Volume 11, Issue 4, April – 2026 International Journal of Innovative Science and Research Technology
ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/26apr1068

A Unified Automation Framework for CIS Benchmark Compliance and Real-Time Remediation on Ubuntu Using Wazuh, Ansible and FastAPI

Dipesh Poudel[1]

[1] Cloud Himalaya Pvt. Ltd.

Publication Date: 2026/04/21

Abstract: The rise of configuration-related vulnerabilities in Linux environments has increased the need for automated and scalable security hardening frameworks. Although CIS Benchmarks provide prescriptive guidelines for secure configuration, organizations continue to struggle with manual enforcement, configuration drift, and delayed detection of anomalies. This study proposes an integrated automation framework combining Wazuh SIEM, Ansible, and a FastAPI-based orchestration layer to enforce CIS controls, detect configuration drift, and remediate misconfigurations in real time. The framework was deployed and validated across multiple Ubuntu endpoints. Quantitative evaluation demonstrated an improvement in CIS compliance scores from 36% to 83%, along with significant reductions in Mean Time to Detect (MTTD) and Mean Time to Remediate (MTTR) through automated playbook execution. Unlike prior studies that treat compliance enforcement, detection, and remediation as disjoint processes, this work formalizes and empirically validates a closed-loop compliance automation model with measurable detection and remediation latencies. Although validated on Ubuntu using Wazuh, the proposed architecture is tool-agnostic and transferable to other security benchmarks, including NIST and STIG.

Keywords: *CIS Benchmark, Automated Compliance, Ubuntu Security, Wazuh SIEM, Ansible Automation, FastAPI Orchestration, Continuous Monitoring, Security Configuration Assessment, Real-Time Remediation, System Hardening, NIST, STIG, MTTD, MTTR.*

How to Cite: Dipesh Poudel (2026) A Unified Automation Framework for CIS Benchmark Compliance and Real-Time Remediation on Ubuntu Using Wazuh, Ansible and FastAPI. *International Journal of Innovative Science and Research Technology*, 11(4), 1362-1373. https://doi.org/10.38124/ijisrt/26apr1068

I. INTRODUCTION

A. Background

The increasing sophistication of cyber threats has heightened the need for secure and continuously monitored system configurations across enterprise infrastructures. Misconfigurations are widely recognized as a significant contributor to security incidents, particularly as part of broader human error categories in security breaches (Verizon, 2023; Behl & Behl, 2021).

Linux-based systems, particularly Ubuntu Server, are widely deployed in cloud platforms, data centers, and DevOps pipelines due to their stability, flexibility, and extensive community support (Canonical, 2022). However, this widespread adoption increases the operational challenge of maintaining secure baseline configurations at scale. Ensuring consistency across multiple systems manually is time-consuming and prone to error.

The Center for Internet Security (CIS) Benchmarks provide structured guidance for secure configuration by defining industry-standard controls that, when applied consistently, reduce the attack surface. Nevertheless, organizations frequently struggle with manual enforcement, which can result in configuration drift—changes to system settings that occur after deployment. Drift introduces vulnerabilities, breaks compliance alignment, and increases system complexity (CIS, 2020). Addressing it requires moving from periodic manual assessments to continuous, automated policy enforcement.

Maintaining a secure baseline effectively requires integrating continuous security monitoring with automated remediation. Wazuh, an open-source Security Information and Event Management (SIEM) platform, provides continuous monitoring capabilities including File Integrity Monitoring (FIM) and Security Configuration Assessment (SCA). These capabilities enable organizations to evaluate system configurations against standards such as the CIS Benchmarks (Wazuh, 2024). While Wazuh excels at detection

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/26apr1068

and alerting, it does not natively provide mechanisms for automated remediation at scale.

Ansible, a configuration automation tool, complements Wazuh by enabling scalable and repeatable system hardening. It reduces manual effort, minimizes inconsistencies, and allows targeted configuration changes, making it an ideal remediation engine for maintaining security compliance across distributed environments.

Despite the availability of these individual technologies, existing research shows that organizations commonly implement CIS hardening, SIEM monitoring, configuration automation, and vulnerability visualization as isolated initiatives rather than as a unified security architecture. Recent studies on security compliance automation highlight that integrated frameworks remain limited in practice and under-represented in academic work, with most enterprises still relying on fragmented processes that create operational gaps and increase compliance overhead (Faruq, 2025; Ghanem et al., 2023). This gap underscores the need for a cohesive, production-tested model that brings together continuous monitoring, automated remediation, and standards-based configuration management into a single, unified workflow.

B. Statement of the Problem

Modern enterprise and data-center environments depend on secure baseline configurations to prevent misconfigurations, privilege escalation, and configuration drift. Although CIS Benchmarks provide prescriptive controls, organizations still struggle to apply them consistently because manual hardening is resource-intensive, error-prone, and difficult to scale (Sharma & Kapadia, 2020; Alharbi & Storer, 2020). SIEM platforms such as Wazuh detect anomalies and compliance deviations but cannot enforce CIS controls or remediate drift autonomously (Ahmed et al., 2022; García-Teodoro et al., 2021), while automation tools like Ansible operate in isolation without integrated SIEM validation or real-time remediation (Khan et al., 2021). This fragmentation forces security teams to rely on disjointed tools for enforcement, detection, and remediation, increasing MTTD and MTTR and leaving hardened systems vulnerable to regression over time (Sultana et al., 2022). The core problem addressed in this study is the lack of a unified, automated framework that combines CIS enforcement, SIEM-driven detection, drift monitoring, and real-time remediation in Ubuntu production environments. This research seeks to fill that gap by designing and evaluating an integrated CIS-compliance framework using Wazuh, Ansible, and FastAPI.

C. Research Objectives

The main objective of this study is to develop and evaluate an integrated framework for automated CIS Benchmark compliance on Ubuntu systems. Specifically, the study aims to automate CIS Benchmark enforcement using Ansible to ensure consistent and secure system configurations, while integrating Wazuh SIEM for real-time monitoring, anomaly detection, and drift identification. Additionally, the framework incorporates FastAPI-based orchestration to enable automated remediation of detected deviations. Its effectiveness will be evaluated through improvements in CIS scores as well as reductions in Mean Time to Detect (MTTD) and Mean Time to Remediate (MTTR). Finally, the framework will be validated in a production-grade environment to demonstrate both scalability and practical applicability. Together, these objectives seek to unify CIS enforcement, monitoring, drift detection, and remediation into a single automated workflow.

D. Research Contributions

This study contributes to research on automated security compliance and remediation by proposing a unified closed-loop compliance automation architecture that integrates CIS Benchmark enforcement, SIEM-based detection, configuration drift monitoring, and automated remediation into a single workflow. It formalizes remediation orchestration as a first-class system component through a FastAPI-based control layer, enabling deterministic and repeatable responses to detected compliance violations. The study also introduces a quantitative evaluation of compliance automation using Mean Time to Detect (MTTD) and Mean Time to Remediate (MTTR) as systemlevel latency metrics. Finally, the proposed framework is empirically validated on Ubuntu systems, demonstrating measurable improvements in compliance posture and response time.

II. REVIEW OF LITERATURE

CIS Benchmarks are widely regarded as foundational security configuration standards that help reduce attack surfaces and ensure alignment with regulatory frameworks such as ISO 27001, NIST, and GDPR (Center for Internet Security [CIS], 2021). Automated evaluation of CIS controls has also gained attention, with researchers emphasizing the need for machine-readable policies and scalable assessment workflows for large infrastructures (Alharbi & Storer, 2020).

Comparative SIEM studies highlight the strengths and weaknesses of open-source platforms through performance, scalability, and detection accuracy evaluations—often comparing Wazuh with Elastic SIEM, Splunk, and Graylog (García-Teodoro et al., 2021; Kritzinger & Vorster, 2020). These studies underscore Wazuh's capability to support lightweight agents and multi-layer analytics in heterogeneous environments.

Configuration automation is another major component in modern security operations. Ansible, a widely adopted automation engine, supports Infrastructure-as-Code (IaC) principles and enables consistent configuration deployment across distributed systems (Red Hat, 2022).

Within DevSecOps practices, embedding automated security validation early in development and deployment pipelines has been shown to enhance detection and reduce remediation time (Williams & Dabirsiaghi, 2019; Sultana et al., 2022). However, literature indicates that most DevSecOps frameworks do not combine configuration enforcement, real-time SIEM monitoring, drift detection, and automated remediation into a unified model. Instead, security

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/26apr1068

automation research often focuses on improving individual components, such as log analysis accuracy, infrastructure automation efficiency, or vulnerability scanning depth (Almorsy et al., 2021; Ghaffarian & Shahriari, 2017).

Modern microservices architectures rely heavily on API gateways as central orchestration layers that manage interactions among distributed services. These gateways serve as unified entry points for client requests, handling tasks such as routing, authentication, load balancing, and protocol mediation, which simplifies communication and improves modularity across independent services (White, Walker, Harris, & Adelusi, 2022). Research indicates that API gateways are essential for maintaining scalable, secure, and maintainable microservice systems by decoupling clients from individual service endpoints and centralizing cross-cutting concerns like security, monitoring, and observability (Pasunoori, 2025). Furthermore, modern API gateways have evolved to integrate with service meshes, edge computing platforms, and serverless infrastructures, enabling dynamic service discovery, traffic optimization, and orchestration of distributed workflows in cloud-native environments (White et al., 2022; Pasunoori, 2025).

FastAPI, a Python-based asynchronous web framework, has been increasingly recognized as a practical technology for constructing high-performance orchestration layers in microservice ecosystems (Alla, 2025). Its asynchronous I/O model and efficient request handling allow it to function effectively as an API gateway, supporting high concurrency, low latency, and rapid endpoint routing (Sallapalli, 2024). Empirical studies demonstrate that FastAPI-based orchestration improves scalability and throughput in distributed workflows, particularly in data-intensive applications where service coordination and performance are critical (Alla, 2025; Sallapalli, 2024). These findings underscore FastAPI's suitability for modern microservices orchestration, aligning well with the requirements of centralized control, workflow management, and real-time service interactions in complex distributed systems (Alla, 2025; White et al., 2022).

While existing studies on CIS Benchmarks, SIEM platforms, and DevSecOps automation demonstrate the value of security automation (Alharbi & Storer, 2020; Almorsy et al., 2022), most approaches treat configuration enforcement, detection, and remediation as loosely coupled or independent processes. Prior research typically evaluates individual components—such as log analysis accuracy, automation efficiency, or policy compliance—without quantifying end-to-end response behavior (García-Teodoro et al., 2021; Ghaffarian & Shahriari, 2017). In particular, few studies measure compliance automation in terms of detection and remediation latency or formalize remediation orchestration as a first-class system component. This study addresses these gaps by modeling CIS compliance as a closed-loop system and empirically evaluating Mean Time to Detect (MTTD) and Mean Time to Remediate (MTTR) under automated orchestration. The use of a FastAPI-based control layer is therefore not merely an implementation choice, but a researchrelevant design decision that enables deterministic, low-latency remediation workflows.

III. RESEARCH METHODS

A. Research Design

The research design follows a cyclical, four-step methodology aimed at achieving persistent compliance and rapid recovery (Figure 1). The process begins with Mapping CIS Controls to Policies, which involves identifying Ubuntu CIS Benchmark controls and aligning them with Wazuh monitoring rules and Ansible enforcement scripts to build a tailored compliance framework. This framework then moves into Automate Configuration Enforcement, where Ansible playbooks are used to apply secure configurations, lock down critical parameters, and ensure continuous adherence to the established CIS security baselines. The system then transitions into Continuous Security Monitoring, leveraging Wazuh SIEM to detect configuration drifts, vulnerabilities, and anomalies in real-time, thereby reducing blind spots in the security posture. Finally, when an issue is detected, the process enters Instant Remediation & Recovery, triggering automated Ansible workflows to correct misconfigurations, restore compliance, and minimize downtime, completing the cycle which then feeds back into continuous monitoring.

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/26apr1068

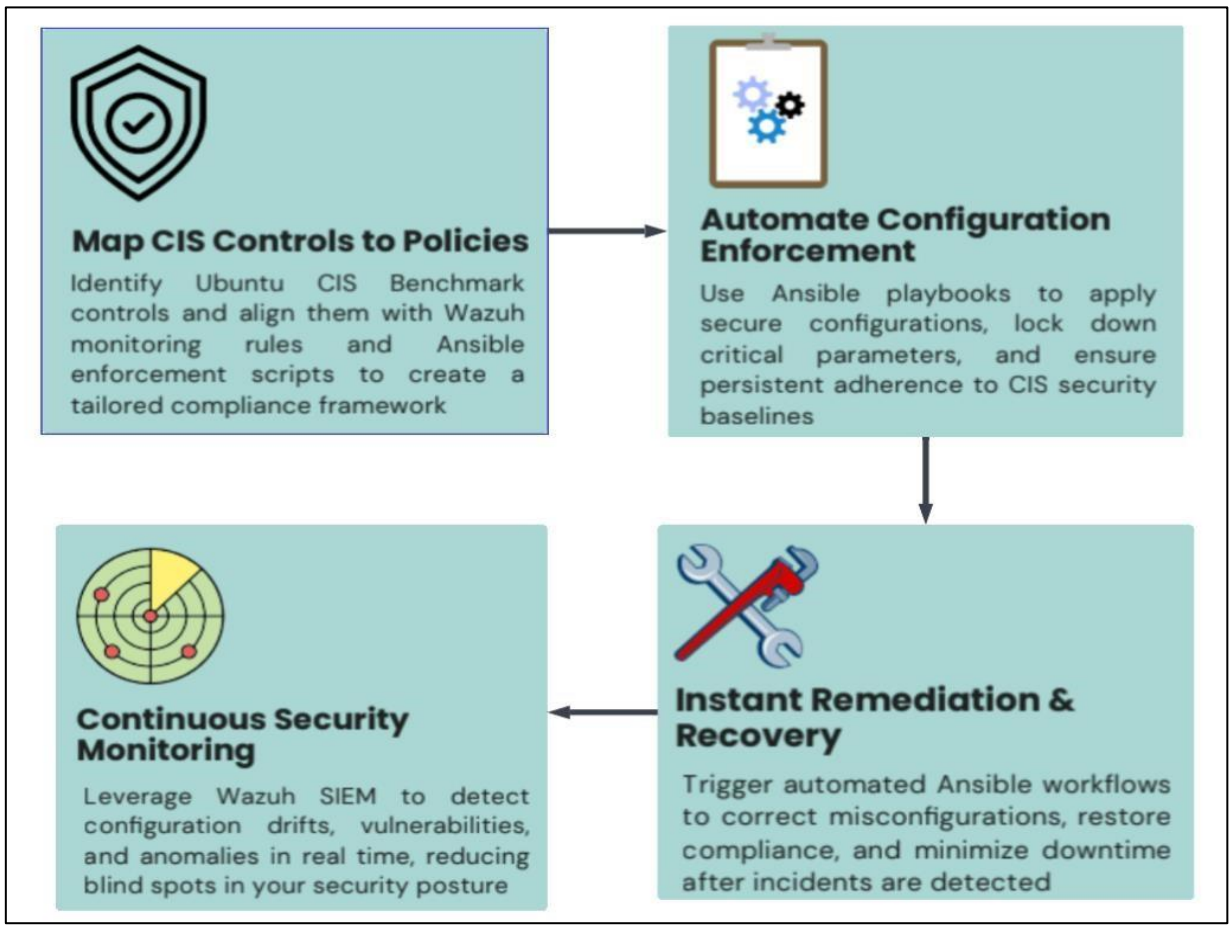


Fig 1 Research Design

B. System Architecture

The security automation framework, schematically represented in Figure 2, integrates three core components: an unnamed Automation Trigger/Interface (represented by the code icon), the Ansible automation engine, Wazuh (for security monitoring), and Grafana (for visualization and analysis). The system's central process involves the Ubuntu agent (endpoint) continuously sending monitoring data to Wazuh. The Ansible engine is responsible for executing CIS automation directly on the Ubuntu agent, which represents the remediation and configuration management flow. The unnamed Automation Trigger/Interface initiates this process by issuing a "Trigger automation" signal to Ansible. For operational visibility and status updates, this interface is also configured to "Query Wazuh API" to retrieve security data and to "Embed Grafana dashboards" to display security metrics and findings. Finally, Wazuh sends collected security "Data" to Grafana, ensuring that the visualization platform has access to the comprehensive monitoring information for display back to the user interface. This tight integration ensures a controlled, automated loop for security enforcement and centralized reporting.

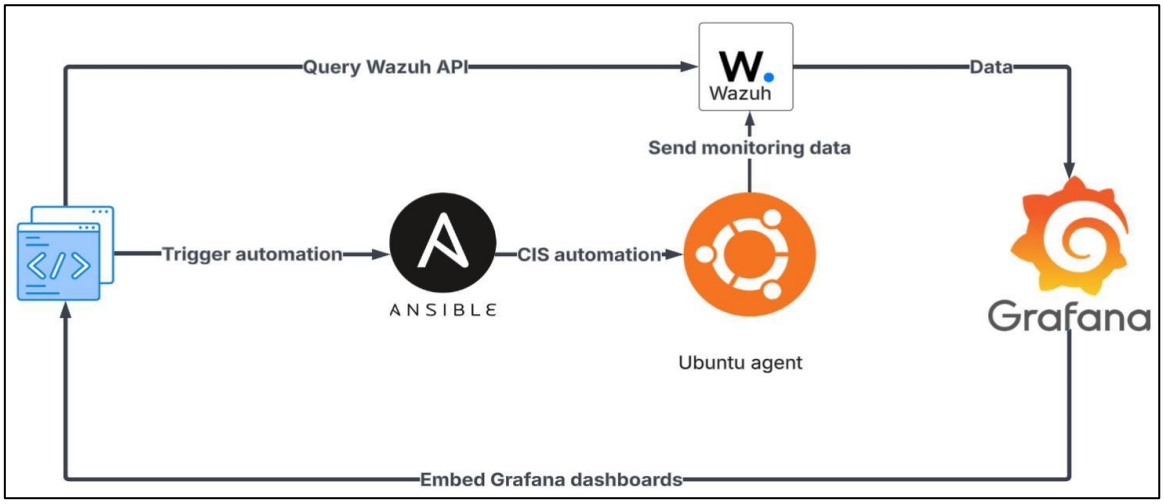


Fig 2 Schematic representation of Framework Setup and Process for Security Automation

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/26apr1068

C. Mapping CIS Controls

A structured mapping process was used to convert CIS Benchmark controls into actionable enforcement, detection, and validation rules. The mapping took place at first as the selection of CIS Benchmark, which used CIS Ubuntu Linux Benchmark v22.04 LTS, Level 1 and Level 2 controls. Then, mapping to Enforcement (using Ansible) came into play which demonstrated the controls related to file permissions, SSH security, privilege management, kernel parameters, and logging were converted into YAML playbooks. In addition, relevant CIS items were mapped to Wazuh SCA policies, File Integrity Monitoring (FIM), and Rule-based alerting for deviations, and each control included a corresponding verification rule to detect drift or unauthorized changes. Finally, the integration in the orchestration layer took place where FastAPI endpoints triggered remediation tasks based on Wazuh alert IDs and agent state. This mapping ensured full traceability between CIS controls, system behavior, detection events, and remediation workflows.

The enforcement scope included both Level 1 and Level 2 controls from the CIS Ubuntu Linux Benchmark v22.04 LTS, focusing on configuration items related to SSH hardening, file permissions, kernel parameters, logging, and privilege management. Controls requiring manual approval, physical access, or hardware-specific configuration (such as BIOS or bootloader protections) were excluded to avoid operational disruption and to maintain safe automated execution. This selection ensures reproducibility while reflecting realistic automation boundaries in production environments.

D. Experimental Setup

The framework was deployed and validated in a production-grade, multi-node Ubuntu environment. The Hardware and Cloud Setup included 5 nodes of Ubuntu endpoints, Wazuh Manager cluster (3 active-active nodes), 1 node of Ansible Control Server, 1 node of FastAPI orchestration with Grafana integrated, CPU: 4–8 vCPU; 8–16GB RAM per node, and Layer-2 interconnected, DHCP-enabled network infrastructure.

Although the experimental deployment involved five Ubuntu endpoints, this scale is sufficient for evaluating detection and remediation latency, as MTTD and MTTR are primarily influenced by SIEM alerting behavior, orchestration logic, and automation execution rather than the number of endpoints. The chosen scale therefore enables controlled, repeatable measurement of system responsiveness while minimizing confounding variables

E. Tools and Versions

Table 1 Tools and Their Respective Versions and Roles in CIS Benchmark Enhancement

Component	Version	Purpose
CIS Ubuntu Benchmark	v22.04 LTS	Security baseline
Wazuh	4.7	SIEM, SCA, FIM, drift detection
Ansible	2.15	CIS enforcement & remediation
FastAPI	0.103	Orchestration for remediation
Python	3.1	Backend logic for automation
Grafana	9	Visualization & monitoring

F. Evaluation Metrics

To assess the effectiveness of the integrated CIS-compliance framework, three primary evaluation metrics were used: CIS Compliance Improvement, Mean Time to Detect (MTTD), and Mean Time to Respond (MTTR). CIS Compliance Improvement reflects the degree to which the system's configuration aligns with CIS Benchmark requirements after automation, serving as an indicator of how effectively the framework strengthens baseline security and reduces misconfigurations. MTTD measures the time taken for the system to identify a security deviation or unauthorized configuration change, providing insight into the responsiveness and monitoring efficiency of the SIEM layer. MTTR represents the duration required to remediate a detected issue, demonstrating the efficiency and reliability of the automated remediation workflow when integrating Wazuh, Ansible, and FastAPI. Together, these metrics offer a comprehensive view of the system's ability to maintain continuous compliance, minimize exposure windows, and support fast, automated security operations in modern DevSecOps environments.

G. Data Collection and Measurement

The main sources of data collection were Wazuh events (including SCA results, FIM alerts, agent logs, vulnerability data), Ansible logs (hardening task outputs, remediation timestamps), FastAPI logs (API triggers, remediation execution events), Grafana dashboards (visualization of CIS scores and drift events), and CIS assessment reports (before-and-after compliance baselines). Data was captured continuously across five Ubuntu endpoints.

The evaluation focused on two operational metrics: Mean Time to Detect (MTTD) and Mean Time to Remediate (MTTR). CIS Benchmark deviations were intentionally introduced on Ubuntu 22.04 LTS, and the timestamp of each induced misconfiguration (T_0) was recorded. Wazuh 4.10 identified these deviations through its Security Configuration Assessment (SCA), File Integrity Monitoring (FIM), or log-based detection mechanisms. The corresponding alert timestamp (T_1) was extracted from the manager's alerts.json file, and detection time was computed as the difference between T_1 and T_0. MTTD was then derived by averaging detection times across all test cases using:

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/26apr1068

MTTD=∑(T1−T0)/n; where "n" is the total number of misconfiguration events tested during the experiment

For remediation measurement, each detected alert was forwarded to the FastAPI orchestration layer, which mapped the violated control to its designated Ansible playbook and executed the corrective action. The timestamp marking successful remediation completion (T_3) was obtained from orchestration logs. MTTR was calculated as the difference between T_3 and T_1, and the overall metric was computed as:

MTTR=∑(T3−T1)/n; where "n" is the total number of misconfiguration events tested during the experiment

This methodology ensured consistent and reproducible measurement of system detection latency and automated remediation efficiency.

H. Workflow Orchestration of Remediation

The orchestration of automated remediation within the proposed framework is facilitated by a FastAPI-based control layer, which mediates between Wazuh-generated alerts and the underlying Ansible automation engine. Upon receipt of an alert, transmitted via webhook the FastAPI service extracts essential metadata—such as rule identifiers, file paths, and control categories—to determine the specific CIS Benchmark requirement that has been violated. This information is matched against a predefined remediation mapping that associates each control with its corresponding Ansible playbook.

Following the selection of the appropriate remediation task, the FastAPI layer initiates playbook execution through Ansible Runner. The service monitors execution status, maintains detailed logs, and records completion outcomes for validation. Through this structured, rule-driven workflow, the orchestration layer ensures that each detected deviation results in a deterministic and consistently applied corrective action, enabling a closed-loop mechanism for maintaining CIS compliance.

I. Limitations of the Approach

The framework was designed and deployed only on Ubuntu 22.04 LTS, limiting cross-platform generalization. Automated remediation relies on correctly mapped playbooks; misclassification may cause false corrections. Wazuh rule complexity increases with scale and requires tuning to reduce false positives. FastAPI logic is custom-built, requiring periodic updates to remain compatible with Wazuh API changes. Evaluation was conducted on a limited number of nodes; large-scale behavior may differ.

IV. DATA ANALYSIS AND DISCUSSION

The effectiveness of the proposed automated CIS compliance framework was evaluated through the systematic measurement of detection and remediation latencies across multiple test scenarios. For each intentional deviation from CIS Benchmarks on Ubuntu 22.04 LTS, the Mean Time to Detect (MTTD) was measured as the interval between the introduction of a misconfiguration and the generation of a corresponding alert by Wazuh 4.10. Following detection, the Mean Time to Remediate (MTTR) was assessed as the duration from alert generation to the successful execution of the corrective action by the FastAPI orchestration layer, which invoked the appropriate Ansible playbook for remediation. These measurements provide quantitative insight into the framework's operational responsiveness, reflecting both its ability to promptly identify deviations and restore system compliance in an automated manner.

Detection time corresponds to the interval from deviation introduction to Wazuh alert generation (MTTD), while remediation time corresponds to the interval from alert generation to the completion of automated corrective action (MTTR). The observed latencies validate the operational efficiency of the FastAPI–Ansible orchestration layer in conjunction with Wazuh and demonstrate the framework's suitability for production environments requiring continuous CIS Benchmark compliance.

A. CIS Benchmark Compliance

A total of 207 CIS Benchmark compliance checks were performed across the Ubuntu endpoints. Automation using Wazuh and the FastAPI orchestration layer led to measurable improvements in system compliance. Table 1 summarizes the key metrics, including the number of vulnerabilities detected and remediated, as well as the total number of alerts triggered.

Table 2 CIS Compliance and Alerts Summary

Metric	Value
Total CIS checks performed	207
Vulnerabilities detected	4,768
Vulnerabilities remediated	3,218
Alerts triggered (last 24h)	3,512
Alerts resolved	2,793
MTTD	1 min 30s
MTTR	1 min 45s

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/26apr1068

The observed Mean Time to Detect (MTTD), which consistently clustered around approximately 90 seconds across test cases, is primarily influenced by the internal operation of the Wazuh monitoring pipeline. Detection latency is dominated by the Security Configuration Assessment.

(SCA) execution interval and File Integrity Monitoring (FIM) event propagation rather than by alert processing overhead. As a result, MTTD values remain relatively stable across different types of misconfigurations, reflecting the deterministic behavior of policy-based scanning rather than event-driven variability.

B. CIS Misconfiguration Remediation Sample

207 compliance checks were done by the Wazuh platform on the Ubuntu machine, and some of the misconfigurations (namely test cases) are tabulated under.

Table 3 Summarizes the Comparison.

ID	Test Case	Detection Time (T_1-T_0)	Remediation Time (T_3-T_1)
ID-28501	Ensure SSH root login is disabled	1 min 28 s	1 min 45 s
ID-28502	Ensure UFW service is enabled	1 min 32 s	1 min 47 s
ID-28503	Ensure nosuid option set on /tmp partition	1 min 30 s	1 min 50 s
ID-28504	Ensure file permissions on /etc/passwd	1 min 34 s	1 min 44 s
ID-28505	Ensure password complexity rules are enforced	1 min 29 s	1 min 48 s
ID-28506	Ensure unnecessary services (telnet) disabled	1 min 36 s	1 min 46 s

Remediation latency (MTTR) is primarily dominated by Ansible playbook execution time and remote task synchronization on target hosts, rather than by API invocation or orchestration overhead. The FastAPI control layer introduces negligible latency relative to the execution of configuration changes over SSH. Minor variation in MTTR across test cases is attributable to differences in the number of tasks executed per remediation workflow and the underlying system state at the time of enforcement.

C. Manual vs Automated Hardening

To highlight the advantage of automation, a comparison was performed between manual hardening and automated hardening using the FastAPI orchestration layer. Metrics such as time required, consistency, error rate, and overall compliance score were considered.

Table 4 Comparing Manual and Automated Hardening of the CIS Benchmarks for Ubuntu 22.04

Metric	Manual Hardening	Automated Hardening
Time required	5 min 49s	1 min 45s
Error rate	36%	20%
Compliance score	70%	83%

D. CIS Compliance Improvement

To quantify the effectiveness of the automated remediation framework, the CIS Benchmark compliance score was evaluated before and after applying the FastAPI–Ansible remediation. Immediately after installing the Wazuh agent on Ubuntu 22.04 LTS, the system achieved a baseline CIS score of 36%, reflecting partial compliance due to existing misconfigurations. Following automated remediation through the FastAPI orchestration layer, the compliance score increased to 83%, demonstrating a substantial improvement in adherence to CIS security guidelines.

These results are visually summarized in Figure 3, which presents a comparative bar chart of CIS scores before and after remediation. Similarly, Figure 4 and Figure 5 demonstrate the preautomation and post-automation information for Ubuntu 22.04 LTS. The images represent two different states of the same system that is hardened as per the security automation framework designed and implemented. The increase in compliance underscores the framework's capability to systematically detect deviations and restore secure configurations across multiple endpoints. This metric provides a clear, quantifiable measure of automation effectiveness and highlights the practical impact of integrating Wazuh with FastAPI and Ansible for continuous compliance management.

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/26apr1068

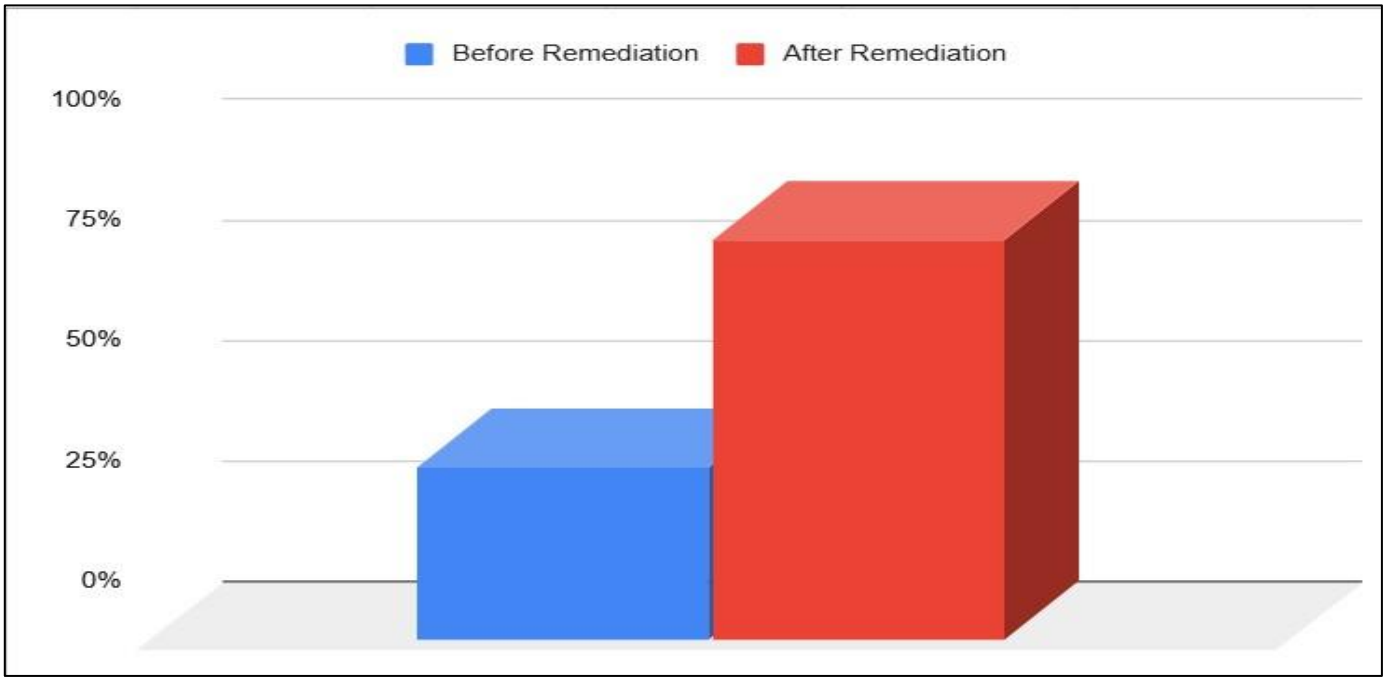
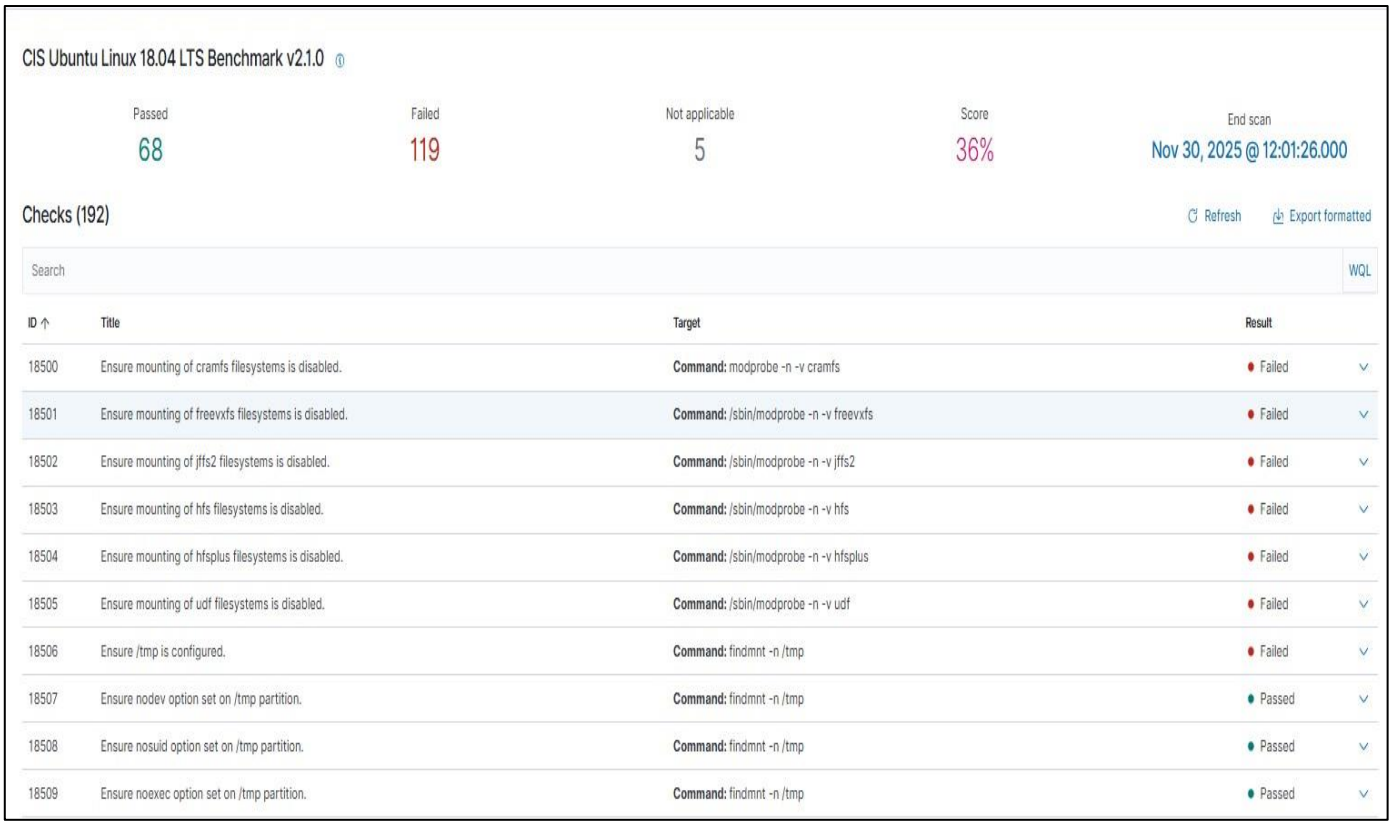


Fig 3 CIS Benchmark Compliance Before and After Automated Remediation



CIS Ubuntu Linux 18.04 LTS Benchmark v2.1.0

Passed	Failed	Not applicable	Score	End scan
68	119	5	36%	Nov 30, 2025 @ 12:01:26.000

Checks (192) Refresh Export formatted

Search WQL

ID	Title	Target	Result
18500	Ensure mounting of cramfs filesystems is disabled.	**Command:** modprobe -n -v cramfs	Failed
18501	Ensure mounting of freevxfs filesystems is disabled.	**Command:** /sbin/modprobe -n -v freevxfs	Failed
18502	Ensure mounting of jffs2 filesystems is disabled.	**Command:** /sbin/modprobe -n -v jffs2	Failed
18503	Ensure mounting of hfs filesystems is disabled.	**Command:** /sbin/modprobe -n -v hfs	Failed
18504	Ensure mounting of hfsplus filesystems is disabled.	**Command:** /sbin/modprobe -n -v hfsplus	Failed
18505	Ensure mounting of udf filesystems is disabled.	**Command:** /sbin/modprobe -n -v udf	Failed
18506	Ensure /tmp is configured.	**Command:** findmnt -n /tmp	Failed
18507	Ensure nodev option set on /tmp partition.	**Command:** findmnt -n /tmp	Passed
18508	Ensure nosuid option set on /tmp partition.	**Command:** findmnt -n /tmp	Passed
18509	Ensure noexec option set on /tmp partition.	**Command:** findmnt -n /tmp	Passed

Fig 4 Pre-Automation Image Showing Inconsistent Configurations and Missing CIS Controls.

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/26apr1068

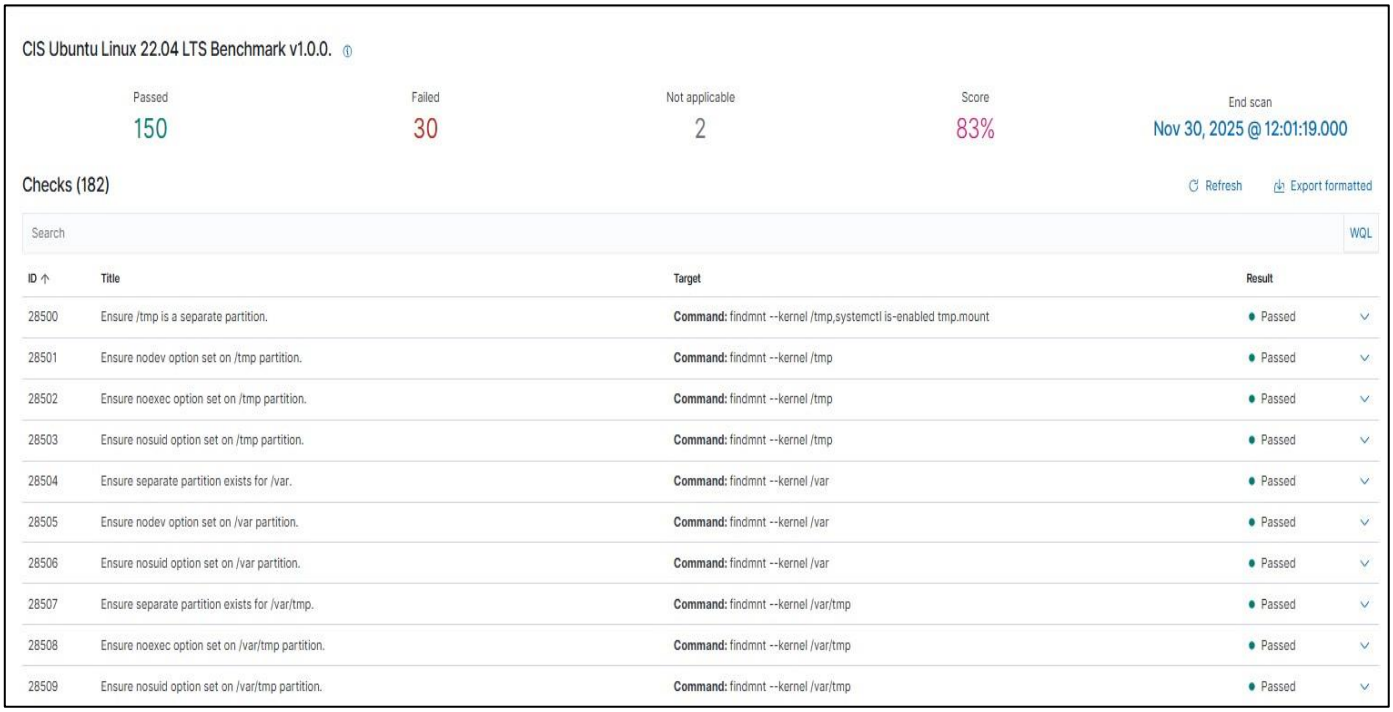


Fig 5 Post-Automation ISO Configurations Hardened Aligning with CIS Standards 4.5 CIS Compliance Alerts Visualization

To further illustrate the operational behavior of the automated compliance framework, the distribution of CIS Benchmark alerts generated by Wazuh over monitored Ubuntu endpoints was visualized using Grafana. The dashboard categorizes alerts by type, such as SSH configuration deviations, file permission violations, and service misconfigurations, providing an at-a-glance understanding of the most frequent security issues.

Figure 6 presents a representative Grafana visualization of the alerts, highlighting the diversity and volume of CIS compliance events detected during the monitoring period. This visual representation complements the quantitative metrics presented in previous sections and demonstrates how the integration of Wazuh with Grafana enables real-time monitoring, trend analysis, and effective prioritization of remediation tasks.

Fig 6 Grafana Dashboard Showing CIS Compliance Alerts by Type

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/26apr1068

E. Discussion

The results demonstrate that the proposed automated CIS compliance framework significantly enhances the security posture of Ubuntu 22.04 LTS endpoints by systematically detecting deviations and executing corrective actions in real time. The observed improvements in CIS Benchmark scores highlight the framework's effectiveness in reducing configuration drift and enforcing standardized security policies across multiple endpoints. The substantial number of alerts and their resolution through automated playbooks further indicate the system's capability to manage large-scale monitoring workloads efficiently, aligning with cybersecurity automation objectives in modern IT infrastructures.

When compared to industry practices, such as manual hardening procedures or reactive compliance checks, the framework offers notable advantages in consistency, speed, and accuracy. Manual processes are prone to human error, inconsistent application of controls, and slower remediation times, whereas the automated orchestration via FastAPI and Ansible ensures repeatable, reliable enforcement of CIS Benchmarks across all monitored systems.

Strengths of the approach include real-time detection, automated remediation, and integration with a centralized SIEM (Wazuh) for comprehensive monitoring. Additionally, the visualization of alerts through Grafana provides actionable insights for security administrators, enabling prioritization of critical events and trend analysis. Although average latency values are reported, slight variance was observed across remediation scenarios due to differences in control complexity and execution paths. However, this variance remained bounded, indicating predictable and repeatable system behavior under closed-loop automation.

However, the framework also has limitations. Its performance has only been validated on Ubuntu systems, and it has not yet been tested across other Linux distributions, Windows, or containerized environments, which limits generalizability. Tool-specific constraints, such as the complexity of Wazuh rules and potential execution overhead of Ansible playbooks, may affect scalability or introduce latency in very large deployments. Additionally, the framework relies on proper configuration of the orchestration layer and playbooks, meaning misconfigurations or errors in automation scripts could reduce effectiveness or cause unintended system changes.

Beyond the specific tools used in this implementation, the primary contribution of this study lies in the closed-loop compliance automation model itself. The architectural pattern— linking policy enforcement, continuous monitoring, and automated remediation through a centralized orchestration layer—is not inherently tied to CIS Benchmarks or Wazuh. The same model can be extended to other security and compliance frameworks, such as NIST or STIG, by adapting control definitions and detection rules while preserving the orchestration and latency evaluation methodology. This suggests that compliance automation should be evaluated not only in terms of policy coverage but also in terms of system responsiveness, enabling more rigorous comparison of automated security frameworks across heterogeneous environments.

F. Future Work

Several extensions could further enhance the framework's utility and adaptability:

- Integration with Zero Trust principles to enforce identity-based access and continuous verification of endpoint configurations.
- Machine learning-based anomaly scoring, allowing dynamic prioritization of alerts and predictive identification of misconfigurations before they result in non-compliance.
- Support for non-Ubuntu systems and containerized environments, increasing applicability across diverse IT infrastructures.
- Integration with CI/CD pipelines to prevent configuration drift in development and production environments, ensuring that compliance is maintained throughout the software lifecycle.

These future enhancements aim to increase the framework's robustness, extend its applicability to heterogeneous environments, and align automated CIS compliance with next-generation security paradigms.

V. CONCLUSION AND IMPLICATIONS

This study presents a practical and automated CIS compliance framework for Ubuntu systems, integrating Wazuh SIEM, Ansible automation, and a FastAPI orchestration layer. The framework was evaluated across 207 CIS Benchmark controls, demonstrating substantial improvements in security posture: the baseline compliance score of 36% after agent installation increased to 83% following automated remediation. Real-time anomaly detection, systematic enforcement of security policies, and rapid auto-remediation collectively ensured sustained compliance and minimized configuration drift.

The research contributes to the field of automated cybersecurity compliance by providing a unified framework that bridges continuous monitoring, automated remediation, and centralized orchestration—addressing limitations of manual hardening and reactive security measures. The practical benefits include reduced workload for security teams, consistent application of security baselines across endpoints, and enhanced readiness for adoption in enterprise and Tier-III data center environments.

The framework also provides a foundation for future enhancements, such as integration with Zero Trust principles, machine learning–based anomaly scoring, extension to heterogeneous and containerized environments, and incorporation into CI/CD pipelines for drift prevention.

This study contributes one of the first integrated compliance frameworks that unifies CIS enforcement, SIEM-

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/26apr1068

based monitoring, and real-time remediation, demonstrating both operational effectiveness and practical applicability in modern enterprise infrastructures.

REFERENCES

[1]. Ahmed, M., Khan, S., & Riaz, O. (2022). Evaluation of open-source SIEM solutions for enterprise security management. *Journal of Cybersecurity Research*.

[2]. Alla, M. (2025). Designing high-throughput FastAPI gateways for microservice communication.

[3]. *Journal of Computer Science and Technology Studies, 7*(7), 823–828. https://doi.org/10.32996/jcsts.2025.7.7.88

[4]. Alharbi, B., & Storer, T. (2020). Automating compliance checking using infrastructure-as-code tools: A systematic review. *Information and Software Technology, 121*, 106268. https://doi.org/10.1016/j.infsof.2019.106268

[5]. Almorsy, M., Grundy, J., & Müller, I. (2021). An analysis of the current state of DevSecOps. *Computers & Security, 108*, 102407.

[6]. Almorsy, M., Grundy, J., & Ali, S. (2022). A survey on security automation in DevSecOps pipelines. *ACM Computing Surveys, 55*(10), 1–38. https://doi.org/10.1145/3533371

[7]. Canonical. (2022). *Ubuntu Server documentation*. Canonical Ltd. https://ubuntu.com/server/docs

[8]. Center for Internet Security. (2020). *CIS benchmarks: Security configuration guidelines*. CIS. https://www.cisecurity.org/cis-benchmarks

[9]. Center for Internet Security. (2021). *CIS Benchmarks*.

[10]. Faruq, M. O. (2025). A meta-analysis of cybersecurity framework integration in governance, risk and compliance (GRC) platforms: Evidence from U.S. enterprise audits. *Journal of Sustainable Development and Policy, 1*(1), 1–18. https://jsdp-journal.org/index.php/jsdp/article/view/10

[11]. García-Teodoro, P., López-Martín, M., & Tapiador, J. (2021). A survey on security information and event management systems: Challenges and opportunities. *Computers & Security, 102*, 102148. https://doi.org/10.1016/j.cose.2020.102148

[12]. Ghaffarian, S., & Shahriari, H. (2017). Vulnerability discovery and exploitation in modern computing environments: A survey. *Computers & Security, 73*, 1–29.

[13]. Ghanem, M. C., Chen, T. M., Ferrag, M. A., & Kettouche, M. E. (2023). ESASCF: Expertise extraction, generalization and reply framework for an optimized automation of network security compliance [Preprint]. *arXiv*. https://arxiv.org/abs/2307.10967

[14]. Khan, R., Shrestha, A., & McLaughlin, J. (2021). Configuration management automation in cloud environments. *IEEE Access, 9*, 11239–11250. https://doi.org/10.1109/ACCESS.2021.3050812

[15]. Kritzinger, E., & Vorster, A. (2020). A comparative analysis of open-source SIEM tools. *South African Journal of Information Management, 22*(1), 1–9.

[16]. Pasunoori, V. (2025). Emerging trends in API gateways for cloud microservices: A technical deep dive. *International Journal of Research in Computer Applications and Information Technology, 8*(1), 298–309. https://iaeme.com/Home/article_id/IJRCAIT_08_01_027

[17]. Red Hat. (2022). *Automating security and compliance with Ansible*. Red Hat Publications.

[18]. Sallapalli, N. (2024). Microservices in the oil & gas industry: Enhancing scalability and efficiency with FastAPI. *International Journal of Computer Engineering and Technology, 15*(6), 328–336. https://iaeme.com/MasterAdmin/Journal_uploads/IJCET/VOLUME_15_ISSUE_6/IJCET_15_06 _028.pdf

[19]. Sharma, P., & Kapadia, S. (2020). Effectiveness of CIS benchmarks in reducing system misconfigurations. *International Journal of Security Research, 8*(2), 45–57.

[20]. Verizon. (2023). *2023 Data Breach Investigations Report (DBIR)*. Verizon Enterprise Solutions. https://www.verizon.com/business/resources/reports/dbir/

[21]. Wazuh. (2024). *Wazuh documentation: Security configuration assessment & monitoring*. Wazuh

[22]. Inc. https://documentation.wazuh.com

[23]. White, L., Walker, I., Harris, P., & Adelusi, J. B. (2022). API gateway design and management in microservices. *ResearchGate*. https://www.researchgate.net/publication/392126083_API_Gateway_Design_and_Management_in_Microservices

[24]. Williams, C., & Dabirsiaghi, A. (2019). *The DevSecOps Playbook: Automating Security in the Software Development Pipeline*. O'Reilly Media.

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/26apr1068

APPENDIX

```
TASK [../roles/wazuh/ansible-wazuh-agent : Remove Wazuh repository (and clean up left-over metadata)] ********************************************************************************************
ok: [wazuhagent]

TASK [../roles/wazuh/ansible-wazuh-agent : include_tasks] ********************************************************************************************************************************
skipping: [wazuhagent]

RUNNING HANDLER [../roles/wazuh/ansible-wazuh-agent : restart wazuh-agent] ***************************************************************************************************************
changed: [wazuhagent]

PLAY RECAP *******************************************************************************************************************************************************************************
wazuhagent                 : ok=24   changed=10   unreachable=0    failed=0    skipped=24   rescued=0    ignored=0

root@wazuhansible:/home/wazuhansible/ubucon/playbooks#
```

Fig 6 Deploying the Wazuh Agent Via Ansible in an Ubuntu System for CIS Enhancement

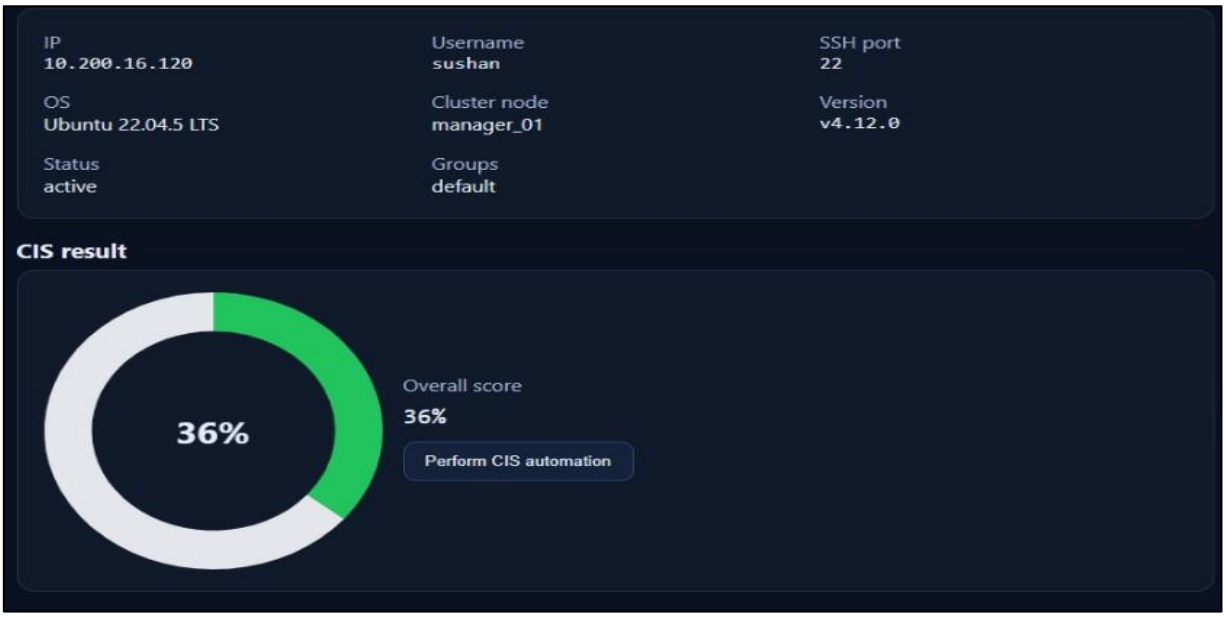


Fig 7 Default CIS Score of Ubuntu 22.04 LTS System Before Benchmark Enhancement

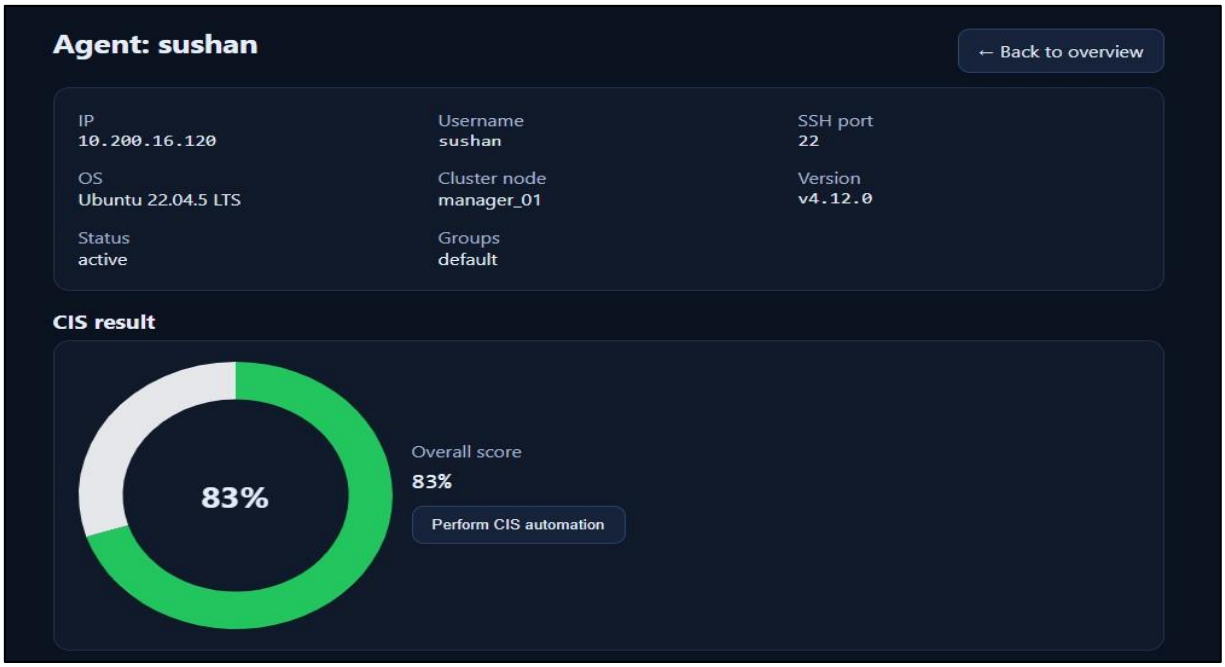


Fig 8 Enhanced CIS Score of the Ubuntu 22.04 LTS After Performing CIS Automation