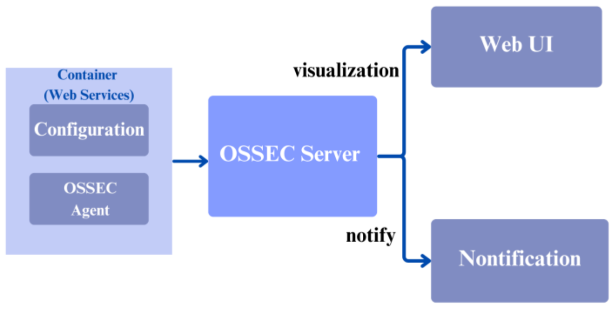


Figure 1. System Architecture Design

Based on the diagram, each container that you want to monitor is equipped with the OSSEC Agent. This agent is tasked with monitoring any changes to files or directories that have been specified within the container, such as the creation of new files, modifications, or file deletions. If a change is detected, the agent will create an event log and send it to the OSSEC Server via the network, using UDP port 1514 secured with a unique authentication key for each agent.

1) *Container with OSSEC Agent*

Any container that you want to monitor (e.g., running a web service) will be equipped with an OSSEC Agent. This agent is responsible for monitoring any file changes to the specified directory within the container.

2) *Log Transmission to Server*

Once the agent detects a change, it immediately creates a log or record of the event related to that activity. This log is then sent to the OSSEC Server via the network, using a secure communication protocol. The log transmission process uses UDP port 1514, which is secured with a unique authentication key to ensure that communication between the agent and the server remains encrypted and verified.

3) *Analysis by OSSEC Server*

The OSSEC Server functions as a data processing center. The server receives log data from all connected agents, then processes and analyzes it using preconfigured rules. Through this analysis, the server can determine whether the file change is normal activity (e.g., routine updates) or an indication of suspicious activity that could potentially pose a security threat.

4) *Visualization and Notification*

The analysis results from the server are forwarded to two main destinations:

- Web UI: All detection and analysis data is visualized through the OSSEC Web UI in the form of informative and easy-to-understand dashboards. These dashboards allow administrators to monitor file integrity status in real time, check alert details, and review activity history more efficiently.

- Email Notification: In addition to visualization, if the system detects changes categorized as warnings with a certain level, the OSSEC Server will automatically send notifications to the administrator's email address. With these notifications, administrators can respond to incidents immediately without having to constantly monitor the dashboard, thereby supporting a fast and accurate security response.

B. *Topology and Communication Design*

In order for the agent and server to communicate, the system is implemented on a virtual network topology where each component has a unique IP address. Communication between the agents and the server is secured by using an authentication key that is generated at the time of agent registration. In this implementation, the OSSEC Server is run as a separate container that serves as the central hub for receiving logs from all agents. Meanwhile, each target container runs the OSSEC Agent independently within it without using a volume sharing mechanism, as file monitoring is done directly by the agents within the container itself.

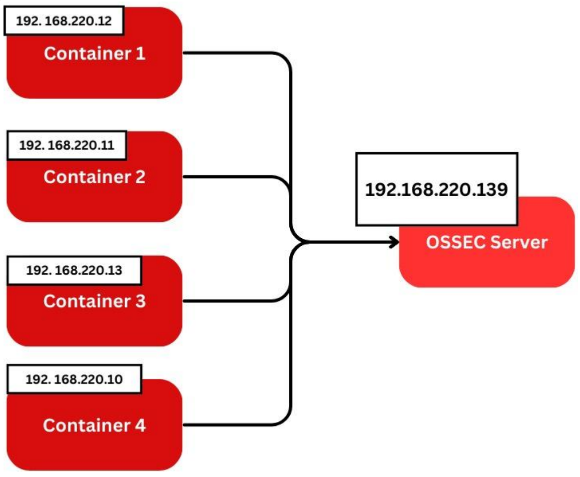
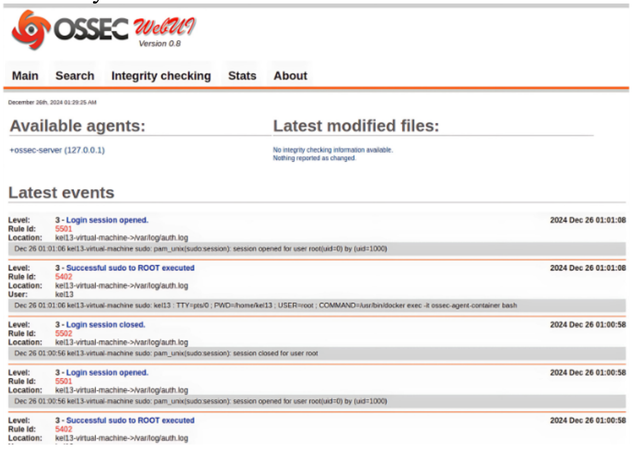


Figure 2. Logical Topology

As shown in the Figure above, each container (for example, Container 1 with IP 192.168.220.12) is configured to send logs to one central OSSEC Server (with IP 192.168.220.139). All this communication happens over standard network protocols, where OSSEC, by default, uses UDP port 1514 to receive data from agents.

C. *User Interface*

The user interface of the system was implemented using the OSSEC Web User Interface (WUI), which presents monitoring data in an informative dashboard format. The OSSEC WUI used is a standard interface built by the OSSEC community, not the result of integration with external tools such as Wazuh or other custom visualizations. The choice of this interface was based on its ease of installation, simplicity of appearance, and direct compatibility with the OSSEC system used in this study.



```
December 26th, 2024 01:29:25 AM

Available agents:                     Latest modified files:
+ossec-server (127.0.0.1)             No integrity checking information available.
                                      Nothing reported as changed.

Latest events

Level:     3 - Login session opened.                                   2024 Dec 26 01:01:08
Rule Id:   5501
Location:  kel13-virtual-machine->/var/log/auth.log
  Dec 26 01:01:06 kel13-virtual-machine sudo: pam_unix(sudo:session): session opened for user root(uid=0) by (uid=1000)

Level:     3 - Successful sudo to ROOT executed                        2024 Dec 26 01:01:08
Rule Id:   5402
Location:  kel13-virtual-machine->/var/log/auth.log
User:      kel13
  Dec 26 01:01:06 kel13-virtual-machine sudo: kel13 : TTY=pts/0 ; PWD=/home/kel13 ; USER=root ; COMMAND=/usr/bin/docker exec -it ossec-agent-container bash

Level:     3 - Login session closed.                                   2024 Dec 26 01:00:58
Rule Id:   5502
Location:  kel13-virtual-machine->/var/log/auth.log
  Dec 26 01:00:56 kel13-virtual-machine sudo: pam_unix(sudo:session): session closed for user root

Level:     3 - Login session opened.                                   2024 Dec 26 01:00:58
Rule Id:   5501
Location:  kel13-virtual-machine->/var/log/auth.log
  Dec 26 01:00:56 kel13-virtual-machine sudo: pam_unix(sudo:session): session opened for user root(uid=0) by (uid=1000)

Level:     3 - Successful sudo to ROOT executed                        2024 Dec 26 01:00:58
Rule Id:   5402
Location:  kel13-virtual-machine->/var/log/auth.log
```

Figure 3. OSSEC User Interface Display

Based on the figure above, the main dashboard is divided into several main sections:

1) *Navigation Menu*:
   It contains links to the main page (Main), search (Search), integrity checking (Integrity Checking), statistics (Stats), and information (About).
2) *Main Content Area:*
   - Available agents: Displays a list of all agents that are active and connected to the server.
   - Latest Modified Files: Provides a summary of recently modified files.
   - Latest events: Displays a real-time log of recent events, complete with description, severity, and time of occurrence.

D. *System Flowchart*

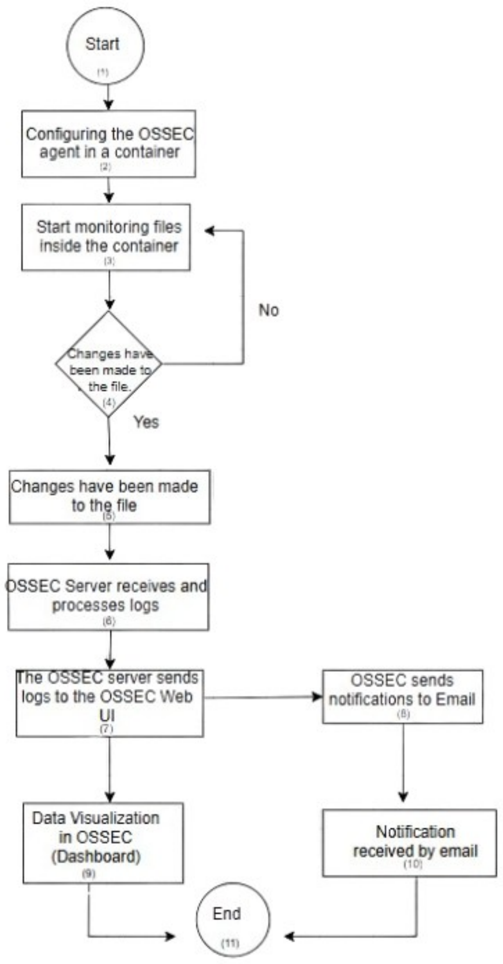

Start
(1)
Configuring the OSSEC agent in a container
(2)
Start monitoring files inside the container
(3)
Changes have been made to the file.
(4)
No
Yes
Changes have been made to the file
(5)
OSSEC Server receives and processes logs
(6)
The OSSEC server sends logs to the OSSEC Web UI
(7)
OSSEC sends notifications to Email
(8)
Data Visualization in OSSEC (Dashboard)
(9)
Notification received by email
(10)
End
(11)

Figure 4. Flowchart of the System

The workflow of the monitoring system from start to finish is depicted in the figure above. The process is as follows:

- Start: The monitoring process initiates when the system starts. This marks the beginning of the file integrity monitoring cycle using OSSEC in a containerized environment.
- Configuring the OSSEC Agent in a Container: Each container that needs to be monitored is configured with an OSSEC agent. This agent is responsible for observing specified files or directories for any integrity violations.
- Start Monitoring Files inside the Container: Once the OSSEC agent is configured, it begins actively monitoring the specified files within the container for any changes such as creation, deletion, or modification.
- Changes Have Been Made to the File? :The OSSEC agent continuously checks for file integrity. If no changes are detected, it loops back and continues monitoring. If any modification is found, it proceeds to the next step.
- Changes Have Been Made to the File: Upon detecting a change (e.g., a file is edited, added, or deleted), the agent logs this event as a potential security concern.
- OSSEC Server Receives and Processes Logs: The logs generated by the agent are forwarded to the OSSEC server, which processes these using predefined rules. These logs typically include details such as the filename, change type, and timestamp.
- The OSSEC Server Sends Logs to the OSSEC Web UI: After processing, the logs are made available on the OSSEC Web User Interface (UI) for further analysis. This allows administrators to visually track system events in real-time.
- OSSEC Sends Notifications to Email: If the change meets a specific alert level threshold, the OSSEC server generates an email notification. This email includes relevant log data to inform administrators immediately.
- Notification Received by Email: The email is delivered to the designated recipient, enabling real-time awareness and prompt response to potential integrity violations.
- Data Visualization in OSSEC (Dashboard): Administrators can view summarized and detailed event data through the OSSEC dashboard. This visualization helps in auditing, analyzing trends, and verifying system health.
- End: This step marks the end of one detection cycle. However, the OSSEC agent continues to monitor files in real-time, ensuring ongoing integrity checks and alerting mechanisms remain active.

III. Result and Discussion

This chapter describes the results of the implementation and series of tests conducted on the File Integrity Monitoring (FIM) system in a container environment by utilizing OSSEC. The discussion includes verification of key functions, such as connectivity between agents and servers, integration of notification systems, and analysis of various test scenarios to measure the capabilities of the system. The focus of the presentation of the test results is on one container, Container-A. This approach was chosen because the four monitored containers have similar OSSEC agent configurations and environments, so the results from one container can be used as a representation of the entire system.

A. System Implementation Verification

1) Agent and Server Connectivity

Agent and Server Connectivity in the OSSEC system describes the relationship between the OSSEC Agent installed in the container (A–D) and the OSSEC Server as the monitoring center, as shown in the figure below.

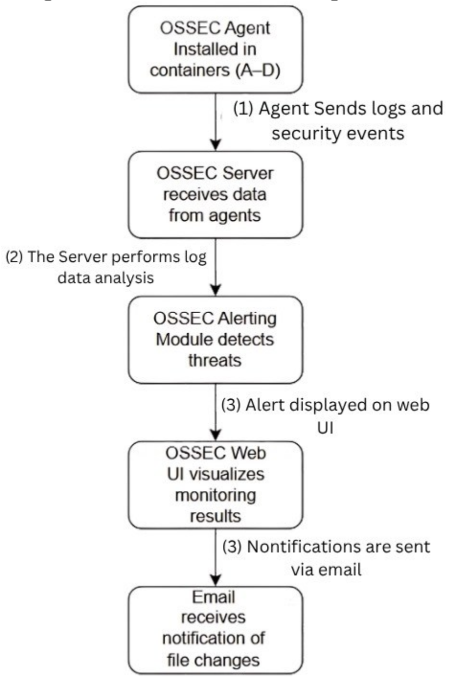
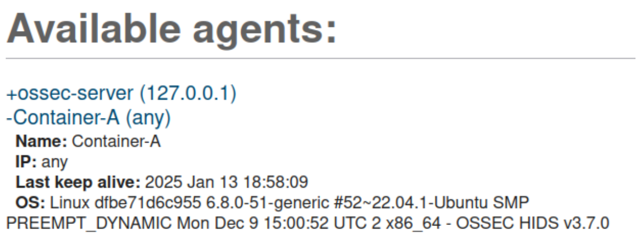

OSSEC Agent Installed in containers (A–D)
(1) Agent Sends logs and security events
OSSEC Server receives data from agents
(2) The Server performs log data analysis
OSSEC Alerting Module detects threats
(3) Alert displayed on web UI
OSSEC Web UI visualizes monitoring results
(3) Nontifications are sent via email
Email receives notification of file changes

Figure 5. Flowchart Agent and Server Connectivity

In the figure above, the process begins when the OSSEC Agent sends logs and security events to the OSSEC Server. After receiving the data, the server analyzes the incoming logs to detect suspicious activity or security threats. This analysis process is performed by the OSSEC Alerting module, which automatically identifies potential threats based on predefined rules. If a threat is detected, the system displays an alert via the OSSEC Web UI so that it can be visually monitored by the administrator. Additionally, notifications are sent to the configured email address to inform the administrator about file changes or activities deemed suspicious. This workflow ensures that all critical activities within the system can be monitored in real-time and addressed promptly.


Available agents:

+ossec-server (127.0.0.1)
-Container-A (any)
Name: Container-A
IP: any
Last keep alive: 2025 Jan 13 18:58:09
OS: Linux dfbe71d6c955 6.8.0-51-generic #52~22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Mon Dec 9 15:00:52 UTC 2 x86_64 - OSSEC HIDS v3.7.0

Figure 6. Agent and Server Connectivity

The image above shows the OSSEC Web UI dashboard in the “Available agents” section. In the list, it can be seen that the agent with the name Container-A was successfully registered and recognized by the server. Evidence of strong connectivity is shown by the “Last kept alive” information with a time of 2025 Jan 13 18:58:09, which indicates that the server has just received a “heartbeat” signal from the agent. In addition, the server also successfully retrieved operating system details from the agent, confirming functional communication.

This connectivity is further reinforced by the specific event log shown in Figure below. The figure shows a Level 3 alert with the message “Ossec agent started.” logged at time 2025 Jan 13 18:57:53. The “Location” field containing *(Container-A) any->ossec* explicitly confirms that this log originated from Container-A and was successfully received and processed by the server.

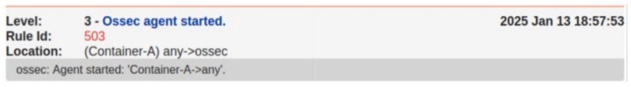

Level: 3 - Ossec agent started. 2025 Jan 13 18:57:53
Rule Id: 503
Location: (Container-A) any->ossec
ossec: Agent started: 'Container-A->any'.

Figure 7. OSSEC Agent Started

In general, the two figures support each other in proving that the connection between the agent running in the container and the central server has been successfully established properly, is stable, and is ready to be used for further monitoring data transmission.

2) OSSEC WUI (Web User Interface)

OSSEC WUI (Web User Interface) is a web-based interface used to display data from OSSEC Server monitoring results in a visual flowchart. The process can be seen in the image below.

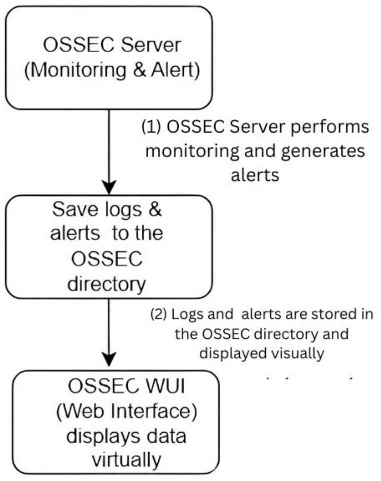

OSSEC Server (Monitoring & Alert)
(1) OSSEC Server performs monitoring and generates alerts
Save logs & alerts to the OSSEC directory
(2) Logs and alerts are stored in the OSSEC directory and displayed visually
OSSEC WUI (Web Interface) displays data virtually

Figure 8. Flowchart OSSEC Web User Interface

In the image above, you can see the OSSEC Server flowchart, which is responsible for monitoring system activity and generating alerts when suspicious events or violations of predetermined rules are detected. Logs and alerts generated by the OSSEC Server are stored in the OSSEC directory. Subsequently, OSSEC WUI reads and displays this data in an easy-to-understand visual format via a web browser. With OSSEC WUI, administrators can quickly monitor system security status, view alert history, and take necessary actions without having to manually access log files.

3) Email Nontification Integration

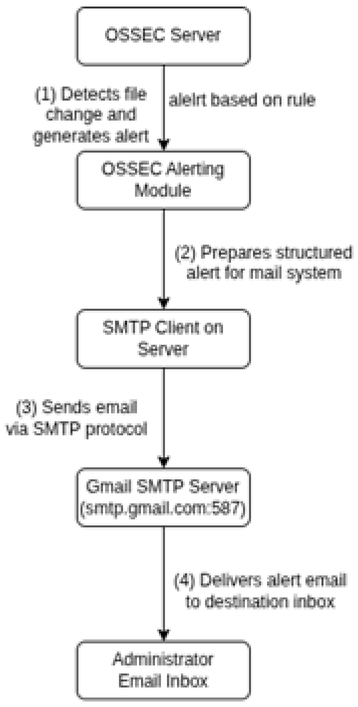

OSSEC Server
(1) Detects file change and generates alert
aleirt based on rule
OSSEC Alerting Module
(2) Prepares structured alert for mail system
SMTP Client on Server
(3) Sends email via SMTP protocol
Gmail SMTP Server (smtp.gmail.com:587)
(4) Delivers alert email to destination inbox
Administrator Email Inbox

Figure 9. Flowchart Email Nontification Integration

When OSSEC detects a rule-based event, such as the start of a service or a file modification, it generates an alert and sends it to the alerting module. This module is configured to forward the alert using an SMTP client (e.g., Postfix or Sendmail). The email message is then transmitted to Gmail's SMTP server using the SMTP protocol. This flow is illustrated in the communication diagram above. The successful delivery of alerts to the administrator's inbox confirms that the email integration has been correctly configured and is functioning reliably. This ensures that any critical activity within the monitored container environment can be immediately brought to the attention of the security team.

B. FIM Scenario Testing Results

FIM scenario testing was conducted to validate the system's ability to detect various types of file integrity changes accurately and in real-time. The test scenarios included detection of new file creation, file content modification, file deletion, as well as testing active response to simulated attacks. The results of the event logs displayed on the Web UI and the recapitulation of email notifications from all these test scenarios are summarized in Table below.

TABLE I

EVENT LOG RESULTS ON OSSEC WEB UI

Timestamp	Level	Rule ID	Location	Event Description
2025 Jan 14 7:21:54	3	5402	k13-virtual-machine->/var/log/auth.log	"Successful sudo to ROOT executed" by user k13 for command sudo nano testingfile.txt
2025 Jan 14 7:21:54	7	554	k13-virtual-machine->syscheck	"File added to the system." Detected file: /etc/testingfile.txt.swp
2025 Jan 14 7:27:13	7	550	k13-virtual-machine->syscheck	"Integrity checksum changed" for file /etc/testingmodifikasi.c
2025 Jan 14 8:32:40	7	553	k13-virtual-machine->syscheck	"File deleted. Unable to retrieve checksum." for file /etc/testingmodifikasi.c
2025 Jan 14 3:25:18	5	2503	k13-virtual-machine->/var/log/auth.log	"Connection blocked by Tcp Wrappers." for Src IP: 192.168.220.1

TABLE II

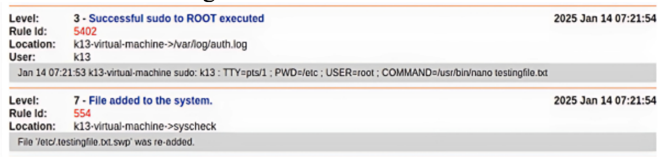
EMAIL NOTIFICATION SUMMARY

Datetime	File Path / Change	Additional Information	Change Status (Legal / Illegal)
2025 Jan 14 7:21:54	/etc/testingfile.txt.swp	File added to the system. Level 7 alert triggered by syscheck.	Legal
2025 Jan 14 7:27:13	/etc/testingmodifikasi.c	Integrity checksum changed. File size changed from '85' to '90'. Content changed on line 5 from printf("Hello, World!\n"); to printf("Testing Modifikasi\n");.	Legal
2025 Jan 14 8:32:40	/etc/testingfile.txt	File deleted. The system could no	Legal

	& /etc/testingmodifikasi.c	longer retrieve the checksum. Two separate alerts were received for each deleted file.	
2025 Jan 14 3:25:18	IP Block	Connection blocked. IP 192.168.220.1 was blocked as an active response to a simulated brute-force attack.	Illegal

1) *File Creation Detection Test*

This test scenario aims to validate the system's ability to detect the creation of new files in the monitored directory. Testing is done by creating a new file called testingfile.txt in the /etc directory using the nano editor with sudo access rights.

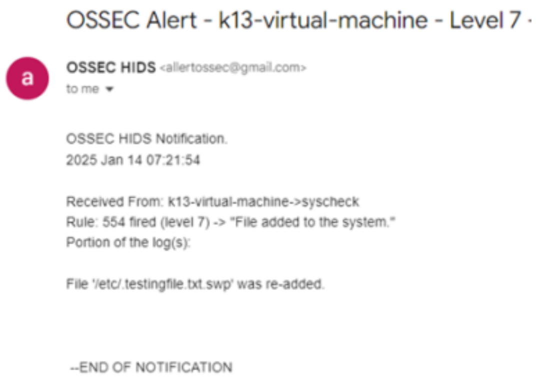


```
Level:      3 - Successful sudo to ROOT executed                    2025 Jan 14 07:21:54
Rule Id:    5402
Location:   k13-virtual-machine->/var/log/auth.log
User:       k13
 Jan 14 07:21:53 k13-virtual-machine sudo: k13 : TTY=pts/1 ; PWD=/etc ; USER=root ; COMMAND=/usr/bin/nano testingfile.txt

Level:      7 - File added to the system.                           2025 Jan 14 07:21:54
Rule Id:    554
Location:   k13-virtual-machine->syscheck
 File '/etc/.testingfile.txt.swp' was re-added.
```

Figure 10. Detection results that appear on the Web UI dashboard

The image above shows the detection results that appear on the Web UI dashboard. There are two relevant alerts recorded in sequence:

- Contextual Alerts (Level 3): The first alert with the description “Successful sudo to ROOT executed” is an audit log noting that user k13 has executed the command sudo nano testingfile.txt at 07:21:54. This alert provides forensic context as to who and how the file was created.
- FIM Alert (Level 7): The second warning with the description “File added to the system.” is a direct result of File Integrity Monitoring. The syscheck module detects that a new file (/etc/testingfile.txt.swp, a temporary file from the nano editor) has been added to the system. This alert has a higher level of 7 because it indicates a direct change to the integrity of the file system.



```
OSSEC Alert - k13-virtual-machine - Level 7 -

OSSEC HIDS <allertossec@gmail.com>
to me

OSSEC HIDS Notification.
2025 Jan 14 07:21:54

Received From: k13-virtual-machine->syscheck
Rule: 554 fired (level 7) -> "File added to the system."
Portion of the log(s):

File '/etc/.testingfile.txt.swp' was re-added.



 --END OF NOTIFICATION
```

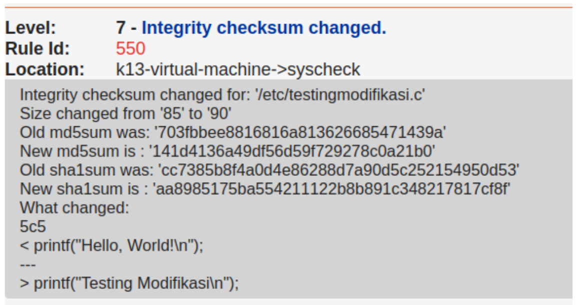
Figure 11. Notification Email File Added

A Level 7 alert notification indicating a change in file integrity successfully triggered the automatic sending of an email, as shown in the Figure above. The content of the email-from the subject “Level 7 - File added to the system.” to the log details-completely matches the alert displayed on the web dashboard.

Test results indicate that the system is capable of detecting new file additions in real-time, while providing audit information that includes the identity of the perpetrator, as well as sending alerts through multiple channels, namely the dashboard and email. This capability is crucial in supporting fast and efficient incident response.

2) *File Change Detection Testing*

This test aims to ensure that the system is capable of detecting any changes or modifications to the contents of existing files. This is the primary function of File Integrity Monitoring (FIM), which is to ensure that the system can recognize when a file has been modified, as well as identify which parts have been changed. In this test, one line of code in the /etc/testingmodification.c file was intentionally modified.



```
Level:      7 - Integrity checksum changed.
Rule Id:    550
Location:   k13-virtual-machine->syscheck
 Integrity checksum changed for: '/etc/testingmodifikasi.c'
 Size changed from '85' to '90'
 Old md5sum was: '703fbbee8816816a813626685471439a'
 New md5sum is : '141d4136a49df56d59f729278c0a21b0'
 Old sha1sum was: 'cc7385b8f4a0d4e86288d7a90d5c252154950d53'
 New sha1sum is : 'aa8985175ba554211122b8b891c348217817cf8f'
 What changed:
 5c5
 < printf("Hello, World!\n");
 ---
 > printf("Testing Modifikasi\n");
```

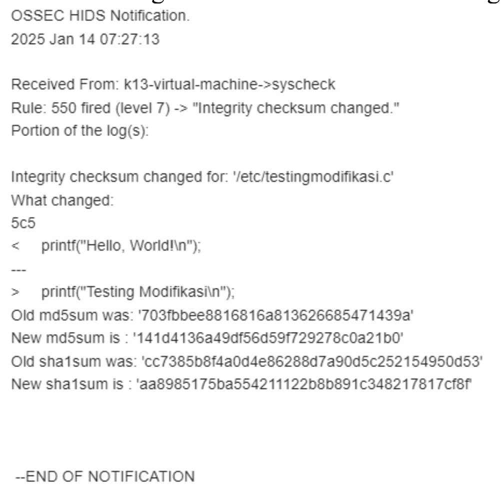
Figure 12. Web UI dashboard of the syscheck module

The figure above shows a Level 7 alert that appears on the syscheck module Web UI dashboard. This alert not only notifies that the file has been modified but also displays detailed forensic information, including:

- File name: It is obvious that the modified file is /etc/testingmodified.c.
- Security code (checksum): The system compares the old and new security codes (MD5 and SHA1) to prove that the file contents are no longer the same.
- The most important feature is the “What changed” section, which shows that line 5 in the file has changed from printf(“Hello, World!\n”); to printf(“Testing Modifications\n”); .

All this information is also sent in full via email, as shown in the following image. As a result, administrators can immediately view the details of the changes in their email inbox without having to open the dashboard. The information displayed includes the name of the file detected as having been modified, namely /etc/testingmodified.c. In addition, the system automatically compares the old and new security code (checksum), to ensure that changes have actually been made to the file contents.

The most important part of this detection result is the What changed feature, which can show specifically which parts of the file have been modifed. Based on this information, it can be seen that line 5 of the file has changed from the command printf(“Hello, World!\n”); to printf(“Testing Modifications\n”);. With this detail, administrators can immediately identify which part of the file has been changed and take action on the change.



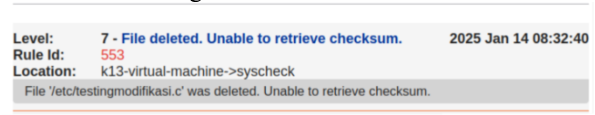
```
OSSEC HIDS Notification.
2025 Jan 14 07:27:13

Received From: k13-virtual-machine->syscheck
Rule: 550 fired (level 7) -> "Integrity checksum changed."
Portion of the log(s):

Integrity checksum changed for: '/etc/testingmodifikasi.c'
What changed:
5c5
<     printf("Hello, World!\n");
---
>     printf("Testing Modifikasi\n");
Old md5sum was: '703fbbee8816816a813626685471439a'
New md5sum is : '141d4136a49df56d59f729278c0a21b0'
Old sha1sum was: 'cc7385b8f4a0d4e86288d7a90d5c252154950d53'
New sha1sum is : 'aa8985175ba554211122b8b891c348217817cf8f'



 --END OF NOTIFICATION
```

Figure 13. Details of the changes via the email inbox

The results of this test prove that the FIM system works very well. Not only can it detect changes, but it also shows the details of the changes and provides strong evidence through checksum codes. This is very important to distinguish whether the changes were made by a legitimate administrator or by an unauthorized party.

3) *File Deletion Detection Testing*

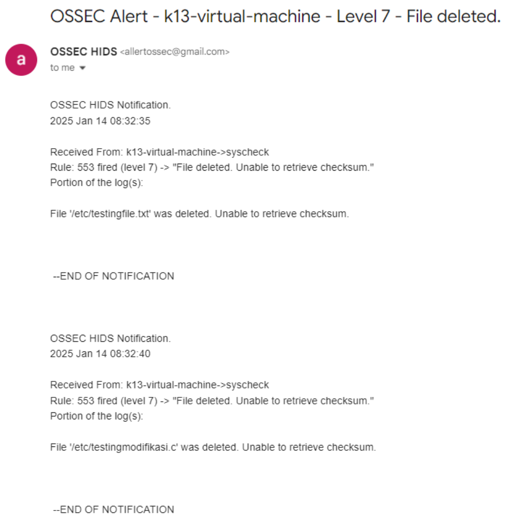
The last test in this FIM series aims to ensure that the system can detect if a file is deleted from the monitored directory. In this scenario, two files previously used in the test-namely /etc/testingfile.txt and /etc/testingmodified.c-were deleted using the rm command.



```
Level:      7 - File deleted. Unable to retrieve checksum.      2025 Jan 14 08:32:40
Rule Id:    553
Location:   k13-virtual-machine->syscheck
  File '/etc/testingmodifikasi.c' was deleted. Unable to retrieve checksum.
```

Figure 14. The Web UI dashboard shortly after the file is deleted

The image above shows the warning that appears on the Web UI dashboard shortly after the file is deleted. The system gives a Level 7 warning with a clear message: "File deleted. Unable to retrieve checksum." This message accurately indicates that the file /etc/testingmodifikasi.c was deleted from the system at 08:32:40.



OSSEC Alert - k13-virtual-machine - Level 7 - File deleted.

OSSEC HIDS <allertossec@gmail.com>
to me

```
OSSEC HIDS Notification.
2025 Jan 14 08:32:35

Received From: k13-virtual-machine->syscheck
Rule: 553 fired (level 7) -> "File deleted. Unable to retrieve checksum."
Portion of the log(s):

File '/etc/testingfile.txt' was deleted. Unable to retrieve checksum.



 --END OF NOTIFICATION



OSSEC HIDS Notification.
2025 Jan 14 08:32:40

Received From: k13-virtual-machine->syscheck
Rule: 553 fired (level 7) -> "File deleted. Unable to retrieve checksum."
Portion of the log(s):

File '/etc/testingmodifikasi.c' was deleted. Unable to retrieve checksum.



 --END OF NOTIFICATION
```

Figure 15. Email notification of File detected

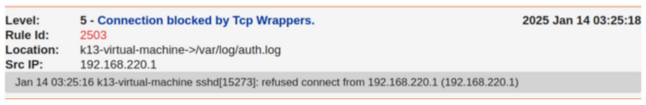
The email notification, shown in Figure above, also indicates the same thing. The email contained two separate alerts, one each for the deleted files: testingfile.txt and testingmodifikasi.c. This shows that the system detected and reported each file deletion separately and accurately.

These test results prove that the File Integrity Monitoring system works well as a whole. The system can not only record when files are created or modified, but also when files are deleted-which is important for detecting acts of sabotage or attempts to erase traces by irresponsible parties.

C. Active Response Testing

This section aims to test OSSEC's active response feature, which is the ability of the system to automatically take action when it detects a threat. In this scenario, a brute-force attack was simulated, which is an attempt to log into the system via SSH by repeatedly entering the wrong username and password from IP address 192.168.220.1. Once the number of failed attempts exceeds a predefined limit, the active response feature automatically blocks the IP address by adding it to the blacklist on the server.

The image below shows the warning that appears on the Web UI dashboard when an attacker tries to reconnect after his IP has been blocked.



```
Level:      5 - Connection blocked by Tcp Wrappers.                2025 Jan 14 03:25:18
Rule Id:    2503
Location:   k13-virtual-machine->/var/log/auth.log
Src IP:     192.168.220.1
 Jan 14 03:25:16 k13-virtual-machine sshd[15273]: refused connect from 192.168.220.1 (192.168.220.1)
```

Figure 16. IP has been blocked

The following is an explanation of the warning:

- The system displayed a Level 5 warning with the message "Connection blocked by Tcp Wrappers." at 03:25:18.
- The log of the SSH service (sshd) also recorded the message "refused connect from 192.168.220.1", which means the connection was refused.
- This indicates that the system was no longer trying to process logins from that IP, but was rejecting them at the network level as they were already on the block list.

This test proves that the OSSEC system is not only able to detect attacks, but also automatically takes action to stop them. This shows that the system works actively in protecting the server, without having to wait for manual action from the administrator.

D. Active Response Testing

To evaluate the efficiency of the system in terms of performance, we measured the resource consumption as well as the time response of the system to file changes. Tests were conducted in two scenarios: 1 container and 8 container scale, with each running a lightweight web service as well as the OSSEC agent on it.

1) Detection and Response Time

   Based on observations from system logs and email delivery times, the average detection time from file change to receipt of email notification is less than 1 minute. This time is ideal for quick response needs in the context of file integrity monitoring.

2) False Positive / False Negative

   During testing on legal scenarios (files created, modified, deleted by authorized users) and simulated brute-force attacks, no false positives or false negatives were found. All activities were successfully recognized and classified according to the severity of the applicable OSSEC rules.

3) Performa Overhead (CPU dan RAM)

   The OSSEC agent's use of resources is relatively light and does not place a significant burden on the container, as summarized in the following Table:

TABLE III

ESTIMATED RESOURCE USE FOR 1 CONTAINER

Komponen	**CPU**	**RAM**
OSSEC agent (idle)	0.1–0.5%	30–50 MB
OSSEC agent (active)	1–3% (burst)	50–100 MB
Web service	0.5–2%	50–150 MB

TABLE IV

ESTIMATED TOTAL FOR 8 CONTAINERS

Komponen	**CPU**	**RAM**
OSSEC agent (idle)	1–4%	400 MB
OSSEC agent (active)	8–12% (burst)	800 MB
Web service	4–16%	800–1200 MB
Total	5–20% CPU	1.5–2.5 GB

IV. CONCLUSION

From the design, implementation, and testing results, OSSEC is proven to be effectively used as a file integrity monitoring (FIM) system in a container environment. It is able to detect activities such as file creation, changes, and deletion in real-time with high accuracy. The workflow for OSSEC-based file integrity monitoring is accompanied by a system architecture diagram. This diagram is intended to facilitate reader understanding and serve as a reference for the development and implementation of similar monitoring systems in container environments. When changes occur, OSSEC also provides detailed information such as differences in file contents and checksums that are helpful in security analysis. In addition, the active response feature works well, where the system automatically blocks the attacker's IP after detecting an attempted brute-force attack. All components, from the agent in the container to the web interface and email notifications, are well integrated to form a responsive and reliable security system.

ACKNOWLEDGEMENTS

Thank you to the respondents who have participated in this research, as well as the entire academic community of the Technology Study Program, Faculty of Vocational Studies, Del Institute of Technology, and various parties who have helped and provided support in completing this research.

REFERENCES

[1] M. Fadlulloh and R. Bik, “Implementasi Docker Untuk Pengelolaan Banyak Aplikasi Web (Studi Kasus : Jurusan Teknik Informatika UNESA),” *J. Manaj. Inform.*, vol. 7, no. Vm, pp. 46–50, 2017.

[2] CBNCloud, “Menguak Praktek Keamanan Terbaik dalam Microservices dan Container.” [Online]. Available: https://cbncloud.co.id/id/blog-news/menguak-praktek-keamanan-terbaik-dalam-microservices-dan-container

[3] Cado Security, “What is Docker architecture?” [Online]. Available: https://www.cadosecurity.com/wiki/what-is-docker-architecture

[4] S. Dwiyatno, E. Rachmat, A. P. Sari, and O. Gustiawan, “Implementasi Virtualisasi Server Berbasis Docker Container,” *PROSISKO J. Pengemb. Ris. dan Obs. Sist. Komput.*, vol. 7, no. 2, pp. 165–175, 2020, doi: 10.30656/prosisko.v7i2.2520.

[5] D. Zhan, K. Tan, L. Ye, H. Yu, and H. Liu, “Container Introspection: Using External Management Containers to Monitor Containers in Cloud Computing,” *Comput. Mater. Contin.*, vol. 69, no. 3, pp. 3783–3794, 2021, doi: 10.32604/cmc.2021.019432.

[6] S. Sultan, I. Ahmad, and T. Dimitriou, “Container security: Issues, challenges, and the road ahead,” *IEEE Access*, vol. 7, no. c, pp. 52976–52996, 2019, doi: 10.1109/ACCESS.2019.2911732.

[7] Y. Cahyaningrum and I. R. Widiasari, “Analisis Performa Container Berplatform Docker atas SeranganMalicious Software (Malware),” *J. Buana Inform.*, vol. 11, no. 1, pp. 47–54, 2020, doi: 10.24002/jbi.v11i1.3279.

[8] E. S. Sinambela, “Evaluasi Performansi Deteksi Serangan pada HIDS OSSEC,” *J. Ilm. Kohesi*, vol. 4, no. 1, p. 35, 2020.

[9] Ronal Hadi, Y. Yuliana, and H. A. Mooduto, “Deteksi Ancaman Keamanan Pada Server dan Jaringan Menggunakan OSSEC,” *JITSI J. Ilm. Teknol. Sist. Inf.*, vol. 3, no. 1, pp. 8–15, 2022, doi: 10.30630/jitsi.3.1.58.

[10] D. P. Widyatono and W. Sulistyo, “Pemodelan Instrusion Prevention System Untuk Pendeteksi Dan Pencegahan Penyebaran Malware Menggunakan Wazuh,” *J. Inf. Technol. Ampera*, vol. 4, no. 1, pp. 113–127, 2023, [Online]. Available: https://journal-computing.org/index.php/journal-ita/index

[11] S. K. Peddoju, “File integrity monitoring tools : Issues , challenges , and solutions,” no. April, pp. 1–8, 2020, doi: 10.1002/cpe.5825.

[12] OSSEC Project Team, “OSSEC.”

[13] P. P. D. Mane, A. Jadhav, O. Sathe, V. Lingade, A. Nagane, and P. Jagdale, “File integrity monitoring,” vol. 05, no. 04, pp. 5574–5576, 2023.

[14] H. Jin, G. Xiang, D. Zou, F. Zhao, M. Li, and C. Yu, “A guest-transparent file integrity monitoring method in virtualization environment,” *Comput. Math. with Appl.*, vol. 60, no. 2, pp. 256–266, 2010, doi: 10.1016/j.camwa.2010.01.007.

[15] J. P. Anderson, “Computer Security Technology Planning Study (Volume II),” *Electron. Syst. Div.*, vol. II, p. 142, 1972, [Online]. Available: https://csrc.nist.gov/csrc/media/publications/conference-paper/1998/10/08/proceedings-of-the-21st-nissc-1998/documents/early-cs-papers/ande72.pdf

[16] D. Teixeira, L. Assunção, T. Pereira, S. Malta, and P. Pinto, “OSSEC IDS extension to improve log analysis and override false positive or negative detections,” *J. Sens. Actuator Networks*, vol. 8, no. 3, 2019, doi: 10.3390/jsan8030046.

[17] Meena R, “What is File Integrity Monitoring and What Files Should I Monitor?” [Online]. Available: https://luminisindia.com/cybersecurity-prism/353-what-is-file-integrity-monitoring-and-what-files-should-i-monitor

[18] J. Jauhiainen, V. Leppänen, and J. Karunen, “Ensuring system integrity and security on limited environment systems,” no. December, 2021.

[19] Linuxhackingid, “OSSEC: Melindungi Sistem dari Ancaman Siber secara Real-Time.”

[20] R. Science, “Perancangan Ids Dengan Teknik Hids (Host Based Intrusion Detection System) Menggunakan Software OSSEC,” vol. 41, no. 0, pp. 1825–1831, 2012.