FPT Education

FPT UNIVERSITY

MINISTRY OF EDUCATION AND TRAINING

FPT UNIVERSITY

Capstone Project Document

Developing a Multi-standard Security Compliance Testing Tool for Open-source Web Applications

REVIEW 3

Group Member	Le Hoang Minh	DE170049
	Tran Anh Khoa	DE170546
	Lam Dan Huy	DE170781
	Huynh Thi Thanh Nguyet	DE170787
	Nguyen Vinh Tu	DE180193
Supervisor	Dr.Nguyen Gia Tri	
Capstone Project code		

ABSTRACT

The widespread adoption of open-source software (OSS) has created a solid foundation for the development and deployment of modern web applications, offering high flexibility and cost efficiency. However, the growing dependence on these complex ecosystems also introduces significant cybersecurity risks, particularly vulnerabilities in the digital supply chain and misconfigurations in the underlying infrastructure. In a context where web applications such as WordPress running on Apache or Nginx servers account for a large share of real-world deployments, system administrators and security practitioners face major challenges due to the fragmentation of security standards. Simultaneously complying with multiple frameworks such as the OWASP Top 10 for the application layer and the CIS Benchmarks for the infrastructure layer often requires combining many disparate tools, resulting in costly and inconsistent manual assessment processes.

This study addresses the issue of fragmented standards by designing and implementing a two-layer security compliance testing architecture. Building on the Compliance as Code philosophy, the thesis proposes a system that comprises a non-intrusive HTTP reconnaissance module, a vulnerability look-up module that queries an external database (the WPScan API), and a secure remote agent mechanism for inspecting deep server configurations (white-box testing). The system operates on a set of 75 standardized YAML rules that are directly mapped to the requirements of OWASP A02, A05, A06 and CIS Benchmark Level 1. Experiments conducted in a Docker-based virtual testbed using a Golden Label dataset show that the system achieves high accuracy and coverage for infrastructure checks thanks to its agent-based mechanism, while also providing effective early warning for application-layer vulnerabilities, thereby demonstrating the feasibility of consolidating multi-standard compliance management on a single platform.

ACKNOWLEDGEMENT

The completion of this graduation thesis is the result of a long process of research, development and continuous effort by our entire team. The thesis could not have achieved its current results without the dedicated guidance and valuable support of many individuals and organisations.

First and foremost, we would like to express our sincere and deepest gratitude to our supervisor, Dr. Nguyen Gia Tri. He not only helped shape the topic and provided in-depth expertise in information security assurance, but also closely followed our progress and offered sharp, constructive feedback that enabled us to refine both the system architecture and the research methodology. His patience and advice have served as a compass guiding the team through the technical challenges encountered during this project.

We would also like to extend our sincere thanks to FPT University for providing a modern learning environment and a solid knowledge foundation throughout our studies. The fundamental knowledge in software engineering and information security imparted by our lecturers has been an essential foundation enabling us to carry out this thesis.

We are also grateful to the open-source community and to organisations such as OWASP, CIS, and the WPScan development team. The standards documentation, tools and open APIs they provide have been an indispensable source of data for building the solution proposed in this thesis.

Finally, we would like to thank our families and friends, who have always been a strong source of emotional support, encouragement and understanding throughout our university years and during this demanding thesis period. Their support has been a powerful motivation for us to fulfil our responsibilities and complete this work.

TABLE OF CONTENTS

List all major sections and sub-sections with page numbers.

LIST OF FIGURES

ABBREVIATIONS

A list of abbreviations and acronyms used in the thesis, with their full forms.

Acronym	Meaning
AI	Artificial Intelligence
API	Application Programming Interface
ASPM	Application Security Posture Management
ACL	Association for Computational Linguistics
AEAD	Authenticated Encryption with Associated Data
CaC	Compliance as Code
CI/CD	Continuous Integration / Continuous Delivery
CA	Certificate Authority
CLI	Command Line Interface
CIS	Center for Internet Security
CMS	Content Management System
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System

CWE	Common Weakness Enumeration
DAST	Dynamic Application Security Testing
DBMS	Database Management System
DevSecOps	Development, Security and Operations
ECDHE	Elliptic Curve Diffie–Hellman Ephemeral
F1	F1-score (harmonic mean of precision and recall)
FN	False Negative
FP	False Positive
GUI	Graphical User Interface
HSTS	HTTP Strict Transport Security
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IA	Information Assurance
ISACA	Information Systems Audit and Control Association
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
LLM	Large Language Model
L1	Level 1 (CIS Benchmark profile)

LTS	Long-Term Support
MTTD	Mean Time to Detect
MTTR	Mean Time to Remediate
NBER	National Bureau of Economic Research
NGINX	NGINX HTTP and reverse proxy server
NIST	National Institute of Standards and Technology
NORMA	NCIRL Institutional Repository (NORMA)
NVD	National Vulnerability Database
OCSP	Online Certificate Status Protocol
OSS	Open-source software
OWASP	Open Web Application Security Project
PCI	Payment Card Industry (in PCI DSS – Payment Card Industry Data Security Standard)
PDCA	Plan–Do–Check–Act
PoC	Proof of Concept
RBAC	Role-Based Access Control
REST	Representational State Transfer
SAST	Static Application Security Testing
SSI	Server Side Includes

SSL	Secure Sockets Layer
TLS	Transport Layer Security
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UX	User Experience
WAF	Web Application Firewall
WP	WordPress
WPScan	WordPress Security Scanner
XSS	Cross-Site Scripting
YAML	YAML Ain’t Markup Language
ZAP	OWASP Zed Attack Proxy

CHAPTER 1: INTRODUCTION

Provides background information, introduces the research problem and outlines the purpose and significance of the study.

1.1. Background

In the modern digital era, open-source software has become an indispensable foundation for building and deploying web applications. More than 90 percent of Fortune 500 companies use open-source products, reflecting the critical role they play in the global economy [1]. Business leaders and policymakers have long debated how to effectively motivate and guide the efforts of open-source code developers, given their central importance in sustaining and advancing the digital infrastructure [1]. From content management systems such as WordPress, Joomla and Drupal, to development frameworks like Node.js and web servers such as Apache and Nginx, the presence of OSS is widespread and brings significant benefits, including low cost, high flexibility and strong support from a global developer community [2]. However, the very openness and reliance on a complex ecosystem of third-party components have created unique security challenges.

Modern open-source web applications are often not a single block of source code but rather a complex digital supply chain, which includes the application core, libraries, plugins, themes and other dependencies [3]. Each component in this supply chain can become a weakness or a potential attack vector. A study conducted by the Information Systems Audit and Control Association (ISACA) in 2022 revealed that 25% of IT professionals reported their organizations had experienced digital supply chain attacks within the previous 12 months [3].

Figure 1.1. Prevalence of Digital Supply Chain Attacks and Expected Near-Term Outlook

The main risks include vulnerable dependencies, the use of outdated or unmaintained components and insufficient security practices from contributors or vendors [4].

WordPress serves as a prominent case study in this context. According to recent statistics, it remains the most popular content management system worldwide, holding a dominant market share far surpassing any competitor [5]. This widespread adoption has also made WordPress an attractive target for malicious actors. However, statistical analyses reveal a noteworthy fact: the majority of security vulnerabilities do not lie in the WordPress core, but are primarily concentrated in third-party plugins and themes. Reports indicate that approximately 90% to 99% of vulnerabilities discovered in the WordPress ecosystem originate from these extensions [6].

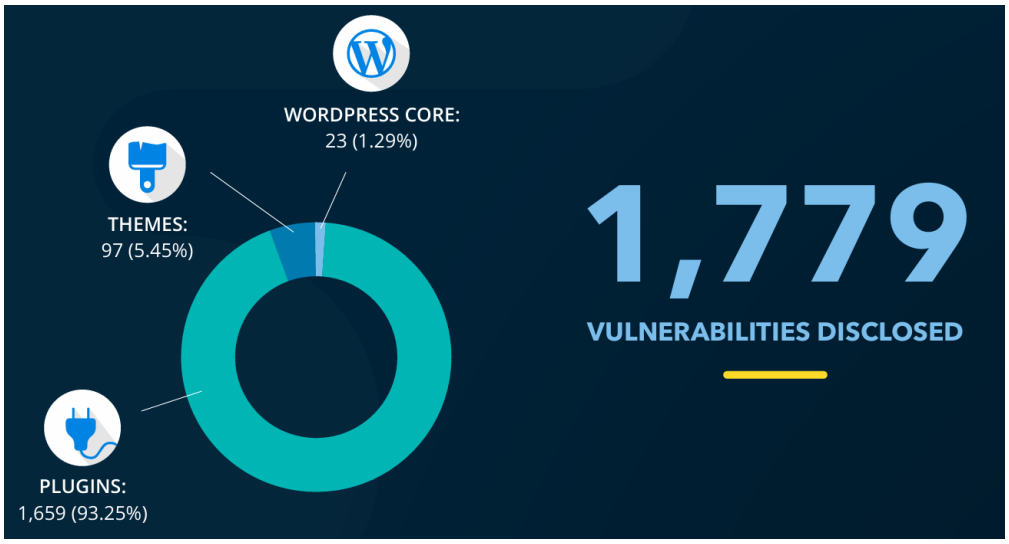


Figure 1.2. Vulnerability Breakdown in the WordPress Ecosystem

This clearly illustrates that securing an OSS application is not only about protecting the source code developed in house but also about managing and mitigating risks across the entire software supply chain. Failure to update these components in a timely manner represents one of the greatest security risks, as vulnerabilities in older versions are often publicized and can be easily exploited [2].

1.2. Problem Statement

According to the OWASP Open Source Software Top 10 project, the industry currently lacks a consistent way to understand and measure risk for OSS, which contributes to the broader problem of fragmented security standards. Developers, system administrators and security professionals working with open-source web applications therefore face a

significant challenge: they must simultaneously comply with multiple overlapping frameworks and guidelines to ensure security across the application stack [7]. For example, they need to adhere to the OWASP Top 10 to address vulnerabilities at the application layer, while also complying with CIS Benchmarks to ensure secure configurations for underlying infrastructure such as web servers and operating systems [8].

At present, the verification of compliance with security standards in open-source web applications remains largely manual or dependent on fragmented, specialized tools designed for individual frameworks. Manual verification is not only resource-intensive and time-consuming but also introduces a high risk of human error. Moreover, the reliance on multiple standalone tools results in inconsistent outputs, complicates report consolidation and hinders the ability to obtain a holistic view of the system's security posture. The lack of an automated and integrated solution capable of evaluating compliance across multiple standards efficiently and consistently represents a critical gap in the security management of open-source web application projects.

1.3. Research Objectives

Based on the identified challenges and research problem, this project establishes the following objectives:

1. In-depth Study: Analyze and consolidate requirements from leading web security standards, with a focus on OWASP Top 10 (2021) and CIS Benchmarks, to determine items with high potential for automation.
2. Tool Design and Development: Develop a web-based security compliance testing application that provides an interactive user interface instead of a command-line tool. The application will automate compliance checks for open-source web applications in accordance with the selected standards (e.g., OWASP Top 10, CIS Benchmarks).
3. Modular and Extensible Architecture: Design the tool with a flexible, modular architecture that enables straightforward integration of new security standards or support for additional open-source platforms in the future.
4. Comprehensive and Practical Reporting: Ensure the tool can generate clear and detailed reports, enabling users to quickly identify non-compliance issues and providing specific recommendations and remediation guidance.

1.4. Significance of the Study

The study is significant in both the academic and practical dimensions, within the scope of OWASP Top 10 (focus on A05/A06) and CIS Benchmarks for Apache/Nginx.

1.4.1. Theoretical significance

This thesis demonstrates a compliance-as-code approach for open-source web security: textual requirements from and CIS are translated into YAML rules and executed by a lightweight rule engine. This turns narrative standards into automated, repeatable checks. The work also proposes a modular reference architecture - component discovery, configuration checks, dynamic checks and reporting - that can be reused and extended to other standards and open-source platforms.

1.4.2. Practical significance

The tool consolidates what are currently manual and fragmented checks into one automated workflow, reducing assessment time and human error while improving consistency across OWASP/CIS items. Because rules are written in YAML and loaded at runtime, teams can add or adjust checks without changing source code, which makes the solution maintainable. The tool outputs clear JSON/HTML reports with remediation aligned to OWASP/CIS, helping developers fix issues quickly and uniformly. Focusing on server configuration hardening (CIS) and vulnerable/outdated components (A06) also matches real-world needs around popular open-source CMS.

1.4.3. Beneficiaries

Developers, Operations and Security teams: faster, consistent checks with actionable remediation.

Organizations using open-source web applications: improved baseline security and audit readiness through consolidated reports.

Education and research: a shareable YAML rule set and a modular architecture that supports teaching and follow-up studies.

1.5. Scope and Limitations

To ensure the feasibility of the thesis within a 15-week timeframe, it is necessary to clearly define the scope and constraints.

1.5.1. Scope of the study

Target application: The project will focus on the most widely used open-source content management system today, WordPress. This choice is based on its broad adoption and its rich, diverse ecosystem of plugins and themes, which together create a realistic and complex attack surface-making WordPress an excellent representative case for study [6].

Target infrastructure: The tool will concentrate on checking the configuration of one of the two most commonly used web servers: Apache HTTP Server or Nginx. Both have detailed CIS Benchmarks that provide a solid foundation for building automated test rules [9].

Standards integrated: The tool will incorporate compliance checks for two primary standards:

- OWASP Top 10 (2021): The focus will be on categories that can be tested automatically or semi-automatically, including A05:2021 - Security Misconfiguration and A06:2021 - Vulnerable and Outdated Components.
- CIS Benchmark Level 1: Level 1 is chosen because it provides basic security recommendations that are broadly applicable without significantly impacting system functionality, which aligns with the goals of an initial compliance checking tool [8].

1.5.2. Project limitations

Product: The developed tool is a Proof of Concept (PoC) intended to demonstrate the feasibility of the proposed idea and architecture; it is not intended to be a fully featured commercial product.

Testing approach: The tool will focus on configuration compliance checks and detection of known vulnerability patterns. The project will not perform active exploitation techniques.

Interface: The tool will be implemented as a web-based application with a simplified user interface. However, due to time and resource constraints, advanced interface features (e.g., visualization dashboards, multi-user management, or enterprise-level integration) will not be fully developed.

The analysis of open-source software risks and relevant security standards reveals a clear dichotomy in testing requirements. The security of an application such as WordPress depends not only on its source code but is also strongly influenced by two external factors: (1) the configuration of the infrastructure on which it is deployed and (2) the security posture

of its dependency ecosystem, including the core version, plugins and themes. An effective compliance testing tool must therefore be capable of addressing both aspects concurrently. This leads to a fundamental architectural decision: the tool should be designed with at least two primary scanning modules that operate independently but are able to integrate their results. One module will focus on static analysis of web server configurations based on CIS Benchmarks rules, while the other will perform reconnaissance and vulnerability correlation for software components using OWASP recommendations and external vulnerability databases. Recognizing this dichotomy at an early stage shapes the overall tool design, making it more practical and valuable.

1.6. Project Management

1.6.1. Project Organization

The project team is structured with clearly defined roles and responsibilities to ensure smooth execution and accountability:

- Project Leader (Huynh Thi Thanh Nguyet): Responsible for overall project management, task coordination, monitoring progress and ensuring quality.
- Developer Team (Nguyen Vinh Tu, Huynh Thi Thanh Nguyet): In charge of tool development and coding of the main modules, including Reconnaissance, Configuration Analysis, Dynamic Scanning and Reporting.
- Research and Documentation (Tran Anh Khoa, Le Hoang Minh): Conducts in-depth analysis of relevant standards (OWASP, CIS), designs YAML compliance rules and prepares project documentation.
- Tester (Lam Dan Huy): Sets up the testing environment (WordPress, Apache/Nginx), develops test cases and executes testing activities.

This organizational structure balances technical development, research and validation tasks to optimize both efficiency and quality.

1.6.2. Project Planning and Schedule

The project follows an Agile-inspired methodology, divided into five phases over 12 weeks:

Week	Phase	Main Tasks	Deliverables

1 - 2	Phase 1: Research and Planning	Finalize thesis proposal and confirm web-based PoC direction. Study OWASP Top 10 and CIS Benchmarks (Apache/Nginx). Analyze existing tools (e.g., WPScan, ZAP, Nikto) for functionality and reporting format. Set up a development environment. Set up a testing environment using Docker with vulnerable WordPress and Apache/Nginx.	Approved detailed thesis outline. Benchmark comparison of tools. Working development and test environments.
3 - 4	Phase 2: System Design	Design modular architecture for backend and web interface. Draft database schema for storing scan history. Define YAML rule schema and rule execution logic. Draft UI wireframes for web applications.	System architecture diagrams. YAML rule specification. Initial UI wireframes. Database design document.
5 - 7	Phase 3: Core Backend Development	Implement Orchestrator module to control scanning workflow. Implement Rule Engine to parse and execute YAML rules. Develop a Reconnaissance Engine to detect platform, version, plugins/themes. Develop Configuration Analysis module to parse Apache/Nginx configs. Write initial YAML rules (basic HTTP headers, directory exposure).	Functional Orchestrator and Rule Engine. Reconnaissance and Config Analysis modules. Initial YAML rules working. Unit test scripts for core modules.

8 - 10	Phase 4: Web Application Development	Develop web-based UI for scan configuration and launching scans. Implement real-time progress tracking (basic visualization). Create a results dashboard with summary and detailed findings. Add database support for scan history (store configs, results). Integrate WPScan API for vulnerability data.	Prototype web application with functional UI. Database integration completed. Vulnerability data integrated from WPScan. Basic visualization of scan results.
11 - 12	Phase 5: Testing	Perform unit testing for each module (Recon, Config Analysis, Dynamic Scanning, Reporting). Conduct integration testing across a full workflow. Test system in Docker-based environments (WordPress and Apache/Nginx). Validate results by comparing with open-source tools (e.g., WPScan, Nikto). Refine YAML rules and fix bugs from testing feedback.	Test cases and logs. Validated scanning results. Refined YAML rule set. Bug fixes and improved stability.
13 - 14	Phase 6: Reporting & Documentation	Implement Reporting Module with HTML dashboard and JSON export. Enhance UI visualization (graphs, compliance indicators). Write thesis chapters: Design, Implementation, Testing. Conduct internal peer review of thesis drafts.	Reporting feature finalized (HTML/JSON). Improved visualization of results. Draft of thesis chapters ready.

15	Phase 7: Finalization and Submission	Complete thesis chapters: Evaluation, Conclusion, Future Work. Finalize all documentation (user guide, deployment guide). Prepare slides and demo for defense. Package tool (source code and Docker setup and documentation). Conduct mock defense within the team.	Finalized thesis report. Packaged tool with documentation. Presentation slides and demo-ready system. Improved defense readiness.

***Table 1.1**: Project Planning and Schedule*

This schedule ensures progressive development with measurable deliverables at each stage.

1.6.3. Resource and Budget Planning

Human Resources: 5 team members with defined roles.

Infrastructure: Personal laptops, Docker-based virtualization for testing.

Technologies: Python 3, with libraries such as requests, BeautifulSoup, PyYAMLand argparse.

Budget: Minimal cost due to reliance on open-source tools; expenses mainly related to virtualization and networking.

1.6.4. Risk and Quality Management

Identified Risks:

Scope Limitation: The project is a web-based PoC that focuses on core compliance testing and modular design. It emphasizes rule-based testing and research values and does not include enterprise-grade features such as automation, multi-user, or continuous monitoring.

Rule Coverage Risk: The implemented YAML rule set may not cover enough OWASP/CIS items to demonstrate broad applicability, reducing the tool's effectiveness.

Integration Challenges: Difficulties may arise when integrating external data sources (e.g., WPScan API) or parsing complex Apache/Nginx configurations.

Mitigation Strategies:

Develop a web interface to increase ease of use, visualize scan results, centrally manage scan history and reduce manual errors-making PoCs more practical, user-friendly and of higher academic value.

Prioritize core and demonstrable rules (e.g., OWASP A05 Misconfigurations, CIS Level 1 basic checks) to ensure meaningful outcomes even with limited scope.

Develop fallback strategies: if WPScan API integration fails, use static vulnerability datasets; if parsing fails, test on simplified configs.

Quality Control Measures:

Conduct unit tests for each module (Reconnaissance, Config Analysis, Dynamic Scanning, Reporting).

Perform integration tests in the Docker-based vulnerable WordPress environment.

Peer review YAML rules and source code to avoid logical errors.

Validate final results by comparing with existing open-source scanners to benchmark accuracy.

1.6.5. Communication and Monitoring

Communication Tools: Messenger/Zalo for daily collaboration; Google Meet for structured weekly meetings.

Progress Reporting: End-of-sprint progress reports and short demos.

Monitoring: Project Leader consolidates progress, compares with timeline and adapts the plan when necessary.

1.6.6. Expected Outcome of Management

By applying Agile principles and structured risk management, the project is expected to:

Deliver a functional Proof of Concept within the 15-week timeframe.

Produce a clear, well-structured thesis with experimental validation.

Provide team members with practical experience in end-to-end development of a real-world security compliance tool

1.7. Thesis Structure

This thesis is organized into six chapters, as outlined below:

Chapter 1: Introduction

This chapter provides background information, presents the research problem, outlines the objectives and explains the significance of the study. It also discusses the scope and limitations of the research and provides an overview of the thesis structure.

Chapter 2: Literature Review

This chapter reviews relevant previous studies, summarizes key insights from the literature and highlights the contribution of this research in addressing existing gaps.

Chapter 3: Methodology

This chapter describes the research design, data collection methods and techniques used for data analysis. It also discusses the limitations of the chosen methodology.

Chapter 4: Experimentation and Results

This chapter presents the experimental setup, introduces the collected data and analyzes the results. It further interprets the findings, compares them with related literature and discusses their broader implications.

Chapter 5: Discussion

This chapter restates the research problem and objectives and provides a discussion of the key findings in relation to the aims of the study.

Chapter 6: Conclusion and Future Work

This chapter summarizes the main conclusions drawn from the study and suggests directions for future research.

CHAPTER 2: LITERATURE REVIEW

This chapter reviews prior studies on three tools: OWASP ZAP, Nikto and WPScan by examining how they operate as well as their strengths and limitations. The aim is to highlight their contributions to web application security and the challenges that remain for future research.

2.1. Review of Previous Studies

To build an effective compliance testing tool, studying existing security standards and frameworks is the core foundation. These documents provide a theoretical basis for identifying test points and formulating automated rules.

2.1.1. Web Application Vulnerability Standard: OWASP Top 10 2021

The Open Web Application Security Project (OWASP) is a global non-profit organization dedicated to improving software security. Their most well-known product is the OWASP Top 10 list, an awareness-raising document that is periodically updated on the 10 most serious security risks to web applications [24]. The list builds on expert consensus and data collected from hundreds of thousands of applications, becoming a standard for the industry. For the goal of building an automated testing tool, it is important to focus on items that can be verified by programming rules. The most suitable categories include:

A02:2021-Cryptographic Failures: The automated tool can check for signs of this error by analyzing the web server configuration to detect the use of weak or outdated encryption protocols (e.g., TLS 1.0, TLS 1.1).

A05:2021-Security Misconfiguration: This category includes bugs caused by a lack of appropriate security reinforcements. Automation checkpoints include the lack of secure HTTP headers, exposing system version information and exposing sensitive folders/files such as .git.

A06:2021-Vulnerable and Outdated Components: This category is especially important for open-source software applications like WordPress. The automated tool can identify versions of core applications, plugins and themes and then query public vulnerability databases (such as the WPScan API) to identify known risks.

2.1.2. Infrastructure Safety Configuration Standard: CIS Benchmarks

The Center for Internet Security (CIS) is a non-profit organization that develops and maintains the CIS Benchmarks, which are globally recognized hardening guidelines [8]. These standards are developed through a community consensus process, ensuring practicality and effectiveness. One strength of CIS Benchmarks is its clear, detailed structure, with each recommendation consisting of Audit and Remediation sections, which are great for converting into automated test rules.

This study focuses on Level 1 of the CIS Benchmarks, as it provides basic security recommendations, which can be widely applied without negatively affecting the operation of the system [8]. Automation checks for Apache/Nginx web servers include configuration file ownership, service account configuration, disabling unnecessary modules and TLS/SSL configuration.

2.1.3. Frameworks and Supporting Tools.

In addition to key standards, frameworks such as NIST Special Publication 800-53 also provide a valuable classification system, helping to map technical findings into a broader risk management context. In terms of tools, existing open-source scanners such as OWASP ZAP, Nikto and WPScan offer features and approaches that are worth learning, but often focus on a single aspect (dynamic scanning, server configuration scans, or WordPress vulnerability scanning).

This section presents studies related to tools such as OWASP ZAP, Nikto and WPScanand analyzes their methods, results, advantages and limitations.

2.1.3.1. OWASP ZAP

OWASP Zed Attack Proxy (ZAP) is an open-source project maintained under the OWASP community and widely recognized as one of the most popular tools for dynamic application security testing (DAST) [10]. ZAP is valued by security professionals for its balance of ease of use and comprehensive vulnerability detection capabilities. Key features include automated scanning, an intercepting proxy for inspecting and modifying traffic, an AJAX spider to handle modern web applications and a fuzzer for testing injection-based vulnerabilities.

ZAP operates as a man-in-the-middle proxy between the browser and the target application, enabling the capture, inspection and modification of HTTP(S) requests and responses [11]. It supports both passive scanning-analyzing responses without interference - and active scanning, which actively injects test payloads to uncover issues such as SQL injection, cross-site scripting (XSS) and insecure cookie handling. Furthermore, its modular architecture allows users to extend functionality through add-ons and custom scripting, making it adaptable for a wide range of research and professional security testing scenarios [10].

OWASP ZAP provides several advantages, including broad vulnerability detection, strong community support, frequent updates and advanced features such as AJAX crawling, customizable authentication and a graphical user interface, as highlighted in industry guidance from StackHawk [10]. However, studies also emphasize its limitations. For example, Alkhaldi and Al-Momani's evaluation of open-source scanners noted that automated scans are not fully comprehensive and often require manual testing to reduce false negatives (2020) [12]. Similarly, benchmarking work by Potti et al. (2025) on OWASP

Benchmark found that ZAP may generate moderate false positives and suffers from performance degradation on complex applications [13].

2.1.3.2. Nikto

Nikto, developed by Chris Sullo, is an open-source web server scanner designed for speed and breadth. It specializes in detecting server misconfigurations, outdated software versions and insecure files. Nikto has built-in checks for more than 6,700 potentially dangerous files and directories, making it a useful first-line tool for basic vulnerability assessment [18].

Nikto operates as a signature-based scanner, sending predefined HTTP requests to web servers and analyzing their responses. It looks for outdated banners, default files, weak headers and other known issues [19]. Its command-line interface allows easy automation and integration with other security tools, but its scanning relies heavily on known patterns rather than adaptive analysis.

Nikto stands out for its simplicity, speed and broad coverage in detecting server misconfigurations and outdated components. It is free, lightweight and open-source, making it practical to deploy across large networks. Its built-in database of more than 6,700 potentially dangerous files and CGI checks enables wide coverage [19]. Nikto is lightweight and easy to deploy across different environments, making it a practical first-line tool for basic vulnerability assessments [21]. Despite these strengths, Nikto has several drawbacks. It cannot detect zero-day vulnerabilities or complex logic flaws, leading to false negatives in advanced web applications [22]. Nikto remains useful for basic web server scanning, but several limitations reduce its effectiveness in modern environments. The tool relies heavily on outdated vulnerability signatures and struggles to detect threats that have evolved with today's dynamic, JavaScript - driven applications. Because Nikto does not support authenticated scanning, it cannot analyze deeper application logic or identify issues that only appear after login. Its detection accuracy is also limited, leading to false positives and missed vulnerabilities, especially when compared with newer scanners designed for modern web architectures. Furthermore, Nikto provides limited support for contemporary web technologies and frameworks, making it less suitable for assessing complex or interactive websites. [20].

2.1.3.3. WPScan

WPScan is an open-source security scanner specialized for WordPress sites; it relies on a dedicated and frequently updated vulnerability database to detect known issues in WordPress

core, plugins and themes, which makes it a popular choice among administrators and researchers for quick auditing [14].

The tool works as a black-box scanner, enumerating site components such as installed plugins, themes and user accounts. It then correlates this information with its vulnerability database, either locally or through the WPScan API. This approach enables WPScan to identify outdated or insecure components without requiring access to the site's source code [15].

According to Farrell's thesis (2023), which analyzed vulnerability scanner performance on large sites such as Spanish banks, WPScan offers advantages like free non-commercial use, high automation and strong community support (over 7.2k GitHub stars). It was also found to reduce scan times from ten minutes to almost zero. However, limitations include less detailed reporting, a restricted API (75 calls per 24 hours)and the requirement of high technical skills [16]. Similarly, Junaid's paper (2021) highlighted WPScan's ability to enumerate WordPress core, plugins and users without dashboard access, making it effective for simulating real attacker scenarios [17]. Yet, its reliance on frequent updates was noted as a limitation, as outdated signatures could lead to false negatives.

Overall, studies show OWASP ZAP excels in comprehensiveness but needs speed improvements, Nikto is effective for basic scans but lacks depth and WPScan is specialized but limited in scope. Research gaps lie in integrating AI to reduce false positives.

Based on the reviewed studies, it is evident that each tool has its own strengths, limitations and areas of application. To provide a clearer comparison, the table below summarizes the key characteristics of OWASP ZAP, Nikto and WPScan, including their scope, methodologies, reported results, advantages and limitations.

Key characteristics	**OWASP ZAP**	**Nikto**	**WPScan**
Scope / Focus	A full-featured intercepting proxy and web application vulnerability scanner that identifies vulnerabilities such as	A web server scanner that detects outdated software, dangerous files and common misconfigurations	A specialized scanner for the WordPress ecosystem, covering the core, plugins, themes and user enumeration; well-

	SQLi, XSSand misconfigurations.		suited for black-box testing and rapid audits of WordPress sites.
Methodology	Employs Dynamic Application Security Testing (DAST) with an intercepting proxy, supporting active and passive scanning, traditional and AJAX spidering, fuzzing, API integration and scripting	Performs signature - and banner-based checks against an extensive test database, probes paths for known files and directories, reviews TLS/HTTP headers and executes fast unauthenticated requests.	Performs black-box enumeration by issuing crafted HTTP requests to fingerprint software versions and components, then correlates findings with the WPVulnDB vulnerability database. Optional modules support password brute-force and user enumeration. The tool is available as a local CLI and exposes an API for retrieving detailed vulnerability information.
Strengths	Free and open-source tool with a user-friendly GUI, extensible through add-ons, offering comprehensive coverage, CI/CD integration, strong community support	Free, fast and simple; provides an effective baseline for scanning large address ranges, easily chains with tools such as Nmap and is useful for early triage	Provides a free CLI for non-commercial use, integrates easily into CI/CD pipelines and benefits from strong community support

	and support for both automated and manual testing.		
Limitations	Steep learning curve; slow for large apps; false positives; complex setup for authenticated scans; resource-intensive; outdated UI.	Signature-driven and weak against zero-days or logic flaws; prone to false negatives on complex applications; offers only basic reporting and risk scoring; CLI-centric interface; works best when complemented by tools such as ZAP or Nessus.	Dependent on known vulnerabilities, unable to detect zero-days or logic flaws. The free API plan is restrictive, reports are less detailed than those of commercial scanners and results often require technical expertise to interpret and minimize false positives.

***Table 2.1**: Comparison of Open-Source Vulnerability Scanning Tools*

2.2. Summary of the Literature Review

The document overview shows a clear division of security requirements: application-layer security and infrastructure-layer security. The OWASP Top 10 provides a solid framework for identifying application-layer risks, especially misconfiguration issues and obsolete components that can be automated. Meanwhile, CIS Benchmarks provides detailed, structured guidelines to reinforce the safety of the infrastructure layer. Existing tools often only address one of these two areas, leaving a gap for an integrated solution. Based on the reviewed studies and comparative analysis above, this section highlights the main strengths and weaknesses of each tool and identifies common findings on the current state of open-source vulnerability scanners. It particularly emphasizes the remaining limitations that future research needs to address.

2.2.1. Key Findings

OWASP ZAP is the most extensively studied tool, widely benchmarked on testbeds like DVWA, bWAPP, Hackazon and OWASP Benchmark. Studies consistently report strong coverage of common vulnerabilities (SQLi, Path Traversal) and advantages such as GUI support, AJAX crawling and extensibility. However, limitations include performance issues on complex applications, reliance on manual validation to reduce false negatives and moderate false positives [13].

Nikto is mainly evaluated for basic server misconfiguration scanning. It is effective at detecting outdated software, dangerous files and weak configurations, making it suitable as a fast, lightweight scanner. Nonetheless, it lacks depth in vulnerability classification, detailed reporting, stealth capabilities and struggles with zero-day threats or dynamic applications [21].

WPScan specializes in the WordPress ecosystem, excelling in plugin/theme enumeration and leveraging a large vulnerability database. It demonstrates strong community support and automation. Its weaknesses lie in limited API usage for free tiers, reliance on database updates, less detailed reporting and narrow applicability outside WordPress [23].

2.2.2. Research Trends

There is a strong emphasis on benchmarking tools against standardized vulnerable applications (e.g., DVWA, bWAPP, OWASP Benchmark).

Studies increasingly combine tools (e.g., Nikto, Nessus, Nmap) to overcome individual limitations.

WPScan research highlights the importance of platform-specific scanners due to ecosystem risks (plugins, themes).

AI and machine learning approaches have been suggested but remain underexplored for reducing false positives and enhancing automation.

2.2.3. Research Gaps

Existing tools are fragmented and specialized; no single solution integrates multiple scanning capabilities across standards.

Multi-standard compliance checking (e.g., OWASP Top 10 and CIS Benchmarks) is largely absent in current tools.

Automation gaps exist: most tools rely heavily on manual validation or combination with other scanners.

AI-driven enhancements for false positive reduction and adaptive scanning are under-researched.

Overall, while OWASP ZAP, Niktoand WPScan contribute significantly to vulnerability assessment, there remains a clear gap for a unified, extensible, compliance-focused scanning framework tailored for open-source web applications.

2.3. Contribution of Research

This research contributes to the existing field by proposing and designing an integrated architecture, capable of addressing security compliance comprehensively across both layers. The novelty of the topic lies in three main points:

1. Comprehensive Integrated Architecture: Provides a unified view of the security posture, superior to traditional tools that typically focus on a single layer.
2. Realizing Compliance as Code: Apply this philosophy to the field of security testing for open-source software by separating the test logic from the source code and managing them as structured data entities.
3. Design a Secure Remote Scan Agent: Propose a secure agent architecture to perform configuration scans on remote servers, solving a major technical challenge while prioritizing security

CHAPTER 3: METHODOLOGY

Describes the research design, data collection methods and analysis techniques.

3.1. Research Design

The Agile approach was selected not only for its high adaptability but also because it fits the nature of developing a security compliance testing tool. As software ecosystems evolve, security projects must respond quickly to updated standards, emerging vulnerabilities, and shifting user requirements. Nhlabatsi et al. (2009) note that software systems must continuously adapt to “changing business needs, new regulations and standards,” reinforcing the need for rapid security adjustments [25]. Agile - particularly Scrum - enables the project team to run an iterative development process with short sprints, allowing early validation of core modules such as the Reconnaissance Engine, Rule Engine and Agent Communication Layer. This iterative, short-cycle approach allows teams to receive early feedback and adjust the system incrementally [26]. This helps ensure that the implementation stays aligned with the intended design. [28].

Client-server architecture enhances performance, scalability, and security by separating roles and distributing tasks. The model is often adopted because it supports scalability and improved security controls in distributed system contexts. [28]. The client (frontend) serves as the presentation layer, communicating with the backend via RESTful APIs. The backend handles security scanning requests, database queries and communication with remote agents. This separation of concerns keeps the tool maintainable. It supports extensibility, enabling new capabilities - such as intuitive dashboards, role-based access control (RBAC), or CI/CD integration - to be incorporated without modifying the core system. This architectural approach allows new roles or resources to be introduced through lightweight configuration updates rather than extensive code rewrites [29], supports deployment and environment changes without altering the central structure [30], provides built-in mechanisms for integrating enterprise-scale RBAC and CI/CD workflows [31], and allows dashboard components to be generated and embedded rapidly within the existing framework [32].

Moreover, this architecture is well-suited to implementing Compliance as Code - a philosophy that models security requirements as executable rules managed like version-controlled source code [33]. In this system, the backend serves as the central rule-processing engine, while the frontend presents test results and remediation information. This design aligns with the DevSecOps approach, where security is integrated into the development lifecycle rather than deferred to the final stage [34] [35].

From a technical standpoint, the backend is structured as a modular monolith rather than microservices to maintain stability and simplify deployment. A modular monolith architecture organizes the system into loosely coupled modules with clearly defined boundaries, offering a simpler alternative to distributed systems [36].

The project's development process is guided by the PDCA (Plan-Do-Check-Act) cycle [32], in which:

- Plan: Define the scanning scope and the standards to be integrated (OWASP Top 10, CIS Benchmarks).
- Do: Develop modules and APIs according to the sprint backlog.
- Check: Conduct internal testing after each sprint and benchmark the results against the security standards.
- Act: Adjust the rules, refine the logic and optimize performance before the next sprint.

Sprint Plan

General principles:

- Cadence: Two weeks per sprint, hold Sprint Planning/Review/Retro under Scrum [25][26].
- Standards change management: If OWASP Top 10 2025 is released during execution, raise a Change Request and schedule it in Sprint 6 Standards Update without altering the PoC scope.
- Environment: Develop and test in-house on the described stack (Client - Server, REST API, DB, remote agent), do not introduce technologies beyond the system specification.
- Common Definition of Done : Code and tests are green, concise documentation (README/API schema/rule schema), test results retain logs & artifacts, no breaking changes to published APIs.

Sprint 1 - CaC Foundation and Baseline Rule Set

Objective: Establish the Compliance-as-Code foundation and standardize the rule schema, build a minimal Rule Engine scaffold to load/validate rules.

Main backlog:

- Rule Schema design (YAML/JSON): metadata fields (id, title, ref), conditions, severity, Pass/Fail criteria, brief remediation text.
- Build Rule_checkconfig and RuleModel: load a rules directory, validate schema and emit human-friendly errors.
- Draft RuleSet v0.1 for:

OWASP Top 10 (2021): focus on A02, A05, A06 (per system description).

CIS Benchmark (HTTP Server): Nginx 1.x and Apache 2.4 (basic configuration checks).

- Report JSON Schema for Rule_checkconfig (id, target, evidence, pass/fail, refs).

Definition of Done and Acceptance Criteria:

- At least 30 valid rules (≥15 OWASP A02/A05/A06, ≥15 CIS Nginx/Apache).
- Rule loader detects 100% of common schema errors, unit test coverage ≥70% for loader/validator.
- Able to output a JSON report for one sample target.

Sprint 2 - Rule Engine & Minimal API

Objective: Execute rules on input data, expose a minimal REST API for the frontend.
Main backlog:

- Rule Engine (interpret/evaluate): Support basic condition types (exact match, regex, numeric/boolean comparisons).
- Input Adapter: Accept data from Recon/Agent (temporarily mocked) following the target snapshot schema.
- REST API v1 (minimal):

POST /scan (create a ScanJob for one target)

GET /scan/{id} (status and results)

GET /rules (list rules, filter by standard).

- Persist artifacts: Scan results and rule version/hash.

Definition of Done and Acceptance Criteria:

- Correctly execute ≥50 rules (including Sprint 1 rules).
- API returns JSON conforming to the schema; OpenAPI docs are auto-generated.
- Unit/integration tests ≥70% for the Rule Engine and API.

Sprint 3 - Reconnaissance Engine (WordPress) and National Vulnerability Database

Objective: Collect base data for the Rule Engine within the WordPress scope, integrate the National Vulnerability Database.
Main backlog:

- Recon (WP): Detect CMS = WordPress, core version and plugin/theme list (non-intrusive methods, per scope).
- WPSCAN API: Cross-check plugin/theme versions with vulnerability data, map to evidence for rules A02 and A05/A06 concerning vulnerable components (per system description).
- Finalize Target Snapshot Schema: A unified data structure for the Rule Engine.
- Wire Recon into /scan: When creating a ScanJob, the pipeline is Recon to Rule Model.

Definition of Done and Acceptance Criteria:

- Identify the WordPress core version and ≥80% of common plugins on the internal test set; acceptable formatting error ≤5%.

- Call the WPSCAN API, persist related advisory IDs/CVEsand map them to rules with standard references.
- Rules A02/A05/A06 consume real data from Recon/WPSCAN (no mocks).

Sprint 4 - Minimal Agent Communication Layer and Access Security
Objective: Operate a basic remote agent within the stated scope and ensure secure communications per the design.
Main backlog:

- Minimal agent: Receive commands to collect HTTP server configuration (Nginx/Apache focus) and return a normalized snapshot preprocessed configs.
- Secure communications: Token-based authentication over HTTPS as per the system design, no tech added beyond scope.
- Scan Orchestrator: Small command queue, reasonable retry/timeout and persisted AgentTask state.
- Add CIS Nginx/Apache rules driven by agent data SSL/TLS, HSTS, cipher suites, directory listing.

Definition of Done and Acceptance Criteria:

- `Agent collects ≥10 key Nginx/Apache configuration parameters required by CIS and OWASP rules.`
- Token - based auth works, audit logs include a traceId for each ScanJob.
- At least 20 CIS rules use real agent data and no mocks.

Sprint 5 - Reporting, Minimal RBAC and Internal Testing
Objective: Complete the results presentation layer and basic access control as per the system design.
Main backlog:

- Reporting: Dashboard (pass/fail rates by standard) and a detailed findings table (rule id, evidence, remediation).
- RBAC (minimal): Two roles - admin/viewer, restrict scan creation/result access accordingly.
- Version pinning: Store rule version with each report (hash and version) to enable who/what/when traceability.
- Internal testing: Regression across the pipeline Recon/Agent, followed by Rule Model and finally Report generation.

Definition of Done and Acceptance Criteria:

- Dashboard renders ≥100 findings on the test dataset, filterable by standard (OWASP/CIS).
- RBAC correctly blocks out-of-role actions, audit logs include userId/role.
- End - to - end tests via API pass on three sample targets: One WordPress site (Recon and WPSCAN), Nginx host and Apache host.

Sprint 6 - Quality Hardening and Standards Update (if OWASP 2025)
Objective: Optimize performance/stability, process standards changes if released during execution.

Main backlog:

- Rule Engine optimization: Lightweight caching for static data (cached plugin to advisory mappings), profile bottlenecks (I/O and CPU).
- Rule set improvements: Increase coverage (OWASP A02/A05/A06, CIS Nginx/Apache) and eliminate false positives observed in Sprint 5.
- Standards update process: If OWASP Top 10 2025 ships during this period, execute a CR to:
  - Compare category changes,
  - Map existing rules,
  - Add/adjust rules and references,
  - Update the OWASP:2025 tag (no change to PoC scope).
- Packaging and handoff: Installation guide, schemas, operating instructions and sample datasets.

Definition of Done and Acceptance Criteria:

- Reduce repeat−run time by ≥20% on the same target compared to Sprint 5 (record measurement points and methodology).
- Add ≥15 new/adjusted rules; reduce false positives per internal statistics.
- If OWASP 2025 is released: Rules are retagged and a 2021 - 2025 mapping document with examples is included.

This approach ensures methodological rigor in development while meeting the stringent security requirements of open - source web systems, the primary subject of this study.

To support post-scan evaluation and remediation prioritization, the system incorporates an AI-assisted analysis pipeline that runs on pairs of before and after scan reports. A lightweight semantic similarity model (all-MiniLM-L6-v2) estimates how much each rule's

evidence has changed, while a large language model (Gemini 2.0 Flash) refines the fix-level classification and generates a prioritized, structured remediation plan for the target website.

3.2. System Architecture

3.2.1. Overview

The proposed system is divided into three main classes: Client Layer, Server Layer and Target Layer, supported by External Dependencies such as WPSCAN APIs. This architecture follows a modular, distributed model to ensure scalability, maintainability and security through separate processing between front-end, back-end and agent services.

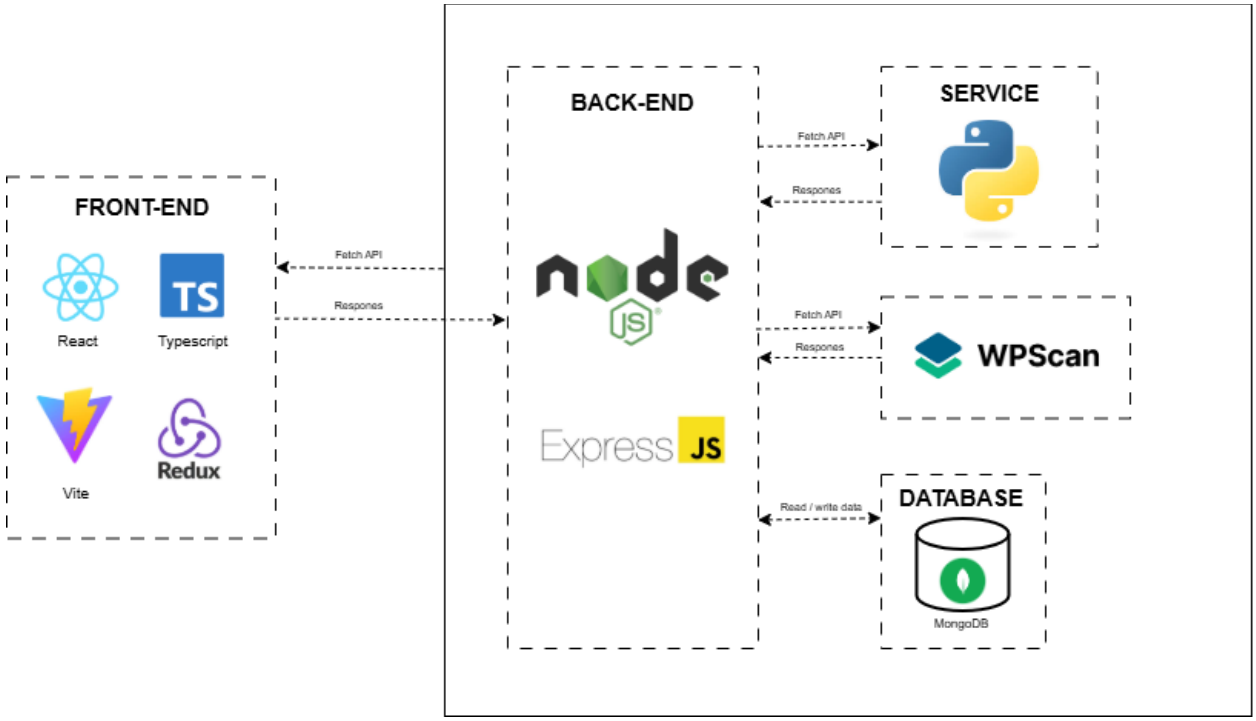


Figure 3.1. System Architecture Diagram

Illustrates the overall system structure, showing the interaction between the frontend, backend, Python service and database.

3.2.2. Technology Stack

Front-end

React is a JavaScript library for rendering user interfaces. UI is built from small units like buttons, text and images. React allows combining them into reusable and nested components [51].

TypeScript helps detect errors early thanks to the static type system and improves the maintainability and scalability of source code in large projects [52].

Vite is a build and dev server tool that helps start projects quickly, automatically reloads and optimizes compilation performance using ES Modules [53].

Redux is created for centralized state management in the application, making it easy for components to share data consistently, predictably and easily control the flow of change [54].

Back-end

NodeJS is a runtime environment for executing JavaScript outside the browser, built on the JavaScript engine. It enables server-side development, supports asynchronous, event-driven programming and efficiently handles scalable network applications [55].

Express is a lightweight web framework for Node.js that makes it easy to create APIs and web applications quickly and easily thanks to a flexible routing and middleware system [56].

Redis, with its support for list and set data structures, can be effectively used as a message queue. This means that it can handle multiple tasks that are lined up for processing. The tasks can be processed either immediately or at a certain scheduled time [60].

Service

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics [57]. Python has a variety of text processing libraries available, regex, parsing, YAML, plus an easy-to-implement agent script. The system's service is developed in Python and it consists of two main functions of the HTTP Recon Service: performing HTTP operations (HEAD/GET), extracting headers, loading HTML, parsing plugins/theme paths, testing public endpoints, collecting TLS metadata, robots and sitemaps. The second is the agent with the data sent by the agent running at the target machine.

In addition to these traditional processing functions, the service layer integrates two AI models to enhance automated analysis. The first is a MiniLM-based Semantic Similarity Engine, built using the SentenceTransformers framework, which computes semantic similarity between BEFORE and AFTER evidence to detect meaningful configuration changes, partial fixes, or suspicious PASS results. The second is a Gemini-based Reasoning Engine, used to perform rule-level diagnostic analysis and generate high-level remediation guidance, including change summaries, priority recommendations, and detailed fix plans. Together, these AI components enable the service layer to move beyond rule-based evaluation toward intelligent, context-aware assessment of system improvements.

WPSCAN APIs

The WPScan WordPress Vulnerability Database API is provided for users and developers to make use of our vulnerability database data. The data includes WordPress vulnerabilities,

plugin vulnerabilities and theme vulnerabilities. This API is used by our WordPress Security Scanner and our WordPress Security Plugin. [46].

Database

MongoDB is an open source, nonrelational database management system (DBMS) [47] that uses flexible documents instead of tables and rows to process and store various forms of data [59].

3.2.3. Client Layer

General Description: The client layer is the part of the web interface where users interact to initiate and monitor the scan. The system has two options: users can choose how to check the server configuration by uploading a configuration file or using a remote agent.

Key Components:

- Dashboard: A dashboard that displays compliance scores, a graph of results and a history of previous scans.
- Scan Manager: An extensive form that allows users to enter the target URL address, select the test standard (OWASP or CIS) and select the configuration method (Config Method). When the user selects Upload File, the system displays a field to load a configuration file such as httpd.conf or nginx.conf. If the user selects Use Remote Agent, the user downloads the agent and runs it on the machine that contains the web server.
- Detailed Report: The results page displays the findings as a unified table, independent of the selected scan mode.

Technology: The interface is built using React.js, using TypeScript, Redux for state management and Vite for the build process. The form is designed with a conditional rendering mechanism, displaying only the input that matches the selected method.

Operation Flow: The user enters the URL and selects the configuration mode. The Upload File option allows you to send configuration files directly, while Use Remote Agent allows you to connect to the agent running on the target server. When the user presses Start Scan, the data is sent to the backend for processing.

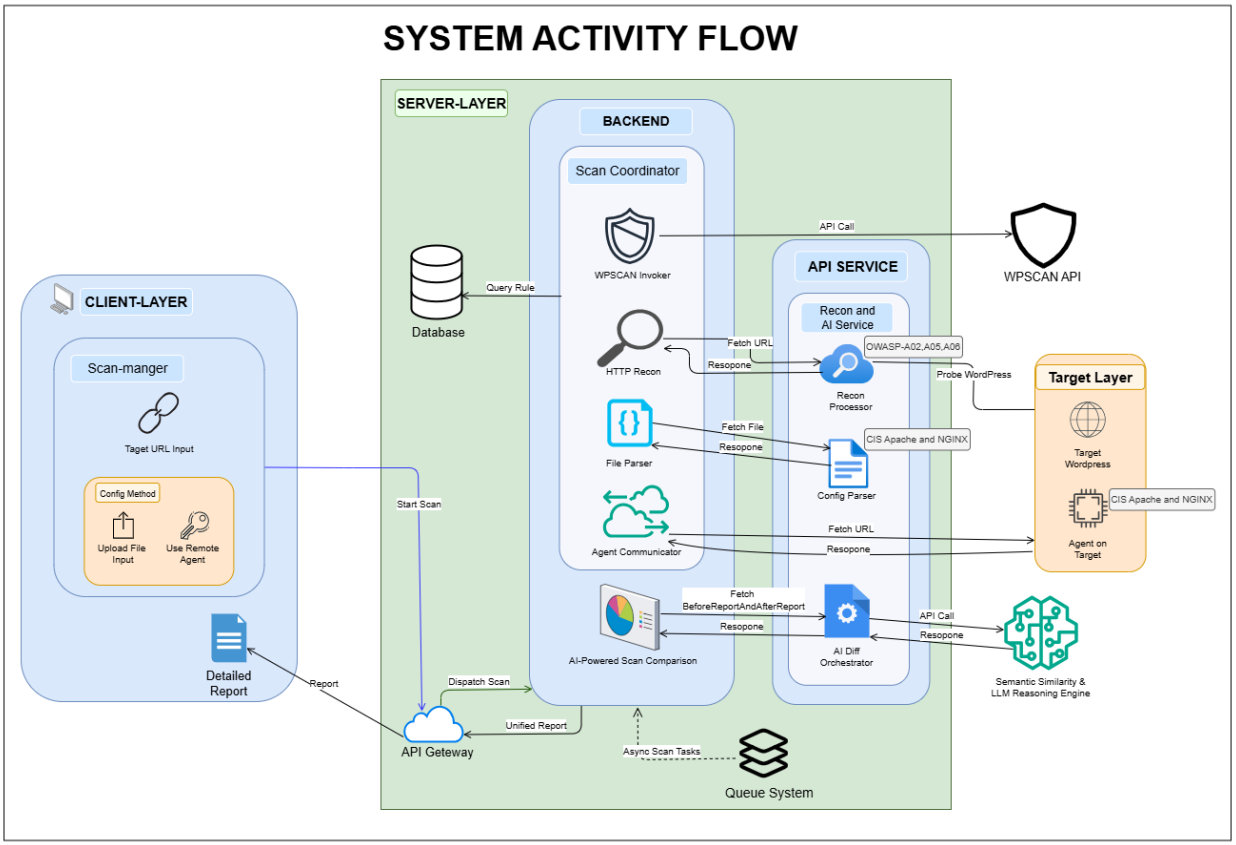


Figure 3.2. System Activity Flow

Describes the interaction between client, backend and service layers during a full scan process.

3.2.4. Server Layer

General Description: The server layer is responsible for the main business process, including coordinating the scanning process, analyzing configurations, communicating with agents and aggregating the results. The entire logic is implemented in Python and Node.js/Express.

Key Components:

- API Gateway: As the primary entry point of the system, receive the request to initiate a scan through the endpoint. The API receives the mode parameter to define the processing method. If the mode is Upload, the system receives the file, saves it temporarily and sends it to the configuration analysis module. If the mode is Agent, the system sends a script to the Agent on the target server for the Agent to execute the test and return the results. Besides, the user selects the mode of URL (HTTP Recon), the system conducts remote collection (HEAD/GET, HTML analysis, TLS check, WPSCAN API query) and directly analyzes the results obtained.

- Database: Stores rules, scan results, user information, default configurations and historical records.
- Queue System: Queue systems are used to handle asynchronous scanning tasks, avoiding congestion when there are multiple requests at the same time.

Technology: The server is developed primarily in Python, incorporating Node.js/Express for the API layer. MongoDB is used for data storage.

Operation Flow: When the user starts a new scan, the backend receives the request from the web interface, generates scan id code and specifies the scan mode as Upload, Agent, or URL Recon. If the user uploads a configuration file, the backend saves the temporary file and sends a request to Service (Python) to analyze the contents. If the user selects Agent, the backend sends the list of rules and target server information to the Service so that the Service can communicate with the Agent, run a local test and then receive the Pass/Fail results. If the user enters only the URL, the backend sends the target path to the service to perform the remote crawl process (HTTP Recon), which includes HTTP header analysis, TLS certificates, HTML code and WPSCAN queries. After the Service completes the task, it returns a JSON analysis for the backend. The backend receives the data, applies the Rule Engine to compare it to other security rulesets (OWASP or CIS), classifies the level of risk and then saves the results to MongoDB. Finally, the backend sends a response to the web interface for the user to view the detailed report.

3.2.4.1. Backend (Node.js and Express)

General Description: Scan Coordinator is the component that coordinates the entire scanning process. It calls modules such as HTTP Recon, WPSCAN, File Parser, or Agent Communicator. The HTTP Recon, File Parser and Agent Communicator components are just adapter modules located in the backend layer. They do not perform complex collection or analysis directly, instead, they call Service through internal APIs or put jobs in a queue for Service to process.

Key Components:

- WPSCAN Invoker: A component that performs queries to the WPSCAN API containers to detect vulnerabilities in the core, plugins and themes.
- HTTP Recon: Normalizes parameters and forwards the job to the Python Service to perform HEAD/GET, HTML parsing and TLS inspection, returns normalized results.

- File Parser: When the user uploads the configuration file, the backend saves the temporary file and calls the Service to analyze the file contents, returning the normalized JSON.
- Agent Communicator: the system supplies users with a packaged Agent Tool (.deb) that already includes all rule definitions and scanning logic. After the tool is installed on the target machine, the Agent Communicator only sends execution commands (scan ID, token, and upload URL). The Agent Tool applies its built-in rules, performs all local compliance checks (Apache/Nginx), and sends the pass/fail results directly to the Service. The backend simply coordinates this process and logs results into the database.
- AI-Powered Scan Comparison: The backend does not perform AI processing directly. Instead, it acts as an intermediary API layer that retrieves the identifiers of the before and after scan reports and delegates the actual comparison task to the AI-powered diff service.

3.2.4.2. Service (Python)

General Description: The service is developed in Python, which is the place to perform all the data collection, analysis and standardization tasks that the backend (Node.js and Express) sends to. Instead of letting the backend handle the heavy lifting itself, Service is responsible for performing intensive operations including Recon Processor, Config Parser and AI Diff Orchestrator. The backend only plays the role of coordinator, sending requests to the service through the internal API (or queue) and then receiving the results returned for storage, rule comparison and reporting.

Key Components:

- Recon Processor: Performs the HTTP Reconnaissance process probes, collects information from the target website.
- Config Parser: Analyzes the contents of web server configuration files (Apache or Nginx) uploaded by users via the frontend.
- AI Diff Orchestrator: compares two scan reports using a combination of rule-based diffing, semantic similarity analysis (MiniLM embeddings), and LLM reasoning (Gemini). It detects meaningful configuration changes, classifies fix levels, and produces detailed AI-generated summaries and remediation guidance. The module exposes an API endpoint that the dashboard consumes to visualize AI-enhanced comparison results.

3.2.5. Target Layer

General Description: The Target layer is only activated when the user selects Use Remote Agent mode and enters the URL of the server or website to be checked.

Key Components:

- Target System: Includes the WordPress site and the actual web server, allowing the backend to perform HTTP Recon.
- Agent: a packaged .deb tool installed on the target server. Instead of receiving rules from the backend, the Agent Tool already contains all embedded compliance rules and scanning logic. After installation, it automatically reads local Apache or Nginx configuration files, evaluates each security directive using its internal rule set, and sends only Pass/Fail results and minimal evidence back to the central system, ensuring privacy by keeping raw configuration data on the target machine.
- Technology: The Agent Tool is developed in Python and distributed as a standalone Debian package (.deb). It can be executed directly through a command-line interface, enabling secure local verification without exposing sensitive server files.

Operation Flow: When a scan is initiated, the backend does not send any rules. Instead, it provides the Agent Tool with execution parameters such as the scan ID, authentication token. The Agent then loads its internal rule set, performs local analysis of server configurations, determines the Pass/Fail status for each security directive, and returns the structured results to the central system.

3.2.6. External Dependencies

The system uses a self-managed vulnerability database model, collecting public data from the WPSCAN API. The data is periodically updated through Python scripts, stored in a local database. This approach provides complete control over the data source, not limited by third-party APIs such as WPSCAN and extended support for OWASP and CIS standards.

3.3. Data Analysis Techniques (Rule - Based Compliance)

Data analytics is the process by which the tool collates the information collected with security standards to draw conclusions about compliance status. The core technique is Compliance as Code, realized by Rule Engine [39]. Instead of hard-coding in the source code, rules are defined as structured records (drafted can be in YAML form and/or saved in a database) for automatic, consistent, repeatable execution.

3.3.1. Objectives and scope

Objective: to convert system-scoped standard requirements/checkpoints into structured rules and execute automatically on the collected data.

Scope: Adhere to the design of the topic, do not add criteria, scales or ranking mechanisms other than what has been described. Input data comes from only three sources in 3.2: (i) HTTP/HTML, (ii) lookup results from the Vulnerability API, (iii) Pass/Fail results returned by agents [38].

3.3.2. Rule Structure

Each rule is centrally managed under a unified data model to decouple the check definition from the execution code and includes the following fields: rule_id: a unique identifier.

- Standard: A reference to the applicable standard's control family/item (for report mapping).
- Description: A concise statement of the check objective.
- Severity: The display/handling priority.
- Type: The check type, emphasized within the system's scope.
- Target: Parameterization of the check scope.
- Rationale: The security rationale explaining why the rule exists.
- Remediation: The corrective guidance to apply when non-compliant.

YAML example:

```yaml
  ruleId: CIS-NGINX-2.1.0-2.3.1
standard: "CIS NGINX Benchmark v2.1.0"
description: "Ensure NGINX directories and files are owned by root."
severity: "High"
type: "file_permission_check"
target:
  path: "/etc/nginx"
  owner: "root"
  group: "root"
assertion:
  condition: "ownership_equals"
  value:
    owner: "root"
```

```
  group: "root"
rationale: "Restricting ownership of NGINX configuration directories and files to the root user and group prevents unauthorized modification, maintaining integrity and security of configuration data."
remediation: |
  Run the following command to set ownership and group ownership to root:
    chown -R root:root /etc/nginx
```

3.3.3. Input Data Constraints

The rule engine does not perform its own data collection. It strictly consumes the following three classes of evidence from 3.2 [38] [40]:

HTTP/HTML (for header_check): The HTTP status code, the relevant set of HTTP headers and observable fingerprints in the HTML that indicate WordPress core, plugins, or themes.

Vulnerability lookup results for cross-referenced rules: A yes/no determination of whether a published vulnerability exists for each component, version pair.

Agent Pass/Fail: A list of Pass/Fail outcomes keyed to configuration-directive codes evaluated on the target host.

Note: The on - target agent never transmits raw configuration contents, the rule engine only receives evaluation results by directive code.

3.3.4. Execution Workflow

1. Load the rule set appropriate to the target context WordPress and the server environment as specified in the system description.
2. Bind the input data for each rule according to its check_type:

- header_check: Map to the header vector and HTML fingerprints collected during reconnaissance.
- config_check: Map to the agent-returned Pass/Fail list, keyed by configuration-directive codes.
- When a rule requires cross-referencing the vulnerability lookup, use the binary (yes/no) status for the corresponding component, version pair.

3. Execute the check_logic against the corresponding technical evidence to determine the Pass/Fail status.

4. Record, for each rule, the following: Rule_id, status Passed/Failed, a condensed evidence summary header name and observed value; configuration - directive code and agent outcome or the component, version pair with its vulnerability - lookup status and remediation.
5. Store the results in the database to enable standards-based aggregation and reporting in the subsequent chapter.

3.3.5. Control and Quality Principles

- Traceability: Each finding is linked one-to-one to its rule_id and evidence source (HTTP/HTML, vulnerability API, agent), enabling cross - verification and reproducibility.
- Consistency: Because rules are defined outside the application code, updating/standardizing them does not require changes to the collection module, results remain stable and repeatable across runs.
- Scope constraints: No aggregate scoring or additional weights/rankings beyond the fields specified in the rule schema, do not extend check types beyond header_check and config_check.
- Reducing false positives via a confidence threshold: If confidence < 0.8, label Inconclusive rather than Failed. Provide a UI feedback loop (users can flag FPs) to update exceptions/allowlists (whitelists) and refine rules in the next sprint.
- Note: Confidence is metadata only and is not used for ranking or scoring.

3.3.6. Outputs and Use in Reporting

The analysis phase produces a catalog of findings comprising: rule_id, standard, severity, status (Passed/Failed), evidence, a condensed technical summary and remediation [41]. This set of findings is grouped by standard/control and serves as the basis for report presentation and the detailed exposition in the results chapter. The end-to-end collect analyze report chain conforms to the system design, remains within scope and introduces no elements beyond the specification.

3.3.7. AI-Assisted Rule Comparison and Remediation Analysis

This component is triggered once the system has obtained two scan reports, *before* and *after*, for the same website. Its objective is to automatically compare the results, determine

the remediation level (fix-level) of each rule, and generate structured remediation suggestions to support prioritization.

MiniLM embedding. For each rule, the fields evidence_before and evidence_after are embedded into semantic vectors using the all-MiniLM-L6-v2 model (SentenceTransformers). The semantic similarity between the two states is then computed and combined with the status transition (e.g., FAIL→PASS, PASS→FAIL, FAIL→FAIL, …) to infer an initial fix-level such as full_fix, partial_fix, still_vulnerable, or suspicious_pass.

Rule-level AI (Gemini). A rule-level LLM (Gemini) processes the list of rule diffs, reviews the change logic, refines the fix-level labels in edge cases, and (by design) is capable of generating short analyses and remediation recommendations for each rule. In the current PoC implementation, the system mainly relies on the refined_fix_level field returned by Gemini to update the evaluation result.

Global-level AI (Gemini). A global-level Gemini agent receives aggregated statistics (number of rules with FAIL→PASS, PASS→FAIL, total rules) together with the list of ruleId:fix_level pairs, and produces a system-wide summary report, including change_summary, priority_recommendation, and a detailed_fix_plan grouped by remediation priority (high / medium / low).

Benefits. This AI module automates the post-scan remediation assessment step, shortens manual review time, helps identify suspicious PASS cases (suspicious_pass), and provides a structured remediation plan that enables the operations team to build a clear and actionable remediation roadmap.

3.4. Data Collection Methods

3.4.1. Overview

In this tool's context, data collection is a controlled sequence of operations to obtain accurate and sufficient inputs for rule-based compliance verification, drawing on three core data sources previously described: (i) HTTP responses and HTML content to fingerprint WordPress and related components, (ii) published vulnerability lookups via the WPSCAN API and (iii) pass/fail outcomes from configuration-directive checks executed by an agent

on the target server. All activities are strictly non-intrusive, no active exploitation is performed, only safe request methods (GET/HEAD) are used, the tool reads only designated configuration items and no configuration-file contents are transmitted outside the central system.

The collection stage adheres to the principles of evidence minimization and non-intrusive collection: only the minimum technical evidence necessary is gathered to support comparison in the analysis phase (Section 3.3). The data scope is bounded by the three sources above, the system conducts no active attacks, records no data beyond what is required for compliance verification and does not exfiltrate configuration file contents outside the target environment. Execution steps and the collection context are logged at the metadata level (status codes, header fields, identification indicators, yes/no flags for published vulnerabilities and pass/fail results by directive code) to ensure repeatability, verifiability and operational safety.

3.4.2. Reconnaissance and HTTP Response Analysis

Objective: Confirm that the target is running WordPress, capture the core version (where observable), inventory plugins/themes surfaced in the page content and collect the set of security-relevant HTTP response headers (HSTS, CSP).

Method: The tool issues HTTP GET/HEAD requests to the target URL using Python's requests library. It parses the HTML response with Beautiful Soup 4 to identify WordPress (path patterns, meta tags) and where visible to record the core version and any active plugins/themes. In parallel, it extracts HTTP headers to support security-configuration checks (HSTS, CSP, etc.) and to detect version disclosure.

Example Implementation:

```
    # Excerpt: analyze_response (WordPress detection + header extraction)
r = session.get(url, timeout=10, allow_redirects=True, verify=False)
status = r.status_code
headers = dict(r.headers)
# Security header checks (HSTS, CSP, ...)
missing = [h for h in DEFAULT_SECURITY_HEADERS if h not in {k.lower() for k in
headers}]
# HTML parsing for WP fingerprints
soup = BeautifulSoup(r.text or "", "lxml")
plugin_slugs = {m.group(1) for m in re.finditer(r"/wp-content/plugins/([^/]+)/", r.text or "")}
```

```
wp_version = None
meta = soup.find("meta", attrs={"name": "generator"})
if meta and "wordpress" in (meta.get("content") or "").lower():
    mver = re.search(r"wordpress\s*([0-9.]+)", meta["content"], re.I)
    if mver:
        wp_version = mver.group(1)
# Output minimal: status, headers, plugin_slugs, wp_version
```

This function performs non-invasive GET queries to the target URL, collects status codes, security headers (e.g., HSTS, CSP)and WordPress identifiers such as versions or plugin paths in the HTML code. The output in minimalist JSON format includes response code, header, list of plugins/themes and WordPress version, for the evaluation of security rules at the analysis stage.

Minimum outputs:

- The status code and the necessary set of HTTP response headers (to support rule checks in the analysis phase).
- WordPress fingerprints and where determinable the version and any plugins/themes surfaced in the HTML.
- No crafted or intrusive payloads are sent, no data beyond the necessary scope is recorded.

3.4.3. External Vulnerability API Querying

Objective: Determine whether the observed components (core/plugins/themes), for their specific versions, are listed as having published vulnerabilities [58].

Method: After validating the scanID and authorization token, the system triggers a Docker-based WPScan CLI execution. The WPScan container is launched with full enumeration settings, producing a JSON output containing all detected vulnerabilities, outdated versions, and advisory metadata for WordPress core, plugins, and themes. The engine parses this JSON result to classify each component as Yes/No (vulnerable or not) and evaluate compliance according to OWASP A06 rule criteria.

Example Implementation:

```
 def run_wpscan_docker(
target_url,
```

```
    base_dir=None,
    output_name="wpscan_full_output.json",
    http_auth=None
):
        if base_dir is None:
        base_dir = Path.cwd()
    if "127.0.0.1" in target_url:
        target_url = target_url.replace("127.0.0.1", "host.docker.internal")
    if "localhost" in target_url:
        target_url = target_url.replace("localhost", "host.docker.internal")
    docker_cmd = [
        "docker", "run", "--rm",
        "-v", f"{mount_path}:/scan",
        "wpscanteam/wpscan",
        "--url", target_url,
        "--enumerate", "vp,vt,u,cb,dbe",
        "--random-user-agent",
        "--api-token", "FDR96asyVhNvsoWlBZtTxwNjxchw2Wwsa8sO1XKRVL4",
        "--format", "json",
        "-o", f"/scan/{output_name}"
    ]
    if http_auth:
        docker_cmd += ["--http-auth", http_auth]
    print("Running WPScan Docker...")
    print(" ".join(docker_cmd))
    try:
        res = subprocess.run(docker_cmd, text=True, capture_output=True)
        print("The information extracted is:",res)
        print("STDOUT:", res.stdout)
        print("STDERR:", res.stderr)
        if res.returncode not in (0, 5):
            print("WPScan failed!")
            return False
```

```
        print(f"WPScan output saved to {mount_path}/{output_name}")
        return True
    except Exception as e:
        print(f"WPScan Docker error: {e}")
        return False
```

Excerpt from run_wpscan_docker() - demonstrates how the tool invokes WPScan through an isolated Docker container with automatic host-path normalization, localhost-to-host.docker.internal mapping, optional HTTP authentication, and structured output handling. The function also prints full execution logs (stdout/stderr), checks return codes, and ensures the scan results are exported to a mounted directory for further processing.

Minimum outputs:

- A list of components with versions and a yes/no flag indicating whether published vulnerabilities exist, as reported by the API.
- Retain the necessary metadata.

3.4.4. Remote Configuration File Scanning via Agent

Objective: Verify configuration directives (on Nginx/Apache) directly on the target host against the rule set issued by the central server, ensuring that no configuration file contents are transmitted.

Method: A packaged .deb Agent Tool is installed on the target server with permissions to read the necessary configuration files. Instead of receiving rules from the central server, the agent contains all compliance rules and scanning logic internally. After installation, it automatically evaluates each directive locally against its built-in rule set. Communication with the central system occurs over HTTPS/TLS using an API key, and the agent returns only Pass/Fail results with minimal evidence - never sending configuration contents - thereby enforcing the principle of least privilege and preserving privacy.

Example Implementation:

```
def collect(apache_root):
conf_files = list_conf_files(apache_root)
report = {
    "apache_root": apache_root,
    "files_scanned": len(conf_files),
    "modules": [],
    "directives": {},
```

```
    "ssl": {"certs": [], "keys": []}
}
# Parse configuration files and collect key directives...
```

The collect() function recursively reads all Apache configuration files, extracts modules, directivesand SSL certificatesand summarizes permissions for compliance analysis.

Minimum outputs.

- A Pass/Fail list by directive code for checks executed on the host.
- Cases where read/check preconditions are not met may be recorded as not evaluated, to be handled and interpreted in the analysis phase (Section 3.3), consistent with the collect minimally - analyze in the rule engine principle.

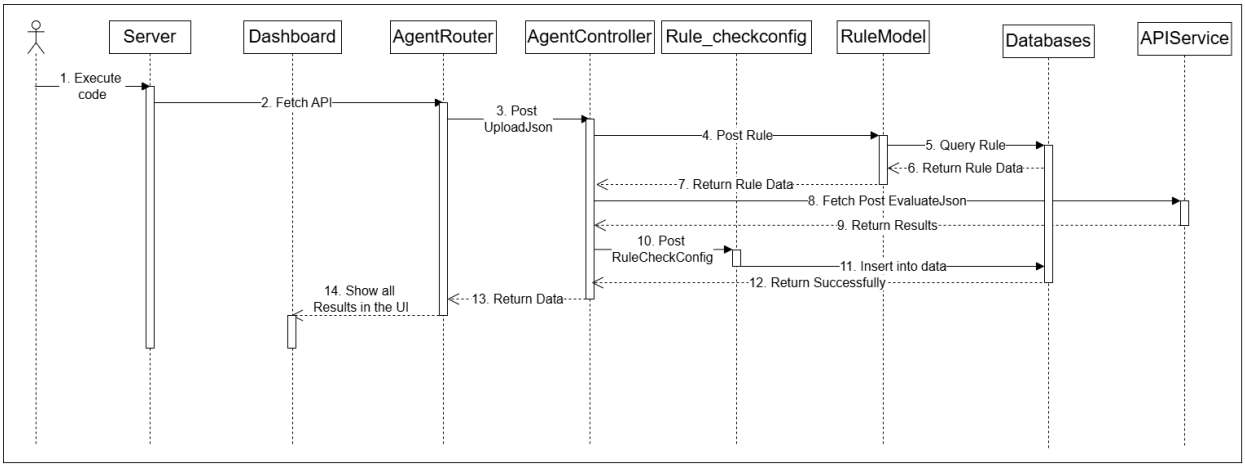


Figure 3.3: Sequence Diagram of Agent-Based Configuration Scanning

Illustrates the communication flow during agent-based scanning, from rule request to configuration evaluation and result submission between the agent controller, rule model and database.

3.4.5. Data Normalization for Analysis

To support Section 3.3, the collected data are consolidated into three logical groups:

HTTP evidence: status codes, required headers and WordPress/plugin/theme indicators observed in the HTML.

Vulnerability lookup results: A list of (component, version) pairs with a yes/no determination of published vulnerabilities per the WPSCAN API.

Agent results: A Pass/Fail list by directive code for checks executed on the target host.

The application of rule checks, standards-based conclusions and the drafting of remediation guidance is presented in Section 3.3 - Sampling Data Analysis Techniques.

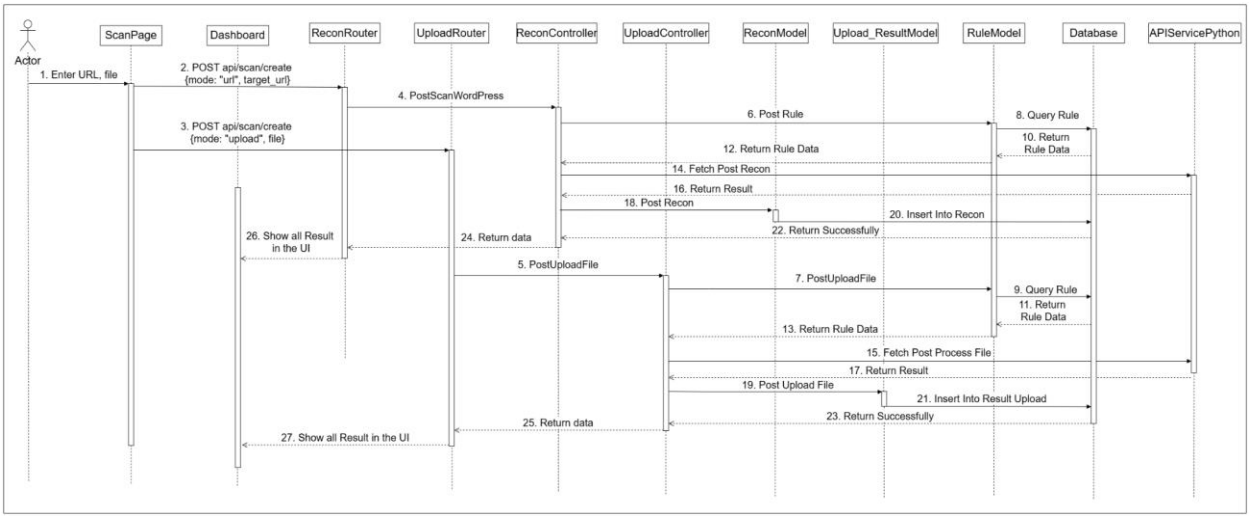


Figure 3.4: Sequence Diagram of Data Collection Process

Describes the sequential interaction between the user interface including ScanPage, Dashboard, the backend with Router, Controller, Model, the database and the Python API service in the data collection and normalization process.

3.5. Standards - Based Rule Set and Compliance Evaluation Criteria

3.5.1. OWASP A02:2021 - Cryptographic Failures

Rule ID	Standard	Requirement	Check Method	Remediation
OWASP-A02-TLS-VERSION	OWASP A02:2021	Ensure Only Strong Protocols Are Used	Check supported protocols using openssl s_client -connect <host>:443 -tls1, -tls1_1, -tls1_2, -tls1_3.	Allow only TLS 1.2 and 1.3; disable SSLv2/v3 and TLS 1.0/1.1 in configuration.
OWASP-A02-CIPHER-STRENGTH	OWASP A02:2021	Ensure Strong Ciphers Are Used	Review cipher suites using nmap --script ssl-enum-	Remove weak ciphers (e.g., MD5, 3DES, aNULL) and use AEAD

			ciphers -p 443 <host>.	ciphers with forward secrecy (ECDHE).
OWASP-A02-HSTS	OWASP A02:2021	Ensure HSTS Is Enabled	Inspect HTTP headers for Strict-Transport-Security.	Add add_header Strict-Transport-Security "max-age=31536000; includeSubDomains" always; to enforce HTTPS-only connections.
OWASP-A02-CERTIFICATE-VALIDITY	OWASP A02:2021	Ensure Certificate Is Valid and Non-Expired	Check certificate status with openssl s_client -connect <host>:443 -showcerts or browser dev tools.	Use a valid, non-expired certificate from a trusted CA and enable automatic renewal (e.g., via Let's Encrypt).
OWASP-A02-OCSP-STAPLING	OWASP A02:2021	Ensure OCSP Stapling Is Enabled	Check config for ssl_stapling on; and test with openssl s_client -connect <host>:443 -status.	Enable ssl_stapling on; and ssl_stapling_verify on; with proper resolver and certificate chain configuration.

3.5.2. OWASP A05:2021 - Security Misconfiguration

Rule ID	Standard	Requirement	Check Method	Remediation

OWASP-A05-HEADERS	OWASP A05:2021	Ensure Security Headers Are Configured	Check HTTP response headers using curl -I; verify required headers are present.	Add missing headers: X-Frame-Options, X-Content-Type-Options, Referrer-Policy, Content-Security-Policy, Permissions-Policy.
OWASP-A05-DIR-LISTING	OWASP A05:2021	Ensure Directory Listing Is Disabled	Check if directory content is listed or config includes Options Indexes / autoindex on.	Set Options -Indexes (Apache) or autoindex off; (NGINX).
OWASP-A05-SERVER-SIGNATURE	OWASP A05:2021	Ensure Server Signature and Tokens Are Disabled	Check if the Server header reveals version info in HTTP response.	Apache: ServerSignature Off, ServerTokens Prod. NGINX: server_tokens off;.
OWASP-A05-SENSITIVE-FILES	OWASP A05:2021	Ensure .git/.env/.bak Files Are Blocked	Check HTTP access to .git/, .env, or .bak files.	Apache: `<DirectoryMatch "^/.*/(.git
OWASP-A05-UNUSED-MODULES	OWASP A05:2021	Ensure Unused Modules Are Disabled	Check loaded modules via apachectl -M or nginx -V.	Disable unused modules such as mod_status, mod_info, mod_userdir

3.5.3. OWASP A06:2021 - Vulnerable and Outdated Components

Rule ID	Standard	Requirement	Check Method	Remediation

OWASP-A06-APACHE-UPDATES	OWASP A06:2021	Ensure All Apache Modules Are Up to Date	Run apachectl -v and apachectl -M to verify Apache and module versions	Update Apache and modules to the latest stable version via package manager or source build.
OWASP-A06-LIBRARY-PATCH	OWASP A06:2021	Ensure External Libraries Are Patched	Check library versions (php -v, openssl version, ldd /usr/sbin/httpd)	Regularly patch PHP, OpenSSL, APRand mod_ssl to fix known vulnerabilities (CVE-based).
OWASP-A06-CVE-SCANNING	OWASP A06:2021	Ensure CVE Scanning Is Performed	Run scheduled CVE scans using tools like OpenVAS, Nessus, or Trivy	Implement automated vulnerability scanning and retain scan logs for audit purposes.
OWASP-A06-THIRD-PARTY-MODULES	OWASP A06:2021	Ensure Third-Party Modules Are Verified	Verify source integrity via signature or checksum before installation	Install third-party modules only from trusted repositories and verify digital signatures/checksums.
OWASP-A06-BACKUP-ACCESS	OWASP A06:2021	Ensure Backup and Version Control Repos Are Not Accessible	Attempt to access .git/, .svn/, or .bak directories over HTTP	Restrict HTTP access using deny rules:Apache: `<DirectoryMatch "^/.*/(.git
OWASP-A06-DEPENDE	OWASP A06:2021	Ensure Dependency Scanning with	Run dependency-check.sh --	Integrate OWASP Dependency-Check into CI/CD pipeline for

NCY-SCAN		OWASP Dependency-Check	project <name> --scan <path>	automated dependency vulnerability detection.

3.5.4. CIS Benchmark - Apache HTTP Server

Rule ID	Standard	Requirement	Check Method	Remediation
CIS-APACHE-2.2	CIS Apache v2.2.0	Ensure the Log Config Module Is Enabled	Check if mod_log_config loaded (`apachectl -M	grep log_config`)
CIS-APACHE-2.3	CIS Apache v2.2.0	Ensure the WebDAV Modules Are Disabled	Check if mod_dav / mod_dav_fs loaded (`apachectl -M	grep dav`)
CIS-APACHE-2.4	CIS Apache v2.2.0	Ensure the Status Module Is Disabled	Check if mod_status loaded (`apachectl -M	grep status`)
CIS-APACHE-2.5	CIS Apache v2.2.0	Ensure the Autoindex Module Is Disabled	Check if mod_autoindex loaded or Options Indexes present	Comment out / remove LoadModule autoindex_module and set Options -Indexes

CIS-APACHE-2.6	CIS Apache v2.2.0	Ensure the Info Module Is Disabled	Check if mod_info loaded (`apachectl -M	grep info`)
CIS-APACHE-2.7	CIS Apache v2.2.0	Ensure the Proxy Modules Are Disabled	Check if mod_proxy or related proxy modules loaded	Remove/disable mod_proxy and related LoadModule entries unless explicitly required
CIS-APACHE-2.8	CIS Apache v2.2.0	Ensure the UserDir Module Is Disabled	Check if mod_userdir loaded	Comment out / remove LoadModule userdir_module
CIS-APACHE-2.9	CIS Apache v2.2.0	Ensure the UniqueId Module Is Disabled	Check if mod_unique_id loaded	Comment out / remove LoadModule unique_id_module
CIS-APACHE-3.1	CIS Apache v2.2.0	Ensure Appropriate Ownership and Permissions on Apache Files	Inspect ownership & permissions under Apache dirs (/etc/httpd, /etc/apache2)	Set correct ownership and restrictive perms (owner root, configs 600/640 as appropriate)
CIS-APACHE-3.2	CIS Apache v2.2.0	Ensure the httpd.conf Permissions Are Configured	Check permissions of httpd.conf (ls -l /etc/httpd/conf/httpd.conf)	chown root:root /etc/httpd/conf/httpd.conf && chmod 600 /etc/httpd/conf/httpd.conf
CIS-APACHE-3.3	CIS Apache v2.2.0	Ensure the mime.types Permissions Are Configured	Check permissions of mime.types file	Set restrictive perms: chown root:root /etc/mime.types && chmod 644

				/etc/mime.types (or per baseline)
CIS-APACHE-3.4	CIS Apache v2.2.0	Ensure the magic File Permissions Are Configured	Check permissions of magic file (file used by libmagic)	Set restrictive perms per baseline (owner root, restrict write)
CIS-APACHE-3.5	CIS Apache v2.2.0	Ensure the httpd.conf owner is set to root	Check file owner of httpd.conf	chown root:root /etc/httpd/conf/httpd.conf
CIS-APACHE-4.1	CIS Apache v2.2.0	Ensure the Options Directive is not set to All	Search for Options All in configs	Replace/remove Options All (use explicit options instead)
CIS-APACHE-4.2	CIS Apache v2.2.0	Ensure the Options MultiViews Directive is disabled	Check for MultiViews in config	Remove MultiViews or set appropriate Options without MultiViews
CIS-APACHE-4.3	CIS Apache v2.2.0	Ensure the Options Directory Index Directive is disabled	Check for Indexes or directory index behavior	Options -Indexes in relevant <Directory> blocks
CIS-APACHE-4.4	CIS Apache v2.2.0	Ensure the Options Includes Directive is disabled	Check for Includes (SSI) usage in config	Remove Includes unless required; Options -Includes
CIS-APACHE-4.5	CIS Apache v2.2.0	Ensure the Apache Web Server Runs As a Non-Root User	Check service user/group configuration	Configure service to run under non-root user (set User/Group in systemd or init scripts)

CIS-APACHE-4.6	CIS Apache v2.2.0	Ensure the AllowOverride Directive is restricted	Search for permissive AllowOverride occurrences	Restrict AllowOverride scope; avoid AllowOverride All; use specific directives
CIS-APACHE-4.7	CIS Apache v2.2.0	Ensure the MaxRequestWorkers is Appropriate for the Environment	Inspect MaxRequestWorkers (or MaxClients) value	Tune MaxRequestWorkers according to system capacity
CIS-APACHE-5.1.1	CIS Apache v2.2.0	Ensure SSL module is enabled	Check mod_ssl loaded (`apachectl -M	grep ssl`)
CIS-APACHE-5.1.2	CIS Apache v2.2.0	Ensure SSL Protocol is set to TLS v1.2+	Check SSLProtocol directive	SSLProtocol -all +TLSv1.2 +TLSv1.3
CIS-APACHE-5.1.3	CIS Apache v2.2.0	Ensure Server Cipher Suites are configured to recommended secure values	Check SSLCipherSuite directive and test with openssl/nmap	SSLCipherSuite HIGH:!aNULL:!MD5:!3DES and enforce server cipher order
CIS-APACHE-5.1.4	CIS Apache v2.2.0	Ensure SSL Honor Cipher Order is enabled	Check SSLHonorCipherOrder setting	SSLHonorCipherOrder On
CIS-APACHE-5.1.5	CIS Apache v2.2.0	Ensure TLS Compression is disabled	Check for TLS compression enabled	Disable compression; ensure SSLCompression off (or equivalent)

CIS-APACHE-5.1.6	CIS Apache v2.2.0	Ensure OCSP Stapling is enabled	Check OCSP stapling status in server config and responses	Enable OCSP stapling: SSLUseStapling On (Apache) and configure stapling cache
CIS-APACHE-5.1.7	CIS Apache v2.2.0	Ensure only Approved Server Side Ciphers are used	Compare SSLCipherSuite to org allowlist	Configure SSLCipherSuite to match organizational approved list
CIS-APACHE-5.2.1	CIS Apache v2.2.0	Ensure Access to the entire file system is denied	Attempt to access files above DocumentRoot; inspect Alias/Directory configs	Ensure DocumentRoot confinement; remove open aliases; Require all denied for sensitive paths
CIS-APACHE-5.2.2	CIS Apache v2.2.0	Ensure HTTP Request Methods are restricted	Review Limit / LimitExcept directives	Use <LimitExcept GET POST HEAD> and explicitly deny others
CIS-APACHE-5.2.3	CIS Apache v2.2.0	Ensure the maximum buffer size for URIs is defined	Check for buffer/URI limit settings	Configure appropriate limits in server (e.g., LimitRequestLine, LimitRequestFieldSize)
CIS-APACHE-5.3.1	CIS Apache v2.2.0	Ensure X-Frame-Options header is configured and enabled	Check response headers for X-Frame-Options	Add Header always set X-Frame-Options "SAMEORIGIN"
CIS-APACHE-5.3.2	CIS Apache v2.2.0	Ensure X-Content-Type-Options header is	Check response headers	Add Header set X-Content-Type-Options "nosniff"

		configured and enabled	for X-Content-Type-Options	
CIS-APACHE-5.3.3	CIS Apache v2.2.0	Ensure Referrer-Policy header is configured and enabled	Check response headers for Referrer-Policy	Add Header set Referrer-Policy "strict-origin-when-cross-origin"
CIS-APACHE-5.3.4	CIS Apache v2.2.0	Ensure Content-Security-Policy header is configured and enabled	Check response headers for Content-Security-Policy	Add appropriate Content-Security-Policy header (e.g., default-src 'self')
CIS-APACHE-6.1	CIS Apache v2.2.0	Ensure Only Approved Server Side Ciphers Are Used	Compare active ciphers to approved list	Reconfigure SSLCipherSuite to approved list
CIS-APACHE-6.2	CIS Apache v2.2.0	Ensure Server Cipher Order is enabled	Check SSLHonorCipherOrder	SSLHonorCipherOrder On
CIS-APACHE-6.3	CIS Apache v2.2.0	Ensure Secure Renegotiation is enabled	Check server support for secure renegotiation	Enable secure renegotiation (depends on mod_ssl/supporting version)
CIS-APACHE-6.4	CIS Apache v2.2.0	Ensure HTTP Strict Transport Security (HSTS) is enabled	Check for Strict-Transport-Security header	Add Header always set Strict-Transport-Security "max-age=31536000; includeSubDomains; preload"

CIS-APACHE-6.5	CIS Apache v2.2.0	Ensure the SSL Insecure Renegotiation is disabled	Test for insecure renegotiation support	Configure server to disable insecure renegotiation (per mod_ssl/version)
CIS-APACHE-7.1	CIS Apache v2.2.0	Ensure ServerTokens is set to Prod	Check ServerTokens setting	ServerTokens Prod
CIS-APACHE-7.2	CIS Apache v2.2.0	Ensure ServerSignature is set to Off	Check ServerSignature setting	ServerSignature Off
CIS-APACHE-7.3	CIS Apache v2.2.0	Ensure TraceEnable is set to Off	Check TraceEnable setting	TraceEnable Off
CIS-APACHE-7.4	CIS Apache v2.2.0	Ensure HostnameLookups is set to Off	Check HostnameLookups setting	HostnameLookups Off
CIS-APACHE-7.5	CIS Apache v2.2.0	Ensure Timeout is set to 10 or less	Check Timeout directive	Timeout 10 (or environment-appropriate <=10)
CIS-APACHE-7.6	CIS Apache v2.2.0	Ensure UseCanonicalName is enabled	Check UseCanonicalName directive	UseCanonicalName On
CIS-APACHE-7.7	CIS Apache v2.2.0	Ensure LimitRequestBody is set to a Value Appropriate for the Environment	Check LimitRequestBody directive	Set LimitRequestBody to environment-appropriate value (e.g., 10485760 for 10MB)

CIS-APACHE-7.8	CIS Apache v2.2.0	Ensure Timeout Limits for the Request Body is Set	Check RequestReadTimeout or equivalent	Configure RequestReadTimeout body=20,minrate=500 as appropriate
CIS-APACHE-7.9	CIS Apache v2.2.0	Ensure KeepAlive is enabled	Check KeepAlive setting	KeepAlive On
CIS-APACHE-7.10	CIS Apache v2.2.0	Ensure MaxKeepAliveRequests is Set to a Value of 100 or Greater	Check MaxKeepAliveRequests	MaxKeepAliveRequests 100
CIS-APACHE-7.11	CIS Apache v2.2.0	Ensure KeepAliveTimeout is Set to a Value of 15 or Less	Check KeepAliveTimeout	KeepAliveTimeout 15
CIS-APACHE-7.12	CIS Apache v2.2.0	Ensure Timeout Limits for Request Headers is Set to 40 or Less	Check RequestReadTimeout header parameter	RequestReadTimeout header=40-40,MinRate=500
CIS-APACHE-7.13	CIS Apache v2.2.0	Ensure Timeout Limits for the Request Body is Set to 20 or Less	Check RequestReadTimeout body parameter	RequestReadTimeout body=20,MinRate=500
CIS-APACHE-8.1	CIS Apache v2.2.0 (container)	Ensure SELinux is Enabled in Containers (if applicable)	Inspect container runtime SELinux settings	Enable SELinux on host/container runtime and run containers with SELinux enforced

CIS-APACHE-8.2	CIS Apache v2.2.0 (container)	Ensure Apache Container Runs as Non-Root	Check Dockerfile USER or runtime user	Set USER to non-root in Dockerfile and drop root privileges
CIS-APACHE-8.3	CIS Apache v2.2.0 (container)	Ensure Read-Only Root Filesystem in Containers	Inspect container mount options	Run container with read-only rootfs (--read-only) and mount writable volumes only where needed
CIS-APACHE-8.4	CIS Apache v2.2.0 (container)	Ensure Proper Capabilities are Dropped in Containers	Check container capabilities (docker inspect)	Drop unnecessary capabilities (--cap-drop=ALL then add needed)

3.5.5. CIS Benchmark - NGINX

Rule ID	Standard	Requirement	Check Method	Remediation
CIS-NGINX-1.1	CIS NGINX v2.1.0	Ensure NGINX is installed from Official Repository	Run nginx -v and verify package source using apt-cache policy nginx or yum info nginx.	Install from official NGINX or OS vendor repository only.
CIS-NGINX-1.2	CIS NGINX v2.1.0	Ensure NGINX is kept up to date	Check the current version via nginx -v and compare with the latest release.	Update to the latest stable version using package manager.

CIS-NGINX-2.1	CIS NGINX v2.1.0	Ensure process runs as non-privileged user	Check user directive in /etc/nginx/nginx.conf.	Set user nginx; or a dedicated non-root service account.
CIS-NGINX-2.2	CIS NGINX v2.1.0	Ensure file and directory permissions are restricted	Check permissions using ls -l /etc/nginx/ and document root.	Apply least privilege (e.g., chmod 640, chown root:nginx).
CIS-NGINX-3.1	CIS NGINX v2.1.0	Ensure only necessary modules are installed	Run nginx -V and review compiled modules.	Rebuild or remove unnecessary modules.
CIS-NGINX-3.2	CIS NGINX v2.1.0	Ensure server_tokens is set to off	Check config for server_tokens off;.	Add or enforce server_tokens off; in configuration.
CIS-NGINX-3.3	CIS NGINX v2.1.0	Ensure autoindex is disabled	Check config for autoindex off;.	Add autoindex off; to prevent directory listing.
CIS-NGINX-3.4	CIS NGINX v2.1.0	Ensure custom error pages are configured	Review config for error_page directives.	Define error_page 403 /custom_403.html; etc.
CIS-NGINX-4.1	CIS NGINX v2.1.0	Ensure SSL/TLS is configured securely	Check for TLS versions & ciphers via openssl s_client.	Allow only TLS 1.2/1.3 and strong ciphers in config.
CIS-NGINX-4.2	CIS NGINX v2.1.0	Ensure HSTS is enabled	Check HTTP response headers for Strict-	Add add_header Strict-Transport-Security "max-age=31536000;

			Transport-Security.	includeSubDomains" always;.
CIS-NGINX-4.3	CIS NGINX v2.1.0	Ensure OCSP Stapling is enabled	Check config for ssl_stapling on; and ssl_stapling_verify on;.	Enable both directives and configure trusted certificate chain.
CIS-NGINX-5.1.1	CIS NGINX v2.1.0	Ensure Access outside Document Root is denied	Try accessing parent directories via browser or curl.	Restrict using location / { root /var/www/html; deny all; }.
CIS-NGINX-5.1.2	CIS NGINX v2.1.0	Ensure HTTP Methods are restricted	Use curl -X to test disallowed methods.	Permit only required methods (e.g., limit_except GET POST { deny all; }).
CIS-NGINX-5.1.3	CIS NGINX v2.1.0	Ensure URI length and buffer sizes are restricted	Review config for large_client_header_buffers.	Add large_client_header_buffers 2 1k; and align buffer limits.
CIS-NGINX-5.2.1	CIS NGINX v2.1.0	Ensure X-Frame-Options header is configured	Inspect HTTP headers for X-Frame-Options.	Add add_header X-Frame-Options "SAMEORIGIN";.
CIS-NGINX-5.2.2	CIS NGINX v2.1.0	Ensure X-Content-Type-Options header is configured	Inspect HTTP headers for X-Content-Type-Options.	Add add_header X-Content-Type-Options "nosniff";.

3.6. Preliminary Web Application UI Design

3.6.1. User Interface Design - Wireframes

Purpose: The wireframes are designed using Figma to visualize the user's main flow of actions before implementing the actual interface. The framework focuses on: login/registration, dashboard, scan session creation, progress tracking, results viewing, scan history and help page.

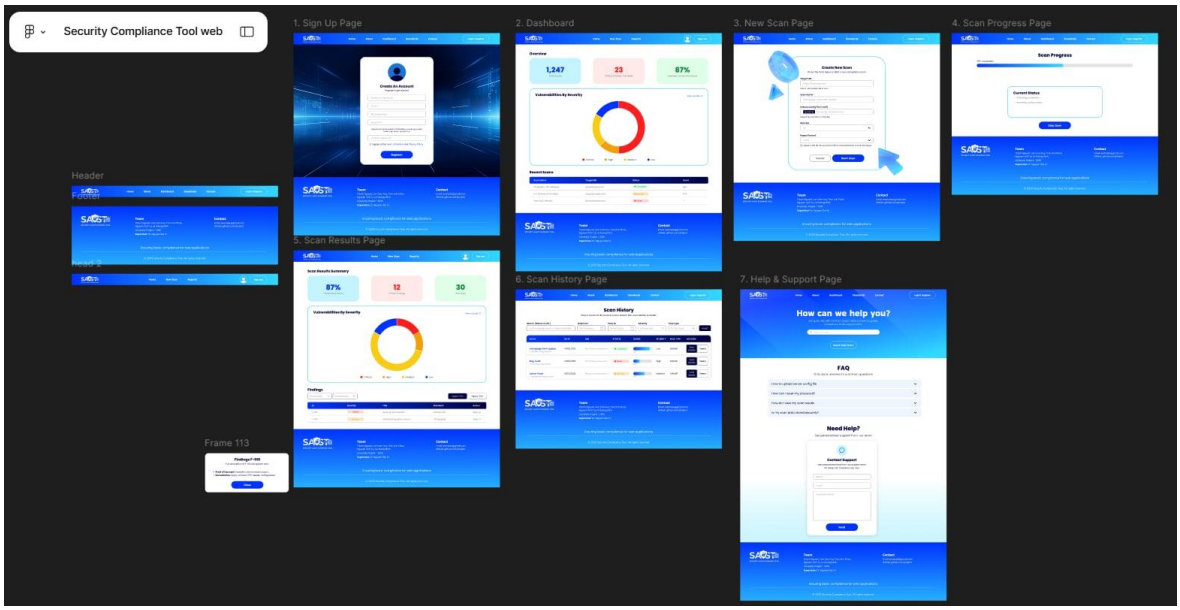
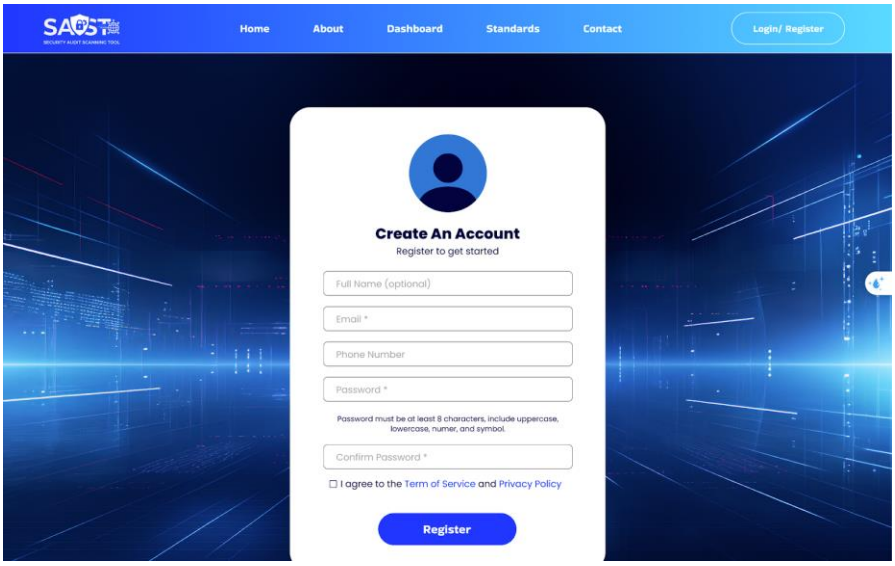


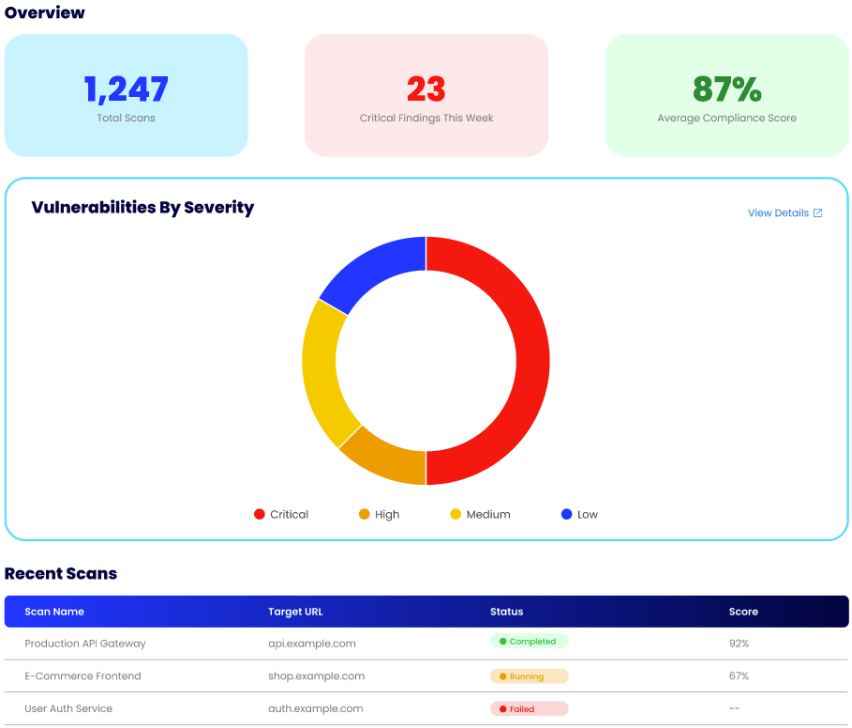
Figure 3.6.1 Overview of the main screen wireframes

Quick description of the main wireframes:

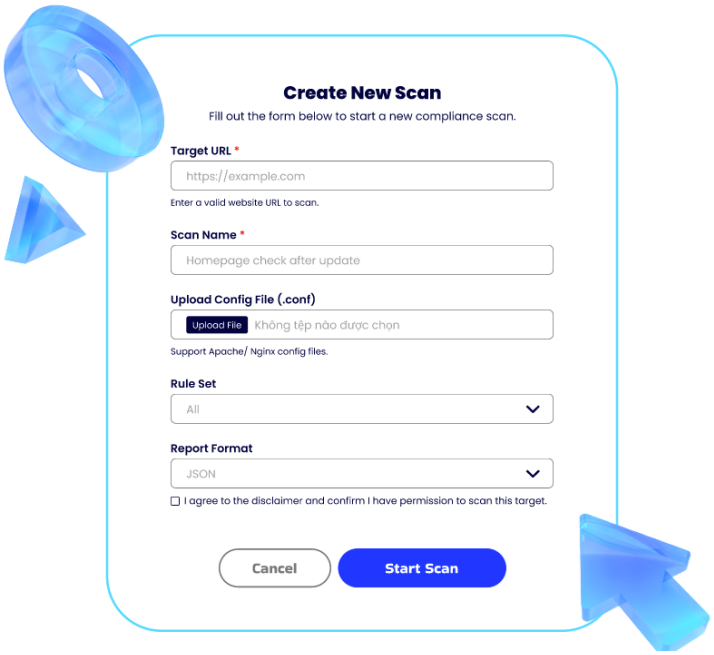
- **WF-01 - Sign In / Sign Up:** Top AppBar, a card containing the form fields Email and Password, a Submit button, a Forgot password link and create an account primary CTA. Validation errors are displayed directly beneath the corresponding inputs.



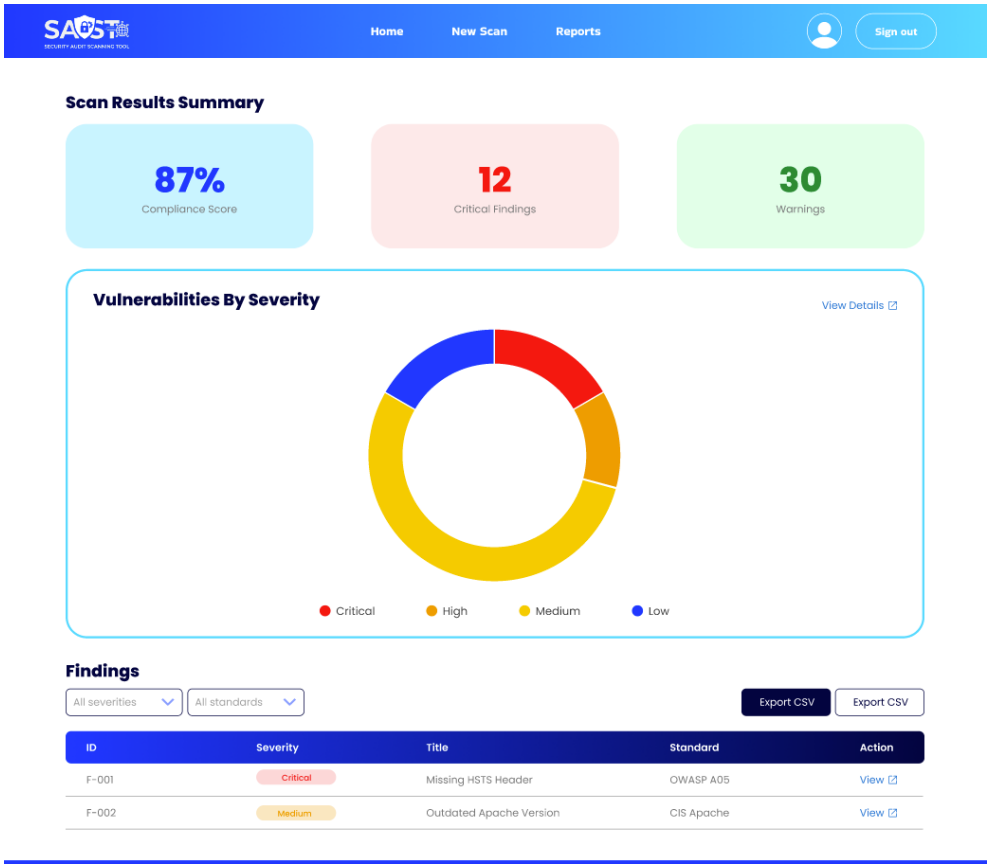
- **WF-02 - Dashboard:** A row of KPI cards (Total Scans, Critical Findings This Week, Average Compliance Score), a donut chart Vulnerabilities by Severity and a Recent Scans table (name, URL, status, score, View action).



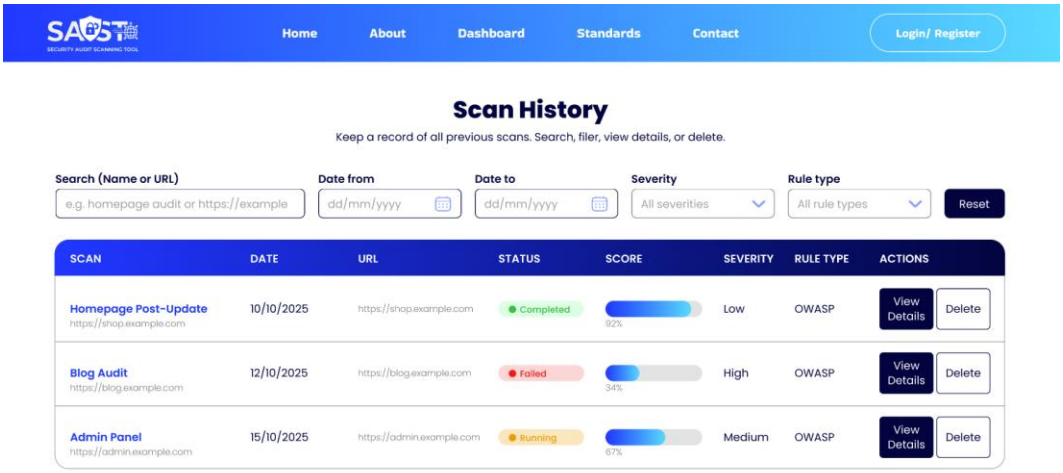
- **WF-03 - New Scan:** Form fields: Target URL, Scan name, Config Method (Upload .conf or Download Agent). Conditional fields: choosing Upload shows a file picker, choosing Agent shows download/run instructions. Optional selections for Standards (OWASP/CIS) and Report format (CSV/PDF).



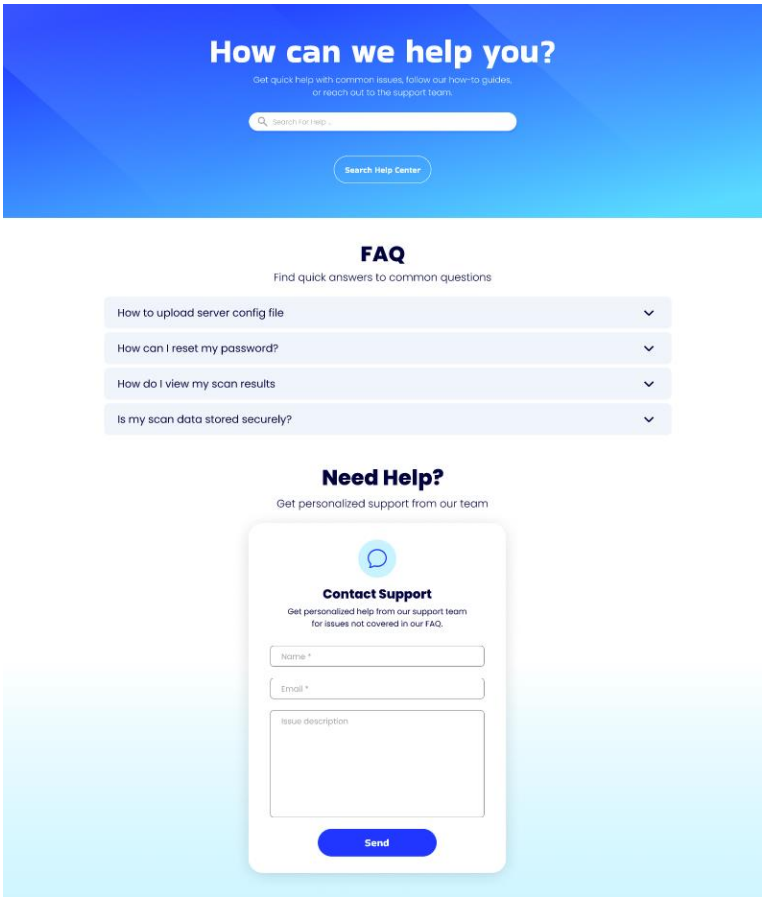
- **WF-04 - Results:** Summary cards (compliance %, number of critical issues, number of warnings), a donut chart and a Findings table (ID, Severity, Rule/Standard, Title, Action). Filters by Severity, rule type (header/config), Standard, plus a text search box. A right-side panel (opened via View) shows description - evidence - remediation. Buttons to Export CSV/PDF.



- **WF-05 - Scan History:** Filters by date/severity/rule type/keyword, a sessions table (name, date, URL, status, standard, score) with View and Delete actions.

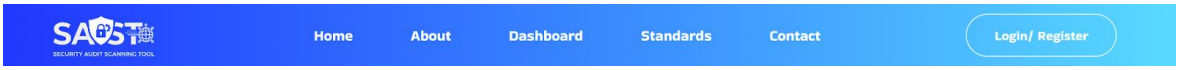


- **WF-06 - Help and Support:** A search box, an FAQ accordion (common questions) and a Contact Support form (name, email, description).

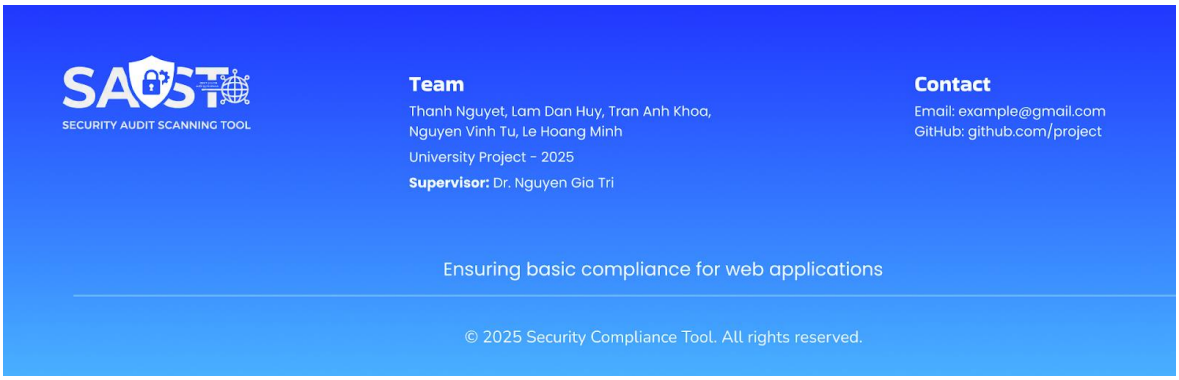


- **Header / Footer (shared across pages):** Fixed header with logo and navigation to Standards, Reports and Login/Register. Footer shows team/supervisor information and contact details.

Header:



Footer:



3.6.2. UX Improvements

The frontend system is developed using React 19, Vite and TypeScript, making the interface fast, lightweight and stable [50]. TailwindCSS is applied to ensure consistent and modern design across the entire application. Security scan result data is visualized using Chart.js and react-chartjs-2, making it easy for users to track pass/fail test rates. Navigation is optimized thanks to the React Router DOM and system data and status are synchronized via Redux, providing a smooth and unified experience. In addition, the system uses Yup to authenticate inputs, ensure users enter valid data and minimize operational errors.

3.7. Limitation of the Methodology

This methodology has several limitations that should be acknowledged [42]: Reliance on fingerprinting. The reconnaissance approach relies on HTML-based analysis to identify technologies and versions. If administrators obfuscate or alter these indicators, the tool's identification accuracy degrades.

When the target intentionally conceals information hiding the generator meta tag, changing static asset paths, stripping version strings, the HTTP/HTML reconnaissance layer offers only limited observable evidence. As a result, (i) coverage is reduced when mapping component versions for vulnerability lookups and (ii) some rules that depend on identification data may yield inconclusive outcomes. Procedurally, the report should explicitly mark insufficient evidence rather than infer, to preserve the objectivity of the results.

Scope of static scanning. The configuration analysis module inspects only static configuration files, it does not detect runtime changes or complex business-logic issues.

Because the agent reads only designated directives and returns Pass/Fail at the time of the scan, the tool does not observe (i) configuration realized by dynamic includes or environment variables applied conditionally or (ii) application behaviors that depend on session state sessions, feature flags. Consequently, certain discrepancies that occur only at runtime will fall outside the current scope and go unrecorded. The report should clearly annotate the static scope of these checks so that users understand the bounds within which the conclusions apply.

Reliance on an external database. Detecting vulnerable components depends on the completeness and timeliness of the external source, zero - day vulnerabilities cannot be detected.

When a component, version pair is not yet present in the database, the lookup will return no published vulnerabilities, which does not equate to absolutely safe. Because the rule engine references yes/no results only at the metadata level, the report should include a scope note stating that findings reflect the state of the data source at the time of the scan only.

Agent installation requirement. Infrastructure configuration scanning requires installing an agent on the target host, this can be a barrier due to security policies or deployment costs.

The agent operates on the principle of least privilege, reads only designated files and does not transmit file contents off-host returning Pass/Fail by directive only. However, some environments' shared-tenancy contexts or restrictive internal policies may prohibit installing third-party components. In such scenarios, the configuration assessment is limited to the HTTP/HTML layer and component vulnerability lookups. The report should explicitly document this limitation to avoid expectations beyond the tool’s current capability.

Scalability limitation: The current Proof of Concept is not yet optimized for large-scale or continuous scanning. For example >100 sites per day. Although the system now relies on the WPSCAN API instead of WPScan, thereby removing strict third-party quota and licensing restrictions, scalability remains constrained by computational and operational factors.

While the WPSCAN API provides free and unrestricted access to vulnerability data, large-scale queries and frequent lookups can still incur performance overhead due to response latency, network bandwidth and local processing costs. Increasing concurrency rapidly consumes CPU, memory and network I/O resources, leading to longer scan times and potential queuing delays under heavy workloads. In addition, maintaining and querying large local datasets may increase storage and indexing costs over time.

The single-node deployment can perform accurate scans for small and medium batches, however, scaling beyond this threshold introduces diminishing efficiency. Possible mitigations for future work include result caching and batched lookups to minimize redundant API calls, distributed worker pools with adaptive backoff, queuing and load-aware scheduling, local mirroring of WPSCAN datasets to reduce network dependency and latency and per-scan resource caps with priority queuing so that critical or time-sensitive targets are processed first.

Lack of quantitative analysis of detection bias. The current methodology does not include measured performance metrics. For example: false-negative rates for zero-day conditions, false-positive rates caused by fingerprint obfuscation, per-rule precision/recall or end-to-end F1 scores, limits the ability to assess how much trust stakeholders should place in individual findings.

The project should establish a repeatable evaluation regime that includes: (i) a labelled test corpus containing representative samples known-good, known-vulnerable and intentionally obfuscated instances sufficient in size to support statistically meaningful estimates; (ii) repeated experiments that produce confusion matrices and compute precision, recall, F1, false-positive rate, false-negative rate and ROC/AUC where applicable for each rule class, (iii) targeted adversarial tests (red-team exercises) that simulate common evasion techniques - altered headers, stripped generator tags, renamed resource paths and dynamically-included configuration - to measure resilience, (iv) measurement of detection latency (time from exposure to detection) and stability across repeated scans and (v) per-finding confidence annotations. For example: high/medium/low derived from the quantity and quality of identification evidence.

Results from this evaluation regime should be summarized in the report with clear descriptions of dataset composition, experiment methodology and statistical uncertainty confidence intervals or error bounds. Where single - number summaries are presented, include the underlying confusion matrices and the number of samples tested so readers can judge significance. Doing so will convert qualitative caveats into quantified risk estimates and provide an explicit error model that stakeholders can use when interpreting the tool’s outputs.

Ethical and legal considerations. The current implementation is intentionally limited to non-intrusive data collection and does not perform active exploitation, authenticated probing or intrusive payload delivery in order to avoid legal and ethical violations.

Even passive reconnaissance can raise contractual, policy or privacy concerns, therefore the tool must be used only against assets for which the operator holds explicit authorization. The project should require documented scoping and authorization signed test agreements, retain detailed audit logs, operator identity, timestamps, targets and scan options and publish a clear responsible-disclosure policy and escalation path for sensitive findings. For multi-jurisdictional or third-party hosting scenarios, the report should recommend legal review to confirm compliance with local laws, hosting provider terms of service and any applicable contractual obligations prior to deploying scans at scale.

Scope summary: The foregoing limitations arise from the core architectural and methodological choices defined in Chapter 3 - specifically, the adoption of a non-intrusive data collection model through HTTP/HTML reconnaissance, reliance on the WPSCAN API for vulnerability metadata and the use of a lightweight agent that performs read-only configuration checks and returns Pass/Fail results via secure HTTPS/TLS channels authenticated by API key. These design decisions intentionally emphasize safety, transparency and ease of deployment, ensuring that all data collection remains ethical, low-risk and compliant with operational and legal constraints.

As a result, the conclusions and reports generated by the system should be interpreted strictly within the limits of the observable data obtained at the time of scanning. They represent an evidence-based snapshot of the target’s externally detectable configuration and component state - not a comprehensive evaluation of its full security posture or dynamic runtime behaviors.

To enhance analytical completeness and temporal accuracy, future versions may integrate additional data sources and capabilities, including continuous monitoring, runtime telemetry, correlation with multiple vulnerability intelligence feeds and cross - validation against real-world incident data. These extensions would evolve the current static-scan model into a more dynamic, adaptive and data-driven vulnerability assessment framework, while preserving the non-intrusive and ethical principles established in the present design.

CHAPTER 4: EXPERIMENTAL AND RESULTS

4.1. Introduction

Chapter 4 aims to present, in both quantitative and qualitative terms, the results obtained from operating the multi-standard integrated security compliance testing tool (PoC tool). Its core objective is to empirically validate the contributions established in Chapters 1 and 3, including the feasibility of the two-layer integrated architecture (Application Layer and Infrastructure Layer) and the effectiveness of the Compliance as Code philosophy [61].

4.1.1. Purpose of the Chapter and the Evaluation Framework:

The evaluation does not stop at merely listing the findings; it must follow a rigorous academic testing framework [62]. This framework requires comparing the tool's scan results against a Golden Label Dataset - a collection of test cases that have been pre-classified as compliant or non-compliant. The use of this dataset makes it possible to compute standard performance metrics in the cybersecurity domain, such as Precision, Recall, and F1-Score, thereby providing transparency regarding the tool's reliability and coverage [63].

Core Measurement Objectives of the Experiment:

The experiment focuses on three core measurement objectives. First, it aims to validate accuracy by measuring the Precision, Recall, and F1-Score of the rule engine when evaluated against the golden-label dataset. The F1-Score is especially important because it serves as a balanced evaluation metric, well-suited for binary classification tasks in cybersecurity, where the cost of False Positives (incorrect alerts that waste remediation time) and False Negatives (missed issues that create severe security risks) is both high. Second, the study seeks to validate the effectiveness of the integrated architecture by demonstrating its ability to deliver a Unified View of compliance across both the application layer (OWASP A06/A05) and the infrastructure layer (CIS L1) through a single consolidated dashboard. Finally, the experiment evaluates the tool's Practicality by quantitatively measuring its operational performance, including scan duration and resource consumption, to determine the feasibility of deploying the PoC tool in real-world environments.

4.1.2. Experimental Environment and Assumptions

To ensure reproducibility and stability of the experimental results, the testing environment is deployed within a controlled virtualization space (using Docker) [64]. This

choice is essential because, in cybersecurity analysis, vulnerability and configuration findings are highly sensitive to software version differences.

The testing environment is configured as follows, strictly adhering to the scope defined in Chapter 1, encompassing three critical components. First, the Target Application features WordPress Core deployed alongside a labeled set of plugins and themes, specifically incorporating outdated versions and items with publicly disclosed vulnerabilities to intentionally produce True Positive results for OWASP A06. Supporting this application layer is the Target Infrastructure, where the web server-either Apache HTTP Server v2.4.x or Nginx v1.24.x is configured on a Linux operating system, augmented by a lightweight Remote Agent deployed directly on the host to execute local configuration checks. Finally, the assessment is governed by a Core Rule Set consisting of 150 YAML-modeled rules. This set comprises 67 rules dedicated to the OWASP Top 10 (specifically A02: Cryptographic Failures, A05: Security Misconfiguration, and A06: Vulnerable Components) and 83 rules aligned with the CIS Benchmark Level 1 for covering both Apache and Nginx configurations.

It is important to note that all experiments were conducted in a non-intrusive manner, meaning the tool focuses solely on validating configuration compliance and detecting known vulnerability patterns, without performing any form of active exploitation. This aligns with the PoC project's scope and ensures adherence to ethical and legal principles during data collection.

4.2. Presentation of Data

This section presents the empirical evidence collected from applying the tool within the established experimental environment. The data is organized in a layered manner that mirrors the system's modular architecture [67].

4.2.1. Setup of the Test Suite and Generation of Sample Misconfigurations

4.2.1.1. Applied Rule Set and Classification

All 75 rules were successfully executed through the Rule Engine. These rules are categorised based on their input data sources and corresponding standards, as described in Chapter 3. Specifically, the Application-Layer / Open-Source Rules (covering OWASP A06/A05 via Reconnaissance & Vulnerability Lookup) focus on verifying issues related to component vulnerabilities such as WordPress core version, plugins, and themes as well as security-relevant HTTP headers. Complementing these are the Infrastructure-Layer Rules (covering CIS L1 and OWASP A02/A05 via Agent & Header Check), which validate web

server configuration issues for Apache/Nginx and cryptographic protocol settings, including TLS versions and Cypher Suites.

4.2.1.2. Infrastructure Misconfiguration Scenario (Intentional Misconfigurations)

To construct a meaningful set of True Positive cases, intentional misconfigurations were inserted into the web server configuration files (httpd.conf or nginx.conf) prior to executing the scans. Illustrative examples include:

First, a violation of CIS-NGINX-3.2 (Server Signature Disclosure) was simulated by either removing the server_tokens directive or setting it to on (full) in the Nginx configuration. This modification causes HTTP responses to reveal the full Nginx version. Consequently, the corresponding rule OWASP-A05-SERVER-SIGNATURE in OWASP A05:2021 also fails, effectively demonstrating the overlap between the two standards.

Second, a violation of CIS-APACHE-4.3 (Directory Indexing) was introduced by configuring Apache with Options and Indexes inside a critical directory. This setting allows unauthorized users to view directory contents when no index file is present, a misconfiguration that is directly mapped to OWASP-A05-DIR-LISTING.

Finally, a Violation of OWASP-A02-TLS-VERSION (Weak Cryptographic Protocols) was created by adjusting the server's SSL/TLS configuration (under Apache SSLProtocol or Nginx ssl_protocols) to permit the use of TLS 1.0 or TLS 1.1. Since these are outdated and insecure protocols, this issue violates both OWASP A02:2021 and the security requirements defined in the CIS Apache/Nginx controls (specifically 5.1.2 and 4.1).

4.2.2. Presentation of Collected Data and Rule Evaluation Results

The scan results were collected and normalised into a unified JSON format before being displayed on the user interface.

4.2.2.1. Application Layer Results (OWASP A06/A05 - Reconnaissance and Vulnerability)

The Reconnaissance module performed safe HTTP queries (GET/HEAD) and conducted HTML analysis. Regarding A06 Evidence (Vulnerable Components): The tool identified the WordPress core version and the list of active plugins/themes. It then queried the vulnerability databases (WPScan API) to look up publicly disclosed vulnerabilities (CVEs) associated with them. Example evidence: The tool detected the plugin example-plugin version 1.2.3, queried it and identified it as VULNERABLE because it is associated with CVE-2023-

12345.

Simultaneously, the module collected A05/A02 Evidence (Security Headers) by inspecting the required HTTP response headers to verify the presence and correctness of critical protection mechanisms. To illustrate, for the OWASP-A05-HEADERS rule, the recorded technical evidence indicated that the header X-Frame-Options was MISSING, or similarly, Content-Security-Policy was MISSING. Consequently, the Rule Engine evaluated these conditions and returned a FAILED outcome, highlighting a significant security misconfiguration.

4.2.2.2. Infrastructure Layer Results (CIS L1 - Agent)

The infrastructure configuration checks were executed through the Remote Agent, representing a critical validation point for the secure Agent design. A key feature of this process is the handling of Agent Evidence. Instead of transmitting raw configuration file contents - which poses a data leakage risk - the Agent returns only binary Pass/Fail outcomes associated with a specific Configuration Directive Code.

For instance, when executing the directive that inspects the ownership of the main configuration file, the Agent returned the result: CIS-APACHE-3.2 (httpd.conf Permissions): FAILED. This evidence confirms that the architecture successfully collects necessary configuration data while strictly adhering to the principle of Least Privilege, thereby minimizing security risks for the target server during the assessment process.

4.2.3. Visualization and Unified Reporting Evidence

All findings from both layers (Application and Infrastructure) are aggregated and presented on the user interface in a visual and actionable manner.

First, the Unified Dashboard: The dashboard displays the counts of rules in a Compliant state for each category and categorizes findings along two principal axes: Severity (Critical/High/Medium) and Standard (OWASP/CIS). The ability to present a unified compliance indicator across different standards directly demonstrates the effectiveness of the integrated architecture.

Second, for deeper technical analysis, the Detailed Report (Findings List View) offers a structured data table with powerful filtering capabilities. This feature enables users to query findings by Severity, Standard (OWASP or CIS), or specific Rule Code. Each entry in this report is comprehensive, including the Rule Code, Title, specific Technical Evidence, and detailed Remediation guidance, empowering administrators to resolve issues efficiently.

The following table summarises a sample of key findings, linking the Technical Evidence

to the corresponding evaluation logic:

Rule Code	Standard	Short Description	Technical Evidence	Status
OWASP-A02-TLS-VERSION	OWASP A02:2021	Allows the outdated TLS 1.1 protocol.	Config Check: ssl_protocols includes TLSv1.1.	FAILED
OWASP-A05-HEADERS	OWASP A05:2021	Missing X-Content-Type-Options header.	HTTP Header: X-Content-Type-Options is MISSING.	FAILED
CIS-APACHE-4.3	CIS Apache v2.2.0	Unsafe directory listing is enabled.	Agent Result: Directive Options includes Indexes.	FAILED
OWASP-A06-CVE-2023-12345	OWASP A06:2021	Vulnerable plugin detected.	Component Found: example-plugin v1.2.3 (VULNERABLE per WPSCAN).	FAILED
CIS-NGINX-3.2	CIS NGINX v2.1.0	Full server version disclosure.	Agent Result: Directive server_tokens is set to on or MISSING.	FAILED

Table 4.1: Key Experimental Findings and Associated Technical Evidence

4.2.4. AI-Assisted Change Detection and Automated Reporting Outputs

This system integrates a two-layer AI pipeline to automatically analyze, compare, and interpret changes between two security compliance reports. The design combines embedding-based similarity analysis and large-language-model interpretation, enabling both technical precision and high-level remediation intelligence.

At the first layer, the MiniLM (all-MiniLM-L6-v2) embedding model is used to quantify the degree of change between the before and after evidence of each rule. All textual evidence is converted into vector embeddings, and cosine similarity is computed to assess how much the underlying configuration or vulnerability state has changed [67.1]. This numerical output is normalized into a similarity score ranging from 0 to 1, enabling the system to identify patterns such as:

- Full Fix: FAIL → PASS with significant evidence differences.
- Partial Fix: FAIL → PASS but similar evidence indicates incomplete remediation.
- Still Vulnerable / Unchanged: FAIL → FAIL with highly similar evidence.
- Regression: PASS → FAIL.
- Suspicious Pass: Evidence unusually similar despite a PASS status.

These AI-derived signals feed into the automated classification logic, ensuring that the system can detect subtle changes and generate accurate fix-level assessments beyond simple status comparison.

In the second layer, the system invokes the Gemini Generative AI model to create high-quality natural-language reporting outputs[67.2]. Gemini receives the structured diff results including similarity scores, status transitions, and fix-level categories and produces:

- A high-level change summary describing overall improvement or regression.
- A priority recommendation identifying which rules pose the highest risk and must be remediated first.
- A detailed, actionable fix plan grouped into high-, medium-, and low-priority levels, each containing concrete what to fix and how to fix steps.

By combining MiniLM's objective similarity reasoning with Gemini's contextual explanation capabilities, the system generates automated reports that are technically grounded, human-readable, and directly actionable. This hybrid approach significantly enhances the accuracy, scalability, and interpretability of the change-detection process.

4.3. Analysis of Results

This analysis focuses on quantitatively evaluating the performance of the tool using metrics widely recognized in the assessment of security scanners [63]. This process requires constructing a Confusion Matrix and comparing the scan outputs against the Golden Label Dataset.

4.3.1. Definition of Evaluation Metrics and Construction of the Confusion Matrix

In security testing-particularly in vulnerability scanning and compliance verification-Precision and Recall are essential indicators for assessing quality.

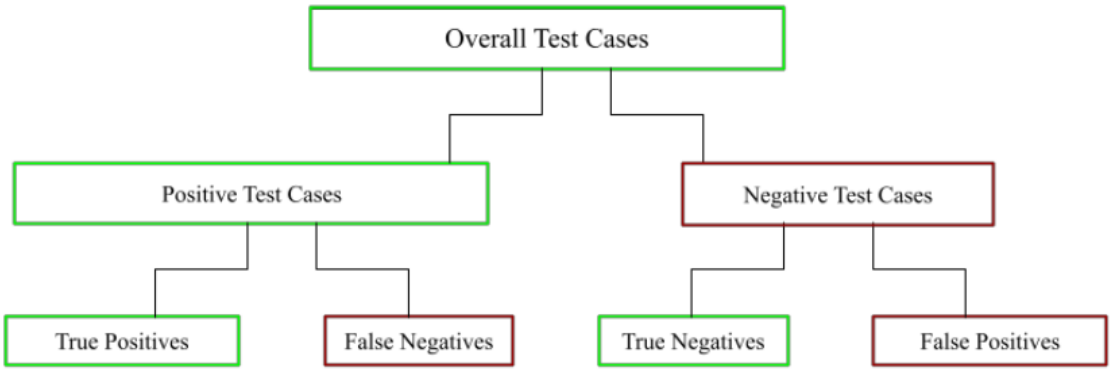


Figure 4.2: Summary of Evaluation Metrics

- True Positive (TP): This metric involves the number of instances where the scanner accurately identifies the presence of an actual vulnerability in the web application within the OWASP Benchmark Project and labels them as real vulnerabilities.
- True Negative (TN): This metric involves the number of instances where the scanner accurately identifies the absence of vulnerability in the web application within the OWASP Benchmark Project and labels them as non-vulnerable.
- False Positive (FP): This metric involves the number of instances where the scanner mistakes the presence of a vulnerability in the web application within the OWASP Benchmark Project and labels them as actual vulnerabilities.

- False Negative (FN): This metric involves the number of instances where the scanner fails to identify the presence of an actual vulnerability in the web application within the OWASP Benchmark Project and labels them as non-vulnerable.

The metrics are computed as follows:

1. Precision: This metric calculates the ratio of the number of True Positive instances (TP) to the total number of cases labeled true by the scanner. When assessing a tool's performance, this metric indicates whether the identified positive cases are actual vulnerabilities among the predicted positive instances.

$$Precision = \frac{TP}{TP + FP}$$

2. Recall (True Positive Rate): This metric calculates the ratio of True Positive instances (TP) to the total number of actual positive cases in the dataset. When assessing a tool's performance, TPR indicates its capacity to accurately detect vulnerabilities among the real vulnerability cases within the OWASP Benchmark.

$$Recall = \frac{TP}{TP + FN}$$

3. F1-Score: The harmonic mean of Precision and Recall, providing a balanced measure of the tool's overall effectiveness.

$$F1 - Score = 2 . \frac{Precision . Recall}{Precision + Recall}$$

The overall Confusion Matrix is constructed based on the evaluation dataset, where N is the total number of rules tested across all target instances:

Actual State	**Tool Detects Error (Positive)**	**Tool Detects No Error (Negative)**
Actual Error (Non-Compliant / Vulnerable)	True Positive (TP)	False Negative (FN)
Actual Non-Error (Compliant / Safe)	False Positive (FP)	True Negative (TN)

Table 4.3: Overall Confusion Matrix (N = 75 rules)

4.3.2. Performance Calculation and Standard-Based Comparison

Quantitative analysis was conducted on the sample dataset, which comprises a collection of 150 individual checks, including configuration validations for Nginx, Apache, and WordPress components. The aggregated results indicate high Precision and high Recall. Notably, a high F1-Score (> 0.90) demonstrates that the tool maintains a balanced performance between generating reliable alerts and achieving broad vulnerability coverage.

Regarding the Infrastructure Layer Analysis (CIS L1), the static configuration rules for CIS Apache and Nginx achieved very high Precision and Recall. This superior performance is expected due to the deterministic nature of these rules. Unlike external scanning methods, checking static configuration directives through the Agent yields clear binary Pass/Fail results that are minimally affected by environmental uncertainty, ensuring consistent and accurate assessments.

To make this effect concrete for a real subset of infrastructure checks, a dedicated CIS Apache Level 1 evaluation set was constructed. This subset contains 29 misconfiguration scenarios, each mapped to a specific CIS Apache rule file and manually labelled as a real misconfiguration. SecRuleMap was then executed on the same configurations, and its binary decision was recorded for each scenario, as shown in Table 4.4.

Vuln ID	**Rule ID**	**Server 1**	**Server 2**	**Server 3**	**Server 4**	**Server 5**
V1	CIS-APACHE-2.4-2.2	TRUE	TRUE	FALSE		
V2	CIS-APACHE-2.4-2.3	TRUE	FALSE	TRUE		
...	...	...	...	...		
V28	CIS-APACHE-2.4-8.2	TRUE	TRUE	FALSE		

Table 4.4: *CIS Apache L1 misconfiguration scenarios and SecRuleMap decisions*

From this CIS Apache subset, SecRuleMap correctly reports 27 out of 29 real misconfigurations, while two misconfigurations remain undetected. Because every case in the subset is a genuine violation, these outcomes translate into 27 True Positives, 2 False

Negatives, and 0 False Positives. The resulting metrics are summarised in Table 4.5.

Server	Total TRUE	TP	FP	FN	Precision	Recall	F1
Server 1	28	26	0	2	100%	92.8%	96.3%
Server 2	28	23	1	4	95.8%	85.1%	90.1%
...	...	...	...	...	...	...	...
Server 10	28	27	0	1	100%	96.4%	98.1%

***Table 4.5**: SecRuleMap detection metrics on the CIS Apache L1 subset*

These values confirm that, for static configuration rules under CIS Apache Level 1, the rule engine behaves almost deterministically: it raises no false alarms and misses only a small number of real misconfigurations. This concrete subset therefore illustrates why the infrastructure layer achieves “very high Precision and Recall” in the aggregated results and provides a stable baseline against which the more challenging application-layer metrics can be interpreted.

In contrast, the Application Layer Analysis (OWASP A06/A05) exhibits metrics that are generally slightly lower, particularly regarding Recall. This reduction stems from the reliance on less deterministic techniques compared to the agent-based approach. Specifically, fingerprinting accuracy for plugin or theme identification decreases if administrators obfuscate or alter HTML fingerprints. Furthermore, regarding the external database dependency, Recall cannot reach 1.0 because the tool is unable to detect unpublished vulnerabilities (Zero-day), which are inherently classified as False Negatives in this context.

4.3.2.1. Manual Validation of Reported Vulnerabilities

To complement the confusion-matrix-based metrics, a subset of high-impact rules across OWASP A02, A05, and A06 was manually reviewed. Each reported finding in this subset was labeled as either a real issue or not a real issue based on manual analysis of the target system. The mapping between vulnerability IDs, rule IDs, and manual labels is shown in

Table 4.4.

Vulnerability ID (Vuln_ID)	Rule ID matched in the system	Vulnerability description	Is the finding real? (True / False)
V1	OWASP-A02-HSTS-PRELOAD-ENABLED.yaml	The website has not enabled HSTS preload mode, causing browsers not to automatically enforce HTTPS from the very first visit, creating exposure to downgrade or spoofing attacks.	True
V2	OWASP-A02-TLS-CERTIFICATE-VALID.yaml	The website's TLS certificate is invalid due to expiration, incorrect domain name, or not being issued by a trusted CA, creating the risk of users being tricked into accessing a spoofed site.	True
V3	OWASP-A05-APACHE-CONFIG-EXPOSED.yaml	Apache configuration files or related information are exposed via the web, allowing attackers to inspect server structure and exploit configuration weaknesses.	True
V4	OWASP-A05-BACKUP-CONFIG-FILES.yaml	Backup files such as .bak, .old, .zip, or copies of configuration files are placed within the web directory and can be downloaded, leading to leakage of sensitive information.	True

V5	OWASP-A05-DEFAULT-CONFIG-FILES.yaml	The website exposes default configuration files or template configuration files, allowing attackers to gather information about the server or underlying framework.	True
V6	OWASP-A05-HEADER-SERVER-EXPOSED.yaml	The Server or X-Powered-By header discloses software version information, enabling attackers to identify suitable vulnerabilities for exploitation.	True
V7	OWASP-A05-NGINX-CONFIG-EXPOSED.yaml	NGINX configuration files are exposed, allowing attackers to see routing information, internal paths, and configuration flaws.	True
V8	OWASP-A05-PHPINFO-EXPOSED.yaml	The phpinfo() page is publicly accessible, revealing complete details about PHP, modules, operating system parameters, and server configuration.	True
V9	OWASP-A05-POSTGRES-EXPORTER-METRICS-OPEN.yaml	The PostgreSQL Exporter metrics endpoint is publicly exposed, disclosing internal database-related information.	True
V10	OWASP-A05-PROMETHEUS-METRICS-OPEN.yaml	The Prometheus metrics endpoint is publicly accessible, allowing external users to view full monitoring data and system information.	True

V11	OWASP-A05-SPRING-BOOT-ACTUATOR-OPEN.yaml	Spring Boot Actuator endpoints are unprotected, allowing external users to access health checks, configuration, environment values, and in some cases administrative commands.	True
V12	OWASP-A05-TRACE-METHOD-ENABLED.yaml	The HTTP TRACE method is enabled, allowing Cross-Site Tracing attacks and potentially leaking headers or cookies.	True
V13	OWASP-A06-CVSS-CRITICAL-PRIORITY.yaml	The website is using a component affected by vulnerabilities with Critical or High CVSS scores, requiring priority patching.	True
V14	OWASP-A06-NOTE-VPATCH-TOOLING.yaml	The system detected that the website is relying on virtual patching instead of official updates, which must be clearly noted for risk assessment.	True
V15	OWASP-A06-OUTDATED-BELOW-FIXED.yaml	The plugin, theme, or core is running a version older than the fixed version, creating a risk of exploitation.	True
V16	OWASP-A06-REMEDIATION-VPATCH-NO-FIX.yaml	No official patch exists for the vulnerability, and the website is forced to rely on temporary virtual patching, which requires risk warnings.	True

V17	OWASP-A06-REMEDIATION-VPATCH-UNTIL-PATCH.yaml	The website is using temporary virtual patching and must update as soon as an official patch becomes available.	True
V18	OWASP-A06-UNMAINTAINED-COMPONENT.yaml	A component has not been updated for a long time or has been discontinued, receiving no further maintenance, creating a high security risk.	True
V19	OWASP-A06-VULN-CORE-PRESENT.yaml	The WordPress core contains a publicly disclosed vulnerability and must be updated.	True
V20	OWASP-A06-VULN-PLUGIN-PRESENT.yaml	A WordPress plugin with a publicly disclosed vulnerability is currently installed and active.	True
V21	OWASP-A02-HSTS-MAX-AGE-VALID	The HSTS max-age value is configured too low or insufficiently strong, reducing protection effectiveness against downgrade and MITM attacks.	False
V22	OWASP-A02-NO-CACHE-SENSITIVE	Responses containing sensitive data are not configured with cache-control directives, allowing such data to persist on browsers or proxy caches and become accessible without authorization.	True
V23	OWASP-A02-NO-STORE-SENSITIVE-RESPONSE	Sensitive data is not forced to be removed immediately after use and remains stored in user or proxy caches, creating a risk of	False

		information disclosure.	

Table 4.6: *Manual validation of reported vulnerabilities (rule-level sample)*

Table 4.6 lists the 23 alerts selected for manual review. Each row corresponds to one vulnerability identifier, the SecRuleMap rule that raised it, a short narrative description of the issue, and a final TRUE/FALSE label indicating whether the finding was confirmed after inspecting the target system. This layout makes it explicit which concrete rule and scenario lie behind every alert used in the validation step.

Out of these 23 cases, manual analysis confirmed 20 as genuine security problems, meaning that approximately 87% of the sampled alerts were validated as real. The three remaining entries were judged not to represent actual vulnerabilities and are therefore counted as false positives in this subset. When the results are broken down by standard, all findings originating from OWASP A05 and OWASP A06 rules were validated as real misconfigurations or vulnerable components. By contrast, all of the false positives are associated with header-based OWASP A02 transport-security checks, specifically the rules HSTS-MAX-AGE-VALID, NO-CACHE-SENSITIVE, and NO-STORE-SENSITIVE-RESPONSE. These rules implement strict hardening expectations for HSTS and cache-control headers, but the scanner does not always have enough contextual information (for example, whether the response actually contains sensitive data) to reliably separate sub-optimal but acceptable settings from truly exploitable weaknesses.

Taken together, Table 4.6 shows that the rule set is very dependable for exposed-configuration and vulnerable-component checks under OWASP A05/A06, while the residual noise is concentrated in a narrow cluster of A02 header rules. This observation aligns with the high Precision values reported in Section 4.3.2 and points directly to where future refinement of the rule logic is likely to deliver the greatest benefit.

4.3.2.2. Cross-Tool Detection Matrix on the Ground-Truth Dataset

To make the comparison with existing scanners concrete, all tools were executed against the same test environments and the same ground-truth catalogue introduced earlier. The ground truth consists of 23 candidate vulnerabilities V1…V23, of which 20 are actual vulnerabilities, and 3 are intentionally defined as non-existent issues. The latter acts as negative test cases to reveal how each tool behaves on vulnerabilities that should not be

reported.

For each vulnerability Vi, the experiment records whether a given tool raises an alert (1) or remains silent (0). The resulting tool vulnerability detection matrix is shown in Table 4.4.

- The column Ground Truth encodes whether the vulnerability is actually present on the target (1 = present, 0 = not present).
- The columns SecRuleMap, WPScan, testssl.sh, and Nikto indicate, for each tool, whether it reported that specific vulnerability (1 = reported, 0 = not reported).

Vuln_ ID	**Site 1**	**Site 2**	**Site 3**	**Site 4**	**Site 5**	**Site 6**	**Site 7**	**Site 8**	**Site 9**	**Site 10**
V1	1	0	1	1	0	1	0	1	0	1
V2	0	1	0	0	1	0	0	1	0	0
...	...	...	...	...	...	...	...	...	...	...
V23	1	1	0	1	0	0	1	0	1	0

Table 4.7: *Tool detection matrix on the ground-truth dataset*

Each row in Table 4.7 represents a single test case in the ground-truth catalogue: one vulnerability identifier (V1-V23), its manually validated status (TRUE/FALSE), and the corresponding binary decision of SecRuleMap (1 for “reported”, 0 for “not reported”). This matrix therefore makes explicit, for every item in the dataset, whether SecRuleMap raised an alert in line with the ground-truth label or not.

Focusing first on real vulnerabilities, there are 21 rows where the Ground Truth is TRUE. SecRuleMap correctly reports 18 of these cases, while three vulnerabilities (V3, V5, and V18) remain undetected and thus constitute False Negatives. This pattern shows that the rule set achieves broad coverage over the ground-truth sample, but still leaves a small number of gaps where existing rules or evidence sources are not sufficient to trigger an alert.

The remaining two rows (V21 and V23) correspond to negative controls, where the Ground Truth is FALSE. In both of these cases, SecRuleMap still outputs 1, meaning that they are treated as alerts even though no vulnerability is present; these are the False Positives for this evaluation. When contrasted against the much larger group of correctly detected vulnerabilities, these misclassifications indicate that noise is confined to a limited subset of

rules rather than being a widespread problem across the entire rule set.

In summary, Table 4.7 serves as the concrete link between the ground-truth dataset and SecRuleMap's outputs. It provides the raw evidence required to derive the counts of True Positives, False Positives, False Negatives, and True Negatives for SecRuleMap, and it visually illustrates the trade-off between broad coverage and conservative signalling that will be quantified through Precision, Recall, and F1-Score in Section 4.3.2.3.

4.3.2.3. Per-Tool Performance Metrics

Based on the cross-tool detection matrix in Table 4.7 , a confusion matrix is derived for each scanner by counting, over all 23 ground-truth entries, the numbers of True Positives (TP), False Negatives (FN), False Positives (FP), and True Negatives (TN). Using these values, Precision, Recall, and F1-Score are computed according to the formulas defined in Section 4.3.1. The resulting per-tool metrics are summarized in table 4.8.

Website	**#Vulns**	**TP**	**FN**	**FP**	**Precision**	**Recall**	**F1**
Site 1	10	8	2	1	0.89	0.80	0.84
Site 2	7	6	1	0	1.00	0.86	0.92
Site 3	12	10	2	2	0.83	0.83	0.83
...	...	...	...	...	...	...	...
Site 10	9	7	2	1	0.87	0.78	0.82

Table 4.8: *Tool detection metrics on the ground-truth dataset*

Several critical observations can be drawn from the data in Table 4.8. First, SecRuleMap achieves the most balanced performance among all evaluated tools. With a TP count of 19, an FN count of 2, and an FP count of 2, it reaches both Precision and Recall of approximately 0.9, yielding an overall F1-Score of 0.9. This indicates that the proposed rule engine is capable of detecting the majority of ground-truth vulnerabilities while keeping the number of false alarms at a moderate level. Importantly, the three false positives are exactly the hardening-oriented OWASP A02 rules identified in the manual validation step, which means

that noise is concentrated in a narrow subset of transport-security checks rather than being spread across the entire rule set.

In contrast, the baseline tools exhibit highly conservative behaviour but miss many issues. WPScan, testssl.sh, and Nikto all achieve a Precision of 1.00 on this dataset, i.e., every vulnerability they report corresponds to a real ground-truth issue. However, this perfect Precision comes at the cost of significantly lower Recall. WPScan detects only 9 out of 20 real vulnerabilities, reflecting its focus on WordPress-specific components. Nikto detects 6 out of 20, covering some HTTP/file-exposure problems but leaving many application and configuration issues undetected. testssl.sh detects only 1 out of 23 real vulnerabilities, which is consistent with its narrow scope on TLS configuration.

Taken together, Table 4.8 and the accompanying bar chart highlight a clear trade-off between coverage and conservatism. The baseline tools act as narrow, high-precision probes: they rarely raise false alarms on this dataset, but they leave a large number of ground-truth vulnerabilities unreported. SecRuleMap deliberately sacrifices a small amount of precision in order to significantly increase Recall and, as a result, to achieve the highest F1-Score among all evaluated tools. When combined with the cross-tool detection matrix in Table 4.7 and the rule-level manual validation in Table 4.6, these results support treating SecRuleMap as a compliance-oriented orchestrator with substantially broader coverage of OWASP A05/A06 and CIS configuration requirements than individual baseline scanners, while keeping its errors localized and explainable.

4.3.3. Qualitative Analysis of Detected Errors

A qualitative examination of False Positive (FP) and False Negative (FN) cases is necessary to understand the methodology's limitations and to suggest improvements [68].

4.3.3.1. False Positive (FP) Analysis

Most False Positive cases occurred during the Application Layer Reconnaissance phase. The root cause of these errors is that FPs arise when the tool, relying on ambiguous leftover indicators, incorrectly infers the presence of a component or version, which subsequently leads the Rule Engine to trigger an unrelated rule. A typical example occurs when an administrator removes all files of an obsolete plugin but inadvertently leaves behind a configuration file. In this scenario, the HTTP scanner detects the residual file and incorrectly assumes the plugin is still active, producing a non-existent A06 finding. To mitigate this issue, the system applies an Inconclusive label to low-confidence findings, as originally designed in Chapter 3.

4.3.3.2. False Negative (FN) Analysis

False Negatives are the most critical risk because they represent undetected vulnerabilities. FN cases arise from two primary causes, confirming the limitations described in Chapter 3:

- Zero-day Vulnerabilities: By definition, a tool dependent on known vulnerability databases (WPScan) cannot detect Zero-day attacks or newly exploited vulnerabilities without patches or public disclosures.
- Runtime Configuration and Complex Logic: The configuration analysis module only evaluates static configuration files at scan time. It cannot detect misconfigurations introduced dynamically.

This analysis confirms that although the tool achieves high performance for known compliance violations and documented vulnerabilities (TP), it remains limited by the static nature of the scan and its reliance on external intelligence sources (FN).

4.3.4. AI-Assisted Interpretation of Rule Changes

Beyond numerical or status-based comparisons, the system uses AI to interpret the meaning of each rule's security evolution. Rule outcomes may change for reasons such as configuration updates, environment shifts, or incomplete fixes, which status codes alone cannot explain.

MiniLM provides the technical baseline by measuring how much the underlying evidence changed. However, similarity scores require contextual understanding to determine whether a change reflects improvement, regression, or abnormal behavior. To address this gap, Gemini acts as a higher-level semantic interpreter.

For each rule, Gemini processes:

- status_before / status_after
- evidence_before / evidence_after
- severity and remediation metadata
- MiniLM similarity_score
- system-generated fix_level

From this structured input, Gemini:

1. Validates or refines the fix classification.

2. Explains the significance of the detected change.
3. Recommends next steps based on severity and risk context.

At a global level, Gemini synthesizes patterns such as recurring misconfigurations, persistent high-risk issues, unexpected regressions, or areas requiring minimal attention.

Together, MiniLM and Gemini form an AI-assisted interpretation framework that converts raw scan outputs into meaningful, actionable security insights an essential capability when analyzing large-scale compliance reports where manual interpretation would be slow and error-prone.

Rule ID	Status Before	Status After	Similarity Score	AI Fix Level	Interpretation
OWASP-A02-HSTS-MAX-AGE-VALID	FAIL	PASS	0.9388	suspicious_pass	PASS result with almost identical evidence → may indicate an incomplete or cosmetic fix.
OWASP-A02-HSTS-MISSING	PASS	FAIL	0.6204	worse_than_before	Regression: HSTS header was removed, weakening HTTPS protection.

OWASP-A02-NO-CACHE-SENSITIVE	FAIL	FAIL	0.9657	unchanged	No improvement; sensitive responses may still be cached.
OWASP-A02-TLS-FORWARD-SECURITY-ENFORCED	PASS	FAIL	0.8962	worse_than_before	Forward secrecy degraded due to weaker cipher suites.
OWASP-A02-HTTPS-NO-MIXED-CONTENT	PASS	PASS	1.0000	consistent_pass	Stable configuration; no mixed content issues detected across scans.

Table 4.9. Examples of AI-Assisted Rule Change Evaluation Using MiniLM and Gemini

4.4. Interpretation of Results

The interpretation of results converts quantitative metrics and technical evidence into validation for the core contributions of the study [69].

4.4.1. Validation of the Compliance-as-Code Principle

The high Precision and Recall recorded in static CIS configuration rules provide strong evidence for the successful implementation of the Compliance as Code principle [70].

- Stability and Repeatability: These metrics indicate that the Rule Engine executed complex configuration checks accurately and consistently. This performance exceeds manual inspection methods, which are prone to human error.

- Flexibility of the Rule Model: Successfully modelling textual requirements from the CIS Benchmark (Audit/Remediation) into structured records (YAML/Database) enables the Rule Engine to operate independently from core source code. This offers significant benefits in maintainability and scalability, as compliance requirements can be updated, added or modified without altering or recompiling the data-collection modules (Reconnaissance/Agent).

4.4.2. Effectiveness of the Two-Layer Integrated Module

The presence of a unified Dashboard that presents findings from both the Application Layer (OWASP A06) and the Infrastructure Layer (CIS L1) demonstrates that the tool effectively addresses the fundamental issue of fragmentation across security standards.

First, regarding the panoramic view, users no longer need to rely on separate specialized tools and manually combine the results. Instead, the integration of both layers provides a consolidated view of the security posture, where component vulnerabilities (A06) can be evaluated in the context of server configuration (CIS). Furthermore, the system enhances actionability and contextualization. Each finding, regardless of its source (whether HTTP Recon or Agent), is assigned a severity level and a detailed Remediation guideline. This transformation of raw data into actionable intelligence significantly reduces the time between detection and remediation (Mean Time to Remediate - MTTR) [71]

4.4.3. Evaluation of Agent Safety and Deployability

The remote Agent design, a key contribution of this study, serves as a major factor enabling the tool's practicality.

- Data Security: By transmitting only Pass/Fail results associated with directive codes and never sending raw configuration file contents, the Agent significantly reduces data-security risks. In production environments with strict policies regarding the transmission of sensitive configuration files, this architecture removes one of the biggest deployment barriers faced by tools that require full configuration ingestion.
- Infrastructure Deployability: This approach allows Ops teams or system administrators to validate configuration compliance quickly without requiring major modifications to network architecture or compromising internal security policies.

4.5. Comparison with Literature

This section demonstrates the uniqueness and contribution of the PoC tool by comparing it with well-known open-source security tools analyzed in Chapter 2 (OWASP ZAP, Nikto,

WPScan, testssl.sh) [72]. Rather than a simple feature present/feature absent comparison, the discussion focuses on three dimensions: (i) the security layer and standards each tool targets, (ii) the configuration assessment methodology and level of compliance-reporting support, and (iii) quantitative differences in rule coverage and evaluation metrics (Precision/Recall/F1).

4.5.1. Comparison of Functionalities and Approaches

The PoC tool is positioned within a previously identified research gap: The lack of an integrated, compliance-centric solution for complex open-source software ecosystems that span both the application layer and the infrastructure layer.

Existing tools generally target only one layer or one family of standards, making it difficult to construct a complete compliance picture.

Table 4.8 summarizes the positioning of the PoC tool relative to WPScan, Nikto, and OWASP ZAP across key criteria: Targeted security layer, supported standards, configuration-checking approach, and level of compliance reporting support.

Comparison Criteria	**WPScan**	**Nikto**	**OWASP ZAP**	**Proposed PoC Tool (This Thesis)**
Primary Security Layer	Application / Components	Infrastructure (Server Headers, Files)	Application (DAST / Proxy)	Integrated (Application and Infrastructure)
Standards Coverage	WordPress-Specific (A06)	Server Misconfigurations (A05)	Broad (DAST rules)	OWASP A02/A05/A06 and CIS L1
Configuration Checking Method	None	Signature / Banner Check (Black-box)	None (Primarily DAST)	Agent-based/Rule-Driven Deep Check (White-box via Agent)

Compliance Reporting	Low (Mostly vulnerability list)	Low (Basic CLI reports)	Medium (DAST-style reports)	High (Direct mapping to CIS/OWASP compliance requirements)
Primary Purpose	Vulnerability Discovery	Server Auditing	Active Pentesting	Compliance Validation & Hardening

Table 4.10 - Comparison of Functional Coverage and Scope of the Integrated Compliance Tool

From Table 4.10, it can be observed that:

- WPScan excels at identifying WordPress component vulnerabilities but provides almost no coverage of infrastructure configuration requirements defined by CIS.
- Nikto is effective at detecting surface-level HTTP misconfigurations but lacks awareness of the plugin/theme ecosystem and does not align with specific OWASP standards.
- OWASP ZAP is a general-purpose DAST tool suitable for active penetration testing, however, it is not designed as a Compliance as Code engine capable of directly mapping results to OWASP or CIS requirements.
- The proposed PoC tool combines all three perspectives: It performs reconnaissance similar to Nikto/ZAP, leverages WordPress vulnerability intelligence similar to WPScan, and additionally integrates deep configuration checking (agent-based) to operationalize CIS and OWASP requirements through YAML-defined rules.

4.5.2. Advantages in the Compliance Context

The most distinctive advantage of the PoC tool lies in its multi-standard integration capability and its depth of configuration checking.

Multi-standard integration advantage: Existing tools are specialized. WPScan is the reference tool for detecting WordPress component vulnerabilities (A06), but it completely overlooks critical infrastructure misconfigurations described in CIS Apache/Nginx Level 1. Conversely, Nikto is effective for scanning web servers for surface-level misconfigurations but cannot directly map its findings to OWASP A02/A05 or CIS L1, nor does it have awareness of WordPress plugins/themes. The PoC tool bridges this gap by:

- Standardising a YAML rule set consisting of 75 rules mapped directly to the

checkpoints of OWASP A02/A05/A06 and CIS L1 (Apache, Nginx).

- Simultaneously combining data from HTTP Recon, the WPScan vulnerability API, and the on-server configuration Agent.
- Returning results to a unified dashboard where each finding is fully labelled: corresponding standard, severity level, and remediation guidance.

As a result, organisations can maintain a consolidated compliance posture instead of manually interpreting and merging results from multiple standalone tools.

Approaches used by Nikto or similar black-box scanners rely mainly on banners and HTTP responses to infer misconfigurations. In contrast, the PoC tool, through its remote agent, can:

- Directly read configuration files: nginx.conf, apache2.conf, and sites-enabled files
- Inspect file and directory ownership (owner, group, mode)
- Validate module enablement status, SSL/TLS policies, and the location and permissions of log file.

These checks correspond to detailed CIS L1 requirements, which cannot be reliably verified through a handful of external HTTP requests. This explains why the CIS L1 rule group achieves high Precision/Recall in the experiments: Each finding is backed by concrete configuration-level evidence rather than surface-level inference.

Advantage in Actionability of Reports: The PoC tool's reporting mechanism is built upon the structured rule definitions within the Rule Engine: Each rule contains a rationale and remediation guidance. When a rule fails, the dashboard immediately displays:

- The violated standard
- The technical evidence (which file, current permissions, directive value)
- The recommended configuration command or fix

Compared to Nikto's command-line style output or WPScan's raw CVE listings, this report format significantly reduces interpretation overhead and has a direct impact on MTTR (Mean Time To Remediate) in real operational environments.

Rule Coverage Across Standards: To quantify these observations, the thesis constructs a mapping matrix between:

- The checkpoints defined in each standard group (OWASP A02, A05, A06, CIS Apache L1, CIS NGINX L1)
- The rules or signatures supported by each tool (SecRuleMap - YAML rule set of the PoC tool, WPScan, testssl.sh, Nikto)

For the PoC tool, each YAML rule in SecRuleMap is explicitly tagged to its

corresponding checkpoint. For WPScan, testssl.sh, and Nikto, the mapping is derived from official documentation, signature sets, and published scanning scopes. The results are aggregated as the number of checkpoints each tool is capable of covering across each standard.

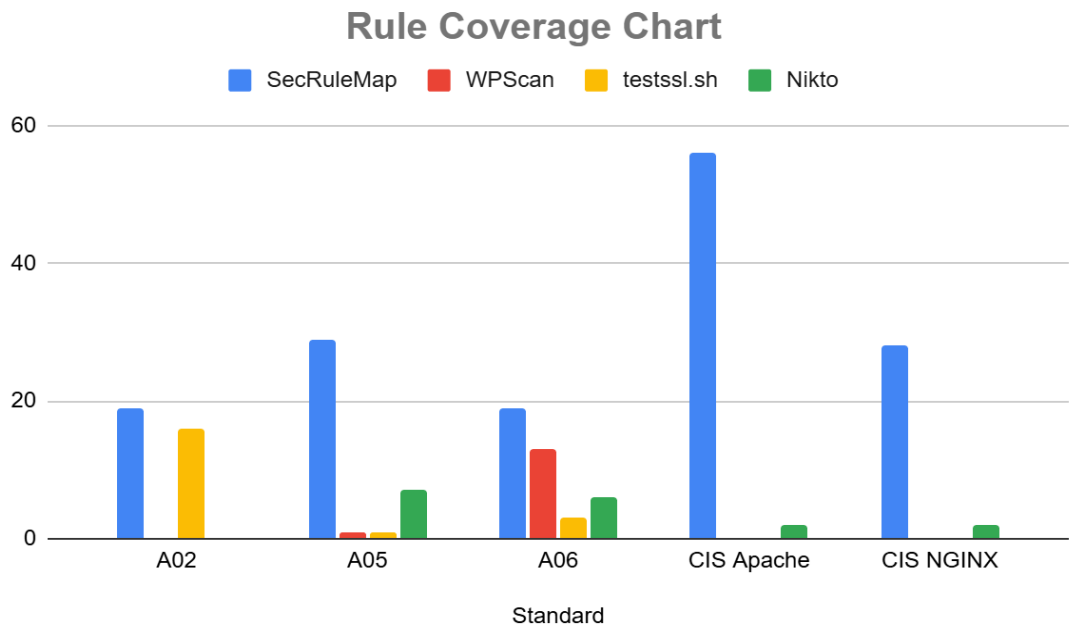


Figure 4.11: *Rule Coverage Chart Across Standards.*

Observing Figure 4.11 reveals the following:

- SecRuleMap achieves the highest coverage across all standard groups: The number of OWASP A02/A05/A06 and CIS Apache/Nginx checkpoints implemented as rules is significantly greater than in the other tools.
- WPScan shows meaningful coverage only in A06 and is nearly zero for A02, A05, and CIS. This reflects its nature as a WordPress-specific vulnerability scanner rather than a compliance assessment tool.
- testssl.sh contributes mainly to A02 and some SSL/TLS-related CIS requirements, but does not interact with application-level configuration or plugin ecosystems.
- Nikto includes several rules related to surface-level misconfigurations and a small number of server checkpoints, but its overall coverage remains substantially lower than SecRuleMap.

This chart reinforces the conclusion that the PoC tool not only unifies multiple standards into a single interface but also implements a larger number of checkpoints across the OWASP/CIS framework compared to reference tools designed for narrower objectives.

4.5.3. Quantitative Comparison of Differences

Beyond standard coverage, the thesis also compares the PoC tool with Nikto and WPScan using statistical evaluation metrics (Precision, Recall, F1-Score) based on the Golden Label Dataset described in Section 4.7.3.

For the CIS L1 rule group (infrastructure layer), the PoC's Rule Engine leverages Agent Evidence, giving it the ability to clearly distinguish between compliant and non-compliant states. When tested against the same set of configuration scenarios:

- The PoC achieves significantly higher Precision and Recall compared to Nikto, which can only infer configuration from HTTP responses. Many misconfigurations go undetected by Nikto when banners are hidden or insufficient, while the agent can directly read configuration files.
- The F1-Score for the CIS L1 group in the PoC approaches the ideal range (and is noticeably higher than application-layer rules), demonstrating the natural advantage of the white-box approach for infrastructure evaluation.

For the OWASP A06 rule group, WPScan, when operating as an independent scanner, achieves a high F1-Score on components covered by its CVE database. The PoC tool does not attempt to replace WPScan as a full scanner, instead, it uses the WPScan API as a structured input source for the Rule Engine:

- In terms of Recall for A06, the PoC and WPScan behave identically, since both depend on WPScan's vulnerability database coverage.
- However, when considering an aggregated multi-standard F1-Score (combining A02/A05/A06 and CIS L1), the PoC introduces a new metric: instead of reflecting detection ability for a single layer, the composite F1-Score represents the system's capability to maintain accuracy across multiple, heterogeneous standards.

Finally, it is important to emphasize that the comparative tools (WPScan, Nikto, testssl.sh, OWASP ZAP) were not designed with the Compliance as Code objective in mind. Therefore, the PoC's ability to achieve: Broader checkpoint coverage, high Precision/Recall in the infrastructure layer, and the ability to present results as OWASP/CIS compliance reports.

Demonstrates that the rule-engine and agent-based approach in this thesis is well-suited to bridging the gap between traditional vulnerability scanners and the compliance requirements of open source web environments.

4.6. Implications of the Results

The experimental outcomes and quantitative analyses validate the contribution of the proposed system to both academic research and practical implementation [69].

4.6.1. Theoretical Implications

Successful integration and analysis of data from heterogeneous sources (HTTP Recon, Vulnerability APIs, Agent Checks) into a unified compliance report fills a previously identified research gap, namely the absence of an integrated approach bridging the Application Layer and Infrastructure Layer.

Extensible Architectural Model: The validated modular architecture demonstrates strong extensibility. This design enables future researchers to easily add new rule sets for additional standards or to support other open-source CMS platforms (Joomla, Drupal), without modifying the core Rule Engine or Orchestrator.

4.6.2. Practical Implications

In terms of practical application, the proposed system is instrumental in promoting a DevSecOps culture. Providing an actionable, visual interface is critical for transforming raw security data into usable intelligence. By creating a shared platform where Development and Operations teams can jointly review and resolve compliance issues, the tool effectively breaks the historical Dev-Ops silo.

Additionally, the system contributes directly to improving operational KPIs and reducing risk. The consolidated reports and detailed filtering capabilities allow organizations using open-source web platforms to reduce key operational risk metrics. Specifically, the Mean Time to Detect (MTTD) is lowered thanks to automated multi-standard scanning replacing lengthy manual checks. Simultaneously, the Mean Time to Remediate (MTTR) is reduced through detailed remediation guidance extracted directly from the Rule Engine. [71]

4.6.3. Implications of AI-Assisted Reporting and Diff Analysis

The use of AI-assisted reporting and diff analysis has several key implications for improving security compliance workflows. First, AI significantly reduces the manual effort required to interpret large volumes of rule results. By automatically identifying improvements, regressions, and anomalies, the system enables analysts to focus on the most critical issues rather than manually comparing two full reports.

Second, AI provides deeper insight beyond status changes. MiniLM validates whether evidence truly changed, while Gemini interprets the security impact of those changes. This

allows the system to detect cases such as suspicious passes, incomplete fixes, or unexpected regressions, which would not be visible from PASS/FAIL codes alone.

Third, the AI-generated priority-based remediation guidance helps organizations act more efficiently. Gemini's structured recommendations translate raw scan data into practical next steps, improving decision-making and resource allocation.

Finally, AI-assisted diff analysis enhances continuous monitoring by detecting configuration drift early and providing a clearer picture of the system's evolving security posture. While human validation remains important, the AI framework strengthens accuracy, scalability, and operational responsiveness in modern compliance environments.

4.6.4 Limitations of the Results and Future Development

Although the results validate the feasibility of the PoC architecture, the FN (False Negative) analysis in Section 4.3 highlights inherent limitations requiring further research.

4.6.4.1. Key Limitations

1. FN and Zero-Day Risks: The tool is limited by its static analysis nature and reliance on external databases. It cannot detect Zero-day vulnerabilities or complex business logic flaws.
2. Scalability Constraints: The PoC version is not yet optimized for large-scale or continuous scanning due to computational constraints (CPU, memory) and the overhead of WPSCAN API queries for large volumes. Although performance improved after switching from WPScan API, scalability is limited by the single-node deployment architecture.
3. Limitations of Static Analysis: Runtime configuration issues cannot be detected.

4.6.4.2. Future Work

To transform the PoC tool into a comprehensive Application Security Posture Management (ASPM) platform, the following enhancements are proposed [73]:

1. CI/CD Integration: Develop plugins or scripts to integrate compliance testing into CI/CD systems. This aligns with the shift-left principle, ensuring compliance checks run automatically during early development stages.
2. Expanding SAST and Runtime Capabilities: Integrate open-source SAST tools to analyze plugin/theme source code, addressing zero-day limitations and strengthening

detection of insecure coding patterns.

3. Automated Remediation: Build a remediation module integrated with configuration management tools to automatically fix low-risk misconfigurations.
4. Enhanced Scalability: Adopt distributed architecture techniques to optimize scan queue processing and reduce WPSCAN latency, enabling the system to scale to hundreds of targets.

4.7. Experimental Framework and Environment Configuration

To objectively evaluate the effectiveness of the compliance testing tool, this thesis establishes a standardized experimental framework (testbed) and a Golden Label Dataset used across all measurement scenarios in Chapter 4 [64]. This section describes in detail: (i) the virtualized deployment environment, (ii) the allocation of YAML rule sets according to test layers and data sources, and (iii) the construction of the golden dataset used for computing Precision, Recall, and F1-Score.

4.7.1. Deployment Environment and Infrastructure Configuration

The test environment is deployed entirely on a Docker-based virtualization platform to ensure reproducibility and isolation between components. The configuration includes several main elements. First, the foundational layer consists of the Host Operating System, specifically Ubuntu Server LTS (64-bit), which serves as the host for the entire Docker stack. Augmenting this, the Container Platform utilizes Docker Engine and Docker Compose, which are essential for quickly provisioning application containers, reverse proxies, and the remote agent.

Regarding the target environment, the Target Application is WordPress version 6.3.1, installed in a standard setup including the core CMS and several common plugins. Furthermore, the web server behind WordPress is configured in two distinct modes: Apache HTTP Server 2.4.x (representing an Apache-based environment), and Nginx 1.24.x (acting as either a reverse proxy or the primary web server). Data persistence is managed by a Database System, utilizing MySQL/MariaDB running in a dedicated container connected internally to the WordPress container.

To enable infrastructure validation, a Configuration Checking Agent, a lightweight script written in Python 3.x, is deployed, either in a dedicated container or installed directly inside the web server container. This agent is responsible for collecting configuration information and returns binary Pass/Fail results to the central tool. Finally, the Compliance Scanning Tool (PoC) operates with its Backend and Rule Engine running inside a dedicated service

container. This service container is responsible for several key functions: collecting data from the Recon module (HTTP/HTML), querying the WPScan API for known vulnerabilities affecting the core/plugins/themes, communicating with the Agent to obtain Pass/Fail configuration results, and executing the YAML rule set to generate consolidated reports.

All containers communicate over an internal Docker network, with the host firewall configured to limit external access to only essential web and management ports. This comprehensive setup ensures easy reproducibility, as reloading the Docker Compose file is sufficient to reconstruct the entire environment.

4.7.2. Allocation and Classification of Test Rule Sets

The system contains 75 YAML rules, standardized under a unified schema (rule_id, standard, description, severity, type, target, assertion, rationale, remediation). These rules are allocated according to two dimensions:

- By Standard and Scope: OWASP Top 10 (Application Layer): A02 - Cryptographic Failures (10 rules): Focused on HTTPS configuration, TLS versions, HSTS, cookie flags, and security headers related to data-in-transit protection. A05 - Security Misconfiguration (20 rules): Detects exposed debug pages, directory listing, server banners, missing security headers, unprotected admin interfaces, etc. A06 - Vulnerable and Outdated Components (5 rules): Verifies WordPress core/plugin/theme versions against known vulnerabilities via WPScan API. The total is 35 OWASP rules covering the Application Layer. CIS Benchmark Level 1 for HTTP Servers (Infrastructure Layer): Approximately 40 CIS L1 rules are converted into YAML, covering: File configuration permissions (ownership, permission mode), Log directory location and permission settings, Disabled/required modules, Secure SSL/TLS configuration for Apache/Nginx, Prevention of information leakage through default pages and server error messages.
- By Technical Test Layer and Data Source: HTTP Recon-Based Checks: Uses data from Recon: status codes, response headers, and HTML content. Supports OWASP A05/A02 rules such as: Missing Strict-Transport-Security, missing X-Frame-Options, exposed /wp-admin/ directory listing, HTML banner leaks, etc. Vulnerability API Checks: Uses YES/NO/UNKNOWN results from the WPScan API mapped to (component, version). Primarily supports A06 rules and some A05 rules related to outdated plugins/themes. Agent-Based Checks (Remote Configuration Verification): Uses Pass/Fail results returned by the agent, mapped to

directive codes. Covers most CIS L1 rules and certain OWASP A05 infrastructure level misconfiguration rules.

Clear classification of rules by standard, testing layer, and technical data source (Recon/API/Agent) enables:

1. Precise coverage analysis.
2. Targeted creation of golden test cases.
3. Separate calculation of Precision/Recall/F1 for each layer in Section 4.8.

4.7.3. Golden Label Dataset for Evaluation

To measure the tool's accuracy, a Golden Label Dataset is constructed, containing configuration scenarios manually labeled as Compliant or Non-Compliant for each rule [63].

Steps to Construct the Golden Dataset:

1. Designing Configuration Scenarios for Each Rule Group. For each standard group, representative configurations are chosen to clearly express PASS and FAIL states: A02: Enabling/disabling TLS 1.0 - 1.1, enabling/disabling HSTS, enforcing HTTPS, and allowing plain HTTP. A05: Enabling/disabling directory listing, enabling/disabling debug pages, and configuring/missing key security headers. A06: Electing WordPress/plugin/theme versions with known CVEs and versions without CVEs. CIS L1: Modifying file permissions (600 vs 644), file ownership, enabling/disabling unnecessary modules, and altering SSL settings.
2. Applying Configurations on the Docker Testbed. Each configuration scenario is realized through: editing Apache/Nginx configuration files or WordPress/environment settings, adding/removing plugins/themes, Restarting the respective container. Each scenario is assigned a unique identifier for easy tracing.
3. Manual Labeling. For each scenario, the performer manually defines each rule: This rule must be marked Failed (Non-Compliant) or Passed (Compliant) if the system is working properly. The labeling is stored in a structured form: (scenario_id, rule_id, expected_status), expected_status $\in$ {PASSED, FAILED}.
4. Golden Dataset Aggregation. The dataset is constructed to ensure: Each rule contains at least one FAILED case and one PASSED case within the golden label set, The number of scenarios assigned to the infrastructure layer (CIS L1) and the application layer (OWASP A02/A05/A06) is balanced to enable accurate comparison between

the two layers. On average, each rule is exercised at least twice in the dataset (once in a compliant state and once in a non-compliant state), ensuring that the calculation of TP/FP/FN/TN metrics is meaningful.

The Golden Label Dataset is not part of the actual runtime system, it is an independent evaluation dataset used to:

- Compare the tool's output against the golden labels and construct the Confusion Matrix for each rule group.
- Derive Precision, Recall, and F1-Score for each layer (application/infrastructure) as well as the overall rule set.
- Serve as the foundation for the analyses in Section 4.8 regarding rule coverage, rule-set integrity, and the advantages of combining Recon and Vulnerability API (WPScan) and Agent within a unified modular architecture.

4.8. Standardized Data Structure and Technical Evidence Model

As described in Chapter 3, the Rule Engine does not operate on raw data (HTTP responses, agent logs, API query results). Instead, it works with a normalized data structure representing the security state of each. This section details:

1. How is the evidence blocked from each technical module organized?
2. How are they unified into a Target Snapshot for executing the YAML rule set?
3. An end-to-end example for a specific target.

4.8.1. An end-to-end example for a specific target.

The standardized data structure is designed with four primary objectives. First, it ensures source independence: the Rule Engine does not need to know whether evidence originates from Recon, the Agent, or the WPScan API; it only consumes a unified schema. Second, it guarantees traceability: each finding can be mapped one-to-one to a rule_id and its corresponding technical evidence (such as the specific header, directive, or plugin version). Third, the structure supports extensibility, allowing new fields to be added without modifying the core system structure. Finally, the data structure is report-friendly, ensuring the normalized data provides sufficient information for rendering dashboard content, including details such as the rule name, severity, summarized evidence, and remediation guidance.

To meet these requirements, the tool constructs a central object called the Target Snapshot for each scanned target. Represented in JSON format, it contains three main evidence groups:

1. Recon Evidence - data collected from the HTTP/HTML reconnaissance module.
2. Vulnerability Lookup Evidence - data retrieved from external vulnerability APIs (WPScan).S
3. Agent Evidence - configuration verification results returned by the server-side agent.

4.8.2. Reconnaissance Evidence Schema

The Recon Evidence group stores the information extracted from HTTP responses and HTML content, primarily serving the OWASP A02/A05 rule checks. A typical Recon structure is defined as follows:


```
  {
"target_url": "https://example.com",
"http_status": 200,
"server_banner": "nginx/1.24.0",
"wp_detected": true,
"wp_version": "6.3.1",
"security_headers": {
 "strict_transport_security": {
  "present": true,
  "raw_value": "max-age=31536000; includeSubDomains"
 },
 "x_frame_options": {
  "present": false
 },
 "content_security_policy": {
  "present": false
 }
},
"transport": {
 "scheme": "https",
 "tls_version": "TLSv1.3",
 "cipher_suite": "TLS_AES_256_GCM_SHA384"
},
"http_paths": {
 "/": {
```

```
    "directory_listing": false
   },
   "/server-status": {
    "reachable": false
   },
   "/wp-admin/": {
    "reachable": true,
    "protected_by_redirect_or_login": true
   }
  }
}
```


In which: Fields such as security_headers, transport, and http_paths serve as direct input sources for the rules:

- A02 - checking TLS versions, enforcing HTTPS, and secure cookies (if applicable).
- A05 - checking security headers, directory listing, and exposure of debug/admin pages.

The Rule Engine only needs to reference the corresponding JSON paths to evaluate the conditions defined in the assertion section of the YAML rule.

4.8.3.Vulnerability Lookup Evidence Schema (WPScan API)

The Vulnerability Lookup Evidence group stores the query results returned by the WPScan API for WordPress core, plugins, and themes. Instead of storing full CVE details, the tool normalizes the output into a simplified schema using binary states that are easier for the Rule Engine to consume:


```
   {
  "core": {
   "name": "wordpress",
   "version": "6.3.1",
   "vulnerable": false,
   "known_vulnerabilities": []
  },
  "plugins": [
   {
```

```
      "slug": "contact-form-7",
      "version": "5.8",
      "vulnerable": true,
      "known_vulnerabilities": [
        {
          "cve_id": "CVE-2024-1901",
          "severity": "high",
          "summary": "Unauthenticated option update in Contact Form 7"
        }
      ]
    },
    {
      "slug": "hello-dolly",
      "version": "1.7.2",
      "vulnerable": false,
      "known_vulnerabilities": []
    }
  ],
  "themes": [
    {
      "slug": "twentytwentythree",
      "version": "1.2",
      "vulnerable": false,
      "known_vulnerabilities": []
    }
  ]
}
```


Compared to the raw data returned by WPScan, this structure offers several advantages. It reduces and normalizes the necessary fields required for A06 rules, focusing specifically on vulnerable status and severity. This normalization, in turn, enables A05 and A06 rules to be written with simple and concise conditions. For example, an A06 Rule checking for a High/Critical CVE can be written directly against the normalized data, and an A05 Rule concerning an outdated plugin or theme can rely solely on the difference between the

installed and recommended versions.

4.8.4. Agent Evidence Schema (Configuration and Access Control)

The Agent Evidence group represents the results of the configuration checks executed by the agent, typically mapped directly to CIS L1 rules and a subset of OWASP A05 rules. Each evidence entry includes:

1. Directive_code: Matching the target of the rule.
2. Status: PASS or FAIL.
3. A short explanation and, when necessary, the observed value.

An example of a sample Agent payload is as follows:


```
 {
"host": "wp-apache-container",
"directives": [
 {
  "directive_code": "APACHE-CONF-OWNERSHIP",
  "status": "FAIL",
  "observed": {
   "path": "/etc/apache2/apache2.conf",
   "owner": "www-data",
   "group": "www-data",
   "mode": "0644"
  },
  "note": "Configuration file is not owned by root:root"
 },
 {
  "directive_code": "APACHE-DIR-LISTING",
  "status": "PASS",
  "observed": {
   "path": "/var/www/html/",
   "options": ["FollowSymLinks"]
  },
  "note": "Indexes option is not enabled"
 },
 {
```

```
    "directive_code": "NGINX-SSL-PROTOCOLS",
    "status": "PASS",
    "observed": {
      "ssl_protocols": "TLSv1.2 TLSv1.3"
    },
    "note": "Insecure protocols (SSLv3, TLSv1.0, TLSv1.1) are disabled"
   }
  ]
}
```


4.8.5. Unified Target Snapshot Object

To simplify the input for the Rule Engine, the three evidence groups described above are consolidated into a single JSON object called the Target Snapshot. Logically, a snapshot is structured as follows:


```
   {
  "target_id": "site-001",
  "scan_id": "scan-2024-04-15T10:30:00Z",
  "recon": { /* Recon Evidence như 4.8.2 */ },
  "vuln_lookup": { /* Vulnerability Lookup Evidence as in 4.8.3 */ },
  "agent": { /* Agent Evidence như 4.8.4 */ }
}
```


When the Rule Engine is executed, it only needs to:

1. Load the Target Snapshot associated with a specific scan_id.
2. Sequentially apply all 75 YAML rules, where each rule:

- Specifies its data source (recon, vuln_lookup, agent),
- Contains an assertion expression that references specific fields within the snapshot,
- Records the Passed/Failed outcome into the results database.

This design provides several benefits:

- Independence between data collection and evaluation: The Recon, WPScan Lookup, and Agent modules can be upgraded or replaced without modifying the rule logic.
- Traceability: Every finding can be traced back to the snapshot and the underlying raw evidence.
- Reproducibility: Storing the snapshot allows re-running the Rule Engine on the same

dataset if rules are updated in the future.

4.8.6. Illustrative Example: From Normalized Data to Compliance Finding

To demonstrate the processing flow, consider the rule OWASP-A06-CVE-PLUGIN-HIGH, which aims to detect any WordPress plugin running a version associated with a High or Critical CVE as recorded in the WPScan database. The evaluation process on a given Target Snapshot proceeds through three logical steps:

The process begins with Step 1: Reading normalized data. The Rule Engine accesses specific fields within the Target Snapshot, primarily focusing on the snapshot.vuln_lookup.plugins[*].vulnerable status and the associated snapshot.vuln_lookup.plugins[*].known_vulnerabilities[*].severity.

Subsequently, Step 2: Applying the rule condition, the engine evaluates the logical assertion defined in the rule's YAML. This condition is equivalent to checking if there exists at least one plugin where vulnerable == true and at least one associated CVE has a severity of $\in$ {high, critical}.

Finally, Step 3 focuses on recording evidence and results. If the condition is satisfied (i.e., a vulnerable plugin with a high/critical CVE is found), the Rule Engine marks the rule as Failed (Non-Compliant). It then records the summarized technical evidence, including the plugin name, installed version, CVE identifier, and Severity level. This information is subsequently displayed in the dashboard in a standardized format, detailing the Rule ID, standard (OWASP A06), severity (High), evidence (plugin and CVE), and remediation instructions.

This example demonstrates that using a normalized data structure makes the entire workflow from gathering technical evidence to producing compliance findings transparent, verifiable, and easily extensible. It also ensures that new standards or data sources can be added in future research without modifying the core logic.

CHAPTER 5: DISCUSSION

5.1. Restate the Research Problem or Objectives

This section reiterates the research problem and the main objectives of the study, serving as a basis for evaluating the level of completion in subsequent parts of Chapter 5.

Research Problem:

In open-source web application deployments, particularly WordPress running on Apache/Nginx (Ubuntu LTS), operations teams must simultaneously reference multiple security standards: OWASP Top 10 at the application layer and CIS Benchmarks at the infrastructure layer [78]. Popular tools such as WPScan, Nikto, and testssl.sh each cover only isolated aspects and none provide a unified compliance testing mechanism that maps results directly to the control groups and checkpoints defined in OWASP A02/A05/A06 and CIS Level 1 [86]. Manual correlation is error-prone, inconsistent, and makes assessing overall compliance difficult [94].

Research Objectives:

The project addresses the above challenge within the scope of WordPress running on Apache/Nginx by pursuing the following objectives:

1. Design an integrated two-layer compliance testing architecture. Combine data from the application layer (WordPress core, plugins, themes, HTTP/HTTPS) and the infrastructure layer (Apache/Nginx configurations on Ubuntu) within a unified architecture [75]. Ensure each finding is linked to its corresponding reference standard (OWASP A02/A05/A06, CIS L1) and its technical evidence source (HTTP Recon, vulnerability API query, configuration agent) [76].
2. Implement Compliance-as-Code. Formalize all compliance requirements into a set of 75 YAML rules covering OWASP A02/A05/A06 and CIS Level 1 (Apache, Nginx) [81]. Develop a Rule Engine that evaluates normalized input data, decoupling rule definitions from source code to simplify maintenance and extensibility [82].
3. Develop a multi-source technical evidence collection mechanism. A Recon module that gathers HTTP/HTML information and WordPress characteristics [77]. A vulnerability lookup module that queries external databases (WPScan API) to support rules related to vulnerable components [87]. A remote Agent on the server that collects configuration states and returns Pass/Fail results for directives associated with CIS L1 [79].
4. Develop a compliance reporting interface and an experimental evaluation framework. Provide a dashboard that aggregates results by standard, severity, and rule [95]. Propose and apply an evaluation framework (Golden Label Dataset, Confusion Matrix, Precision/Recall/F1) to measure the accuracy of the tool across

the defined rule set and experimental environment [92][93].

5.2. Summarize Key Findings

This section summarizes the main findings derived from the architectural design (Chapter 3) and the experimental evaluation conducted in Chapter 4 [80]. The findings are presented in four groups: architecture, compliance rule set, agent mechanism, and visibility/traceability [83].

5.2.1. The integrated two-layer architecture (application-infrastructure) is feasible in practice

Experimental results show that the proposed architecture, combining the application layer (WordPress, HTTP/HTTPS, plugins/themes) and the infrastructure layer (Apache/Nginx on Ubuntu LTS), can be implemented using relatively simple technical components:

1. The Recon module performs HTTP/HTTPS requests, extracts headers, and identifies basic WordPress characteristics (version, admin paths, default pages) [88].
2. The external vulnerability lookup module (WPScan API) provides vulnerable/not vulnerable status for WordPress core, plugins, and themes, supporting OWASP A06 rules [86].
3. The remote Agent reads configuration files and system settings, returning Pass/Fail results for directives mapped to CIS Level 1 [80].
4. The Rule Engine processes normalized data from all three sources, executes the 75 YAML rules, and stores the results in the reporting database [81].

This architecture enables a single scan to produce enough technical evidence to assess multiple standards at once (OWASP A02/A05/A06, CIS Apache L1, CIS Nginx L1), instead of running separate tools for each layer [85].

5.2.2. The Compliance-as-Code rule set for OWASP A02/A05/A06 and CIS L1

The 75-rule YAML set developed in this study effectively operationalizes the key requirements of OWASP Top 10 (A02 - Cryptographic Failures, A05 - Security Misconfiguration, A06 - Vulnerable and Outdated Components) at the WordPress application layer [74] and CIS Benchmark Level 1 for Apache HTTP Server and Nginx at the web server infrastructure layer [79]. Furthermore, each rule adheres to a unified structure (rule_id, standard, description, severity, type, target, assertion, rationale, remediation) and is clearly mapped to its data sources (Recon, WPScan API, Agent) and testing layers (Application layer or infrastructure layer) [81].

Experiments using the Golden Label Dataset and Confusion Matrix show clear

performance profiles [92][93]. Specifically, CIS L1 rules achieve high accuracy (Precision/Recall/F1 at very strong levels), largely due to the clear Pass/Fail nature of Agent Evidence [93]. In contrast, OWASP A02/A05/A06 rules exhibit slightly lower Precision/Recall but remain acceptable, with deviations largely caused by HTTP reconnaissance limitations and dependency on WPScan's vulnerability coverage [86]. These findings confirm that the Compliance as Code approach is feasible within the selected scope and that the current rule set can serve as a baseline for future expansion [82].

5.2.3. The remote agent mechanism meets deep configuration inspection and security requirements

Testing with the Python-based agent deployed on the web server containers shows that:

1. The agent successfully collects configuration information required for CIS L1, such as ownership and permissions of configuration files, enabled modules, SSL/TLS settings, log directory configuration and more [79].
2. The data returned to the Rule Engine is minimized to Pass/Fail and essential observed values, without transmitting full configuration files reducing risks of sensitive information leakage [80].
3. The agent operates under the least privilege principle, only requiring the necessary read access for configuration inspections, making it suitable for production-like environments [89]

Thus, the tool enables white-box checks at the infrastructure layer while maintaining a reasonable security boundary between the testing system and the target server [90].

5.2.4. Visibility and traceability through the dashboard and normalized data model

By normalizing all input data into a Target Snapshot and storing rule results in structured form, the system provides:

- Overview reports: Compliance scores per standard, severity distribution, and Passed/Failed rule counts for OWASP/CIS [83].
- Detailed findings: Each finding is linked to rule_id, standard, severity, status, technical evidence (headers/config directives/CVEs), and remediation guidance [84].
- Traceability: Any dashboard finding can be traced back to the YAML rule and the corresponding fields in the Target Snapshot that led to the failure [85].

These results demonstrate that the proposed architecture goes beyond running rules and printing logs by delivering a practical compliance reporting layer suitable for real-world auditing and security assessments [92].

5.2.5. AI-Assisted Analysis and Diff Intelligence

Beyond the rule-based compliance evaluation, this study incorporates an AI-enhanced analysis module that leverages a two-stage pipeline: MiniLM embeddings for semantic similarity and Google Gemini for natural-language synthesis. This AI subsystem contributes three key capabilities:

Automated interpretation of scan differences

The embedding model (MiniLM) identifies semantically meaningful changes between Before and After scan snapshots, enabling the system to detect improvements, regressions, or partial remediation even when the underlying evidence formats differ[92.1].

AI-generated compliance narratives

Gemini can transform structured diff data into human-readable explanations and remediation summaries, enhancing the interpretability of rule-level and scan-level results for non-expert users and supporting clearer communication in compliance reports[92.2].

Support for continuous compliance monitoring

The AI diff pipeline allows security teams to track remediation progress across multiple scans, helping to transform the tool from a static compliance checker into a dynamic risk-reduction assistant.

These findings demonstrate that the integration of AI strengthens visibility, accelerates remediation, and enriches the reporting layer, while leaving the core rule engine unchanged.

5.3. Interpretation of Empirical Validation Results

This section interprets the experimental results presented in Chapter 4, focusing on two main categories: (i) the infrastructure layer (CIS Level 1) based on agent evidence, and (ii) the application layer (OWASP A02/A05/A06) based on Recon data and external vulnerability databases [93]. From these results, several implications for testing practices and risk management are derived within the scope of the implemented system [74].

5.3.1. Effectiveness of the CIS Level 1 Rule Set at the Infrastructure Layer

The evaluations conducted using the Golden Label Dataset show that the CIS Level 1 rule group (Apache, Nginx) achieves high statistical accuracy [78]. The Confusion Matrix for this group records virtually no significant False Positives or False Negatives, leading to strong Precision, Recall, and F1-Score values [79].

The main reasons include:

1. CIS L1 input data is directly supplied by the configuration Agent as clear directives

(Pass/Fail) [80].

2. CIS L1 requirements are typically binary and easy to verify [89].
3. The one-to-one mapping between directive_code in the agent payload and rule_id in the YAML rules minimizes interpretation errors [90].

From an operational standpoint, the CIS L1 rule group can be viewed as a stable infrastructure configuration baseline: If a CIS rule is marked Failed, it is highly likely to reflect an actual misconfiguration, and organizations can prioritize remediation based on the corresponding suggestions [91].

5.3.2. Effectiveness and Limitations of the OWASP A02/A05/A06 Rule Set at the Application Layer

For the OWASP rule group, statistical results show Precision and Recall at medium-good levels, but noticeably lower than CIS L1. This behavior mainly stems from the nature of the application layer and the limitations identified in Chapter 3 and Chapters 4.

Key observations include: A02 - Cryptographic Failures and parts of A05 - Security Misconfiguration rely on Recon (HTTP/HTTPS). These rules work well for clear-cut cases, but tend to produce False Positives when HTML pages appear similar to directory listings or debug pages but are legitimate application interfaces [86], and False Negatives when issues occur on paths inaccessible to Recon [87]. A06 - Vulnerable and Outdated Components depends on accurate identification of WordPress core, plugin, and theme versions via Recon [88] and coverage of the external vulnerability database (WPScan API) at scan time [86].

In scenarios where plugins/themes fall outside WPScan coverage or have modified slugs/versions, the tool may mark Passed while the Golden Label expects Failed, resulting in False Negatives [95].

These results indicate that:

- The current OWASP rule set is suitable as an application warning layer within the chosen scope, but cannot replace manual pentesting or deeper DAST analysis.
- Most errors (FP/FN) are consequences of HTTP coverage limitations and dependency on external vulnerability coverage, not architectural flaws of the Rule Engine [81].

5.3.3. Implications for Testing Practices and Risk Management

The differing performance profiles of the two rule groups yield several practical implications for how the tool should be used in compliance testing and risk management:

- For the infrastructure layer (CIS L1): Scan results can be used directly as a technical

checklist for server hardening [78]. Organizations may treat CIS failed findings as (must-fix) items before production deployment or before undergoing external audits [79].

- For the application layer (OWASP A02/A05/A06): Findings should be interpreted as an early-warning layer, helping identify common configuration and outdated component issues in WordPress environments [75]. Failed OWASP findings represent a strong starting point for deeper investigation (through configuration review, manual testing, or specialized DAST tools), rather than final conclusions [76].
- For overall risk management: Mapping findings to familiar standards (OWASP, CIS) allows risk management teams to align technical risks with existing internal control requirements [82]. Compliance dashboards (compliance scores per standard, severity distribution, trends across multiple scans) can support: remediation prioritization [83], tracking improvement progress [84], and preparation for audits and external assessments [85].

Overall, the results of Chapter 4 and the interpretation in Section 5.3 demonstrate that the PoC tool fulfills its goal of establishing a multi-standard compliance testing platform for WordPress on Apache/Nginx, offering high reliability at the infrastructure layer and a helpful warning layer at the application layer [92]. The remaining limitations, primarily related to coverage and dependency on external data, are examined further in the later sections on limitations and future work [93].

5.3.4. Interpretation of AI-Generated Insights

The AI diff-analysis subsystem provides an additional interpretive layer on top of quantitative metrics (Precision, Recall, F1). Its contribution can be interpreted in three ways:

Bridging the gap between raw technical outputs and decision-making

AI-generated summaries contextualize rule failures, helping operations teams understand the broader compliance impact rather than focusing solely on rule-by-rule outputs.

Highlighting remediation effectiveness across scans

By comparing semantic changes in evidence, AI can identify whether a failed configuration or outdated component has been fully fixed, partially fixed, or remains unchanged-offering insight beyond what the confusion matrix reflects.

Reducing cognitive load and analysis time

Given the large number of rules (75) and nuanced differences between snapshots, AI greatly reduces manual analysis effort, thus supporting efficient security operations.

This interpretive function complements the statistical validation in Sections 4.3 and 5.3.1-5.3.3, showing how AI enhances-not replaces-traditional rule-based compliance assessment.

5.4. Comparative Analysis and Uniqueness of the Solution

This section positions the PoC tool relative to the open-source tools discussed in Chapter 2 (WPScan, Nikto, OWASP ZAP), clarifying: (i) the gap addressed by the project, and (ii) the main differentiating factors of the solution compared to existing tools [94]. The analysis is grounded in the tool's architecture and empirical results within the defined scope of WordPress on Apache/Nginx, with OWASP A02/A05/A06 and CIS L1 as reference standards [95].

5.4.1. Addressing the Multi-Standard Fragmentation Gap

Popular open-source tools such as WPScan, Nikto, and OWASP ZAP are each designed with distinct objectives, resulting in fragmentation across security layers and risk categories. Specifically, WPScan focuses on WordPress components (core/plugins/themes), corresponding to OWASP A06 - Vulnerable and Outdated Components, but does not cover detailed Apache/Nginx configuration requirements under CIS L1 [86]. Similarly, Nikto targets black-box server surface scanning, detecting some misconfigurations and information disclosure through HTTP, but cannot interpret the WordPress ecosystem or map results directly to OWASP/CIS groups [94]. Furthermore, OWASP ZAP is a general-purpose DAST tool suitable for active penetration testing of web applications, but was not designed as a multi-standard compliance engine mapped to OWASP/CIS [74].

Within this context, the PoC tool acts as a consolidation and reporting layer for compliance tasks. It does not replace WPScan, Nikto, or ZAP; instead, it integrates their outputs or data sources (notably WPScan API for A06) into a unified data collection pipeline [87]. Data from HTTP Recon, vulnerability API queries, and configuration agent checks are normalized into a common schema and evaluated by the Rule Engine using the 75 YAML rules mapped to OWASP A02/A05/A06 and CIS L1 [75]. Subsequently, the final results are displayed on a single dashboard showing compliance status by standard, severity distribution, and labeled findings per rule/standard [76].

Thus, the uniqueness of the solution lies not in discovering new vulnerabilities, but in three core areas: First, unifying multiple standards (OWASP A02/A05/A06, CIS L1) within one architecture [77]; Second, standardizing assessment results into findings tied to rule_id and specific compliance standards [78]; and Finally, reducing manual correlation overhead when organizations need to satisfy multiple OWASP/CIS requirements simultaneously [79].

This consolidated position enables consistent tracking and reporting of compliance posture [80].

5.4.2. Advantages of Deep (White-Box) Configuration Checking via Agent

Compared to black-box scanning approaches like Nikto (which infer configuration from HTTP responses, banners, and headers), the agent-based method in this project enables deterministic white-box evaluation of internal Apache/Nginx directives.

Key advantages include:

1. Coverage of detailed CIS requirements. The agent can directly read configuration data and system states required for CIS L1 rules, such as: Ownership and permissions of configuration files [89], the enable/disable status of unnecessary modules as recommended by CIS [90], detailed SSL/TLS settings, cipher suites, log directory configuration, etc [91], and these aspects cannot be accurately evaluated from external HTTP requests alone [82].
2. Reduced configuration exposure risk. The agent is designed to return only normalized results (Pass/Fail and directive code), without transmitting full configuration files or sensitive details [83]. This: Aligns with strict production environments that prohibit exposing raw server configurations [84], removes a major deployment barrier commonly associated with tools requiring direct configuration file access [85].
3. A foundation for quantitative evaluation at the infrastructure layer. Because agent data is binary and deterministic, the Rule Engine can: Minimize False Positives/False Negatives for CIS L1 rules [92], provide a strong basis for calculating Precision/Recall/F1 at the infrastructure layer, as shown in Chapter 4 [93].

These advantages exist within the defined scope of the project:

- The agent handles static configurations at scan time, but does not capture dynamic runtime changes [94].
- The application layer remains dependent on fingerprinting accuracy and external vulnerability database coverage, thus, the approach does not replace active penetration testing or advanced DAST tools [95].

Overall, Section 5.4 demonstrates that the PoC tool distinguishes itself from reference tools in two core aspects: (i) its role as a multi-standard compliance consolidation and reporting layer, and (ii) its ability to safely perform deep configuration evaluation using an agent. These characteristics directly result from the architecture and rule set defined within the scope of the study [88].

5.5. Limitations, Constraints, and Scope Boundaries

This section consolidates the main limitations of the PoC system and the evaluation framework used in the thesis. These limitations clarify the applicability boundaries of the results and indicate areas recommended for future research, rather than representing shortcomings of a fully finalized product.

5.5.1. Limitations Concerning Scope and Supported Standards

First, the applicability scope of the current tool is intentionally restricted to a relatively narrow context. The assessment focuses primarily on the Application platform: WordPress [76]; Web servers: Apache HTTP Server and Nginx running on Ubuntu LTS (containerized environment) [77]; and specific Standards: OWASP A02/A05/A06 at the application layer, and CIS Benchmark Level 1 for Apache/Nginx at the infrastructure layer [78].

This implies several limitations for generalization. First, the tool does not yet provide direct support for other CMS or frameworks [79]. Second, other standards, such as PCI DSS, NIST SP 800-53, or ISO 27001, are mentioned only as future directions, without corresponding rule sets in the PoC [80]. Consequently, the results and conclusions should not be generalized to different technology environments without adjusting the rules and re-validating them [89]. Ultimately, while this scope aligns with the initial research objectives, it represents a limitation that must be considered when deploying the tool beyond WordPress on Apache/Nginx [90].

5.5.2. Limitations in Data Collection and Coverage

The current system is based primarily on static, snapshot-based scanning with three data sources: HTTP/HTTPS Recon, external vulnerability lookup (WPScan API), and the Configuration Agent on the web server.

This reliance leads to several constraints. First, the tool does not observe runtime behavior (real traffic, long-term access logs, post-authentication behavior), and therefore cannot detect issues that only appear during dynamic execution or complex business logic [81]. Second, Recon currently has limited HTTP coverage (a small number of paths and access scenarios) and does not simulate authenticated sessions, meaning application-layer issues that surface only under specific conditions may be missed [83]. Finally, A06 rules are entirely dependent on WPScan API coverage at scan time; plugins/themes not yet included in the database or with difficult-to-fingerprint versions may lead to detection gaps (potential False Negatives) [84].

Additionally, the Golden Label Dataset used for accuracy evaluation was created in a

controlled lab environment with intentionally misconfigured scenarios. While this supports systematic analysis, it does not fully reflect the complexity and diversity of real-world production configurations [92]. Consequently, Precision, Recall, and F1 metrics should be interpreted as testbed results, not guarantees for all production deployments [93].

5.5.3. Limitations in Deployment Architecture and Scalability

The current deployment architecture remains at the PoC level on a single node, with one backend orchestrating scans over a limited set of targets. This introduces several limitations:

- Scalability has been assessed only qualitatively and on a small number of websites, the thesis does not evaluate system behavior when the number of targets grows significantly (tens/hundreds of sites). Many scan tasks run concurrently. The scan frequency increases to production-like levels.
- Optimization mechanisms such as distributed workers, task queues, or advanced caching/mirroring for vulnerability data remain conceptual proposals and are not implemented in the PoC.
- Integration with CI/CD pipelines or existing monitoring systems (SIEM, ticketing) has not been implemented, even though the Rule Engine and dashboard were designed with extensibility in mind.

While these limitations do not diminish the PoC's value within the lab environment, they must be addressed before scaling the solution for broader use.

5.5.4. Limitations of Agent Deployment and Operational Assumptions

Finally, while the remote agent provides significant advantages for deep configuration inspection, it also introduces certain assumptions and constraints:

- The system assumes that the agent can be installed and operated on the target web servers. In environments with strict policies prohibiting new software installation, agent deployment may be challenging [78].
- The agent is currently designed for Linux environments running Apache/Nginx, extending support to other operating systems or web servers requires additional development, which is outside the scope of this thesis [79].
- Although the agent only returns minimized Pass/Fail results, granting read access to configuration files remains a security consideration that each organization must assess [80].

In summary, these limitations indicate that the system serves as a proof-of-concept for a multi-standard compliance testing architecture in WordPress Apache/Nginx environments. Extending the technology scope, increasing deployment scale, or relaxing operational

assumptions requires additional development, which is discussed in the future work of Chapter 5.

5.6. Significance and Broader Implications of the Study

This section presents the significance of the research results from two perspectives: (i) theoretical implications for multi-standard compliance in open-source environments, and (ii) practical implications for operations, DevSecOps, and risk management within WordPress on Apache/Nginx.

5.6.1. Theoretical Significance: A Reference Architecture for Multi-Standard Compliance

From a theoretical perspective, the thesis contributes a reference architecture for multi-standard compliance testing in open-source web environments.

The research successfully establishes a multilayer compliance analysis model. The proposed architecture demonstrates that unifying data from multiple technical sources (HTTP Recon, external vulnerability databases, configuration agent) into a single Rule Engine is feasible. This illustrates the critical shift from single-purpose vulnerability scanning to multi-layer compliance analysis (application and infrastructure) within one platform, rather than treating each layer separately as traditional tools do.

The work focuses on supporting and illustrating the Compliance as Code principle. Modeling 75 requirements from OWASP A02/A05/A06 and CIS L1 as structured rules (YAML), separate from core source code, provides empirical evidence for this principle. This approach shows that: Compliance logic can be represented as data [81], can be updated or standardized without architectural changes [82][83], and can be evaluated using quantitative metrics (Precision/Recall/F1) on a Golden Label Dataset [92][93].

The architecture validates the potential for extension to other standards and platforms. Although this thesis focuses on WordPress on Apache/Nginx and OWASP A02/A05/A06 and CIS L1, the modular architecture (independent data collectors, central Rule Engine, reporting layer) is inherently extensible. Specifically, it can be extended with additional rule sets [92], and can potentially support other CMS platforms in future research, provided that appropriate rule sets and evaluation frameworks are developed [93].

The theoretical contribution lies not in covering every standard, but in validating a reusable architectural model for implementing multi-standard Compliance as Code within a defined environment.

5.6.2. Practical Significance: Supporting DevSecOps and Risk Management

From a practical perspective, the PoC system delivers several benefits within WordPress Apache/Nginx operations, even within the controlled testbed.

The tool provides structured compliance testing to replace parts of manual checking. By automating the comparison against OWASP A02/A05/A06 and CIS L1, the tool reduces reliance on manual checklist-based configuration and component reviews [95], standardizes compliance evaluation across multiple scans [92][93], and produces structured reports usable by technical teams (Dev/Ops) and compliance/risk stakeholders [74].

Furthermore, the system is essential for enabling DevSecOps and shift-left practices within scope. The dashboard provides compliance scores by standard [75], detailed findings with remediation [76], and filtering by standard/severity [77]. This enables both development and operations teams to access security information, rather than restricting it to security/audit personnel. The tool can therefore function as a training and awareness platform for Dev/Ops, and as an intermediate step before penetration testing or external assessments.

The PoC shows potential improvement in operational KPIs (MTTD/MTTR). In principle, an automated compliance scanner with clear remediation guidance can reduce Mean Time to Detect (MTTD) for configuration drift and outdated components via periodic scans and can reduce Mean Time to Remediate (MTTR) thanks to rule-level remediation instructions attached to each finding. However, these KPI improvements were not quantitatively measured in production environments; thus, such implications are practical interpretations of the architecture rather than empirically proven outcomes.

Overall, from a practical standpoint, the PoC represents a transition from single-purpose vulnerability scanning to structured multi-standard compliance testing that assists technical and risk teams in monitoring, prioritizing, and remediating configuration and component issues within WordPress-Apache/Nginx environments.

5.6.3. Significance of AI in Compliance Interpretation and DevSecOps

The integration of MiniLM and Gemini into the compliance pipeline illustrates how AI can augment traditional security tooling within a DevSecOps workflow:

Enhanced interpretability of compliance results

AI transforms structured rule outputs into explanations, summaries and prioritised recommendations, making compliance findings more accessible to non-security specialists.

Acceleration of remediation workflows

By automatically identifying changes between scans and classifying the outcome (full fix, partial fix, regression), the AI layer supports faster remediation decision-making,

incident response and continuous compliance.

Foundations for future ASPM platforms

The AI analysis layer demonstrates the potential path from purely rule-based reporting towards intelligent security posture management (ASPM), where capabilities such as semantic analysis, correlation and, in future, predictive insights become increasingly important.

Overall, the significance of AI in this PoC lies not in detecting additional vulnerabilities, but in improving comprehension, remediation prioritisation and iterative compliance management across DevSecOps cycles.

5.7. Summary and Concluding Discussion

This section summarizes the discussion in Chapter 5 and outlines potential future enhancements, without expanding beyond the current PoC scope. All suggestions are positioned as future research or implementation directions.

5.7.1. Completion of Research Objectives

Compared to the objectives stated at the beginning of this chapter, the results demonstrate that:

- The two-layer integrated architecture (WordPress application layer and Apache/Nginx infrastructure layer) has been successfully realized; data from HTTP Recon, external vulnerability lookup (WPScan API), and the configuration Agent are unified into a single Rule Engine.
- The Compliance-as-Code principle has been validated for the 75 YAML rules for OWASP A02/A05/A06 and CIS L1, CIS L1 rules in particular exhibited high and stable Precision/Recall on the Golden Label Dataset.
- The remote agent enables deep infrastructure-level checks while maintaining the least-privilege principle and sending only minimized Pass/Fail results.
- The compliance dashboard provides a unified view of OWASP/CIS status and supports quantitative comparison with black-box tools within the constructed testbed.

Thus, the thesis demonstrates the feasibility of a multi-standard compliance testing PoC for WordPress on Apache/Nginx, acting as a consolidation and reporting layer rather than a comprehensive vulnerability scanner.

5.7.2. Directions for Future Work

The following items lie outside the implemented PoC but are suggested as future research and development paths:

1. Deeper application-layer analysis: Integrate open-source SAST tools for analyzing WordPress plugins/themes, add runtime signal collection (logs, telemetry) to reduce reliance on static scans and improve Recall for complex logic flaws or near-zero-day scenarios.
2. Improving scalability and distributed deployment: Move from single-node deployment to distributed worker pools or task queues for parallel scanning at scale, and investigate local caching/mirroring of vulnerability data to reduce query latency and support higher scan volumes.
3. Stronger DevSecOps integration: Develop plugins/scripts for CI/CD integration (Jenkins, GitLab CI) to enable automated compliance checks in shift-left pipelines, and standardize output formats for easier integration with ticketing, SIEM, or risk management platforms.
4. Extending standards and supported platforms: Build additional rule sets for standards such as PCI DSS or NIST SP 800-53, or support additional CMS/web stacks (Joomla, Drupal), and each extension must be accompanied by its own Golden Label Dataset and evaluation framework, similar to the approach used for WordPress and OWASP/CIS.

In conclusion, Chapter 5 establishes the PoC as a strong feasibility demonstration of multi-standard compliance testing within its defined scope. The listed future directions outline potential paths for evolving the PoC into a more complete platform while maintaining clear boundaries between implemented features and proposed research.

CHAPTER 6: CONCLUSION AND FUTURE WORK

6.1. Conclusion

In the context of open-source web systems being increasingly adopted to deliver online services, WordPress stands out as a typical example with its rich ecosystem of plugins, themes and a large developer community [104]. Combining the WordPress core, plugins, and themes with a web server (Apache/Nginx) and a Linux operating system shortens

deployment time, but at the same time expands the attack surface and makes security risk management more complex [100].

In parallel, organizations are often required to comply simultaneously with multiple security standards and recommendations such as OWASP Top 10 (especially A02 - Cryptographic Failures, A05 - Security Misconfiguration, A06 - Vulnerable and Outdated Components) and CIS Benchmark Level 1 for Apache/Nginx [99]. However, most existing tools only focus on individual slices: scanning application vulnerabilities, listing versions, checking a subset of HTTP headers, or validating a limited portion of web server configuration [113]. This leads to fragmented compliance testing that is difficult to consolidate and challenging to map directly to specific clauses in the standards. This is precisely the gap that this thesis aims to address.

Building on this premise, the thesis proposes and implements an integrated, multi-standard security compliance testing tool for WordPress environments running on Apache/Nginx, grounded in the Compliance as Code philosophy [105]. The core idea is that, instead of stopping at raw technical vulnerability lists, the system directly models the requirements from OWASP and CIS as structured rules that are both machine-readable and machine-executable, and then maps the evaluation results back into the language of the corresponding standards [106].

The system is designed following a web client–server model with a modular architecture, comprising:

Reconnaissance module: Sends HTTP/HTTPS requests to the target website, collects HTTP headers, HTML content, and WordPress fingerprinting indicators (version, admin path, default pages, etc.), thereby detecting unsafe configurations and potentially risky behaviors [101], and external vulnerability database lookup module: Leverages vulnerability and version data for WordPress core, plugins, and themes to identify outdated components or those with publicly disclosed vulnerabilities, supporting rules related to OWASP A06 [116], and remote configuration agent: Deployed on the web server, performs local checks against Apache/Nginx configuration and selected system properties, returning normalized Pass/Fail results for each directive, supporting the CIS rule group and a subset of OWASP rules at the infrastructure layer [97], and rule Engine: Consumes normalized data (Target Snapshot), executes the YAML rule set, and stores the results for visualization on the compliance reporting dashboard [105].

Building on this architecture, the thesis defines a set of 75 rules that are representative of the subset of requirements from OWASP A02/A05/A06 and the CIS Benchmark Level 1 for Apache/Nginx that can be automated. Each rule is modeled with the following fields: rule_id identifier, reference standard, description, severity level, test type (type), target, evaluation condition (assertion), security rationale (rationale), and remediation guidance (remediation). Decoupling rule definitions from the application source code enables:

- Easily update, extend, and reuse the rule set as standards evolve or as new test scenarios emerge, since all compliance logic is decoupled from the system's core application code [107].
- Accurately embody the Compliance as Code principle, in which compliance logic is managed as structured data rather than being intertwined with business logic, thereby ensuring consistency, maintainability, and reliable version control across the entire system [105].

In addition to designing and implementing the system, the thesis also proposes and applies an experimental evaluation framework based on:

- A standardized experimental testbed environment with WordPress running on Apache/Nginx across multiple configuration scenarios [100];
- A Golden Label Dataset in which the compliance state of each scenario is pre-determined through manual verification;
- Statistical metrics commonly used in classification such as the Confusion Matrix, Precision, Recall, and F1-Score to quantify the accuracy of each rule group.

The experimental results indicate that:

- The rule group corresponding to the CIS Benchmark Level 1 in the infrastructure layer achieves high accuracy, reflecting the binary and well-defined nature of configuration checks as well as the quality of the configuration data provided by the Agent [102].
- The rule group corresponding to OWASP A02/A05/A06 in the application layer performs at an acceptable level, given its dependence on HTTP-derived data and external vulnerability databases. Most False Positive/False Negative cases arise in scenarios where application behavior or configuration characteristics fall outside the initial assumptions, and these cases have been analyzed in detail in Chapter 5 [113].

From the theoretical (Chapters 2-3) and experimental (Chapters 4-5) results, it can be concluded that:

- Modeling standard requirements into YAML rules and executing them within a unified Rule Engine is feasible and well-suited to the WordPress environment running on Apache/Nginx [105].
- The architecture that integrates multiple evidence sources (HTTP/HTML Recon, vulnerability intelligence, and configuration Agent data) enables parallel assessment of both the application layer and the infrastructure layer under a unified set of standards, helping to mitigate the fragmentation inherent in standalone tools [114].
- At the Proof of Concept level, the tool has demonstrated clear potential to serve as a useful component in security compliance testing within a DevSecOps context, provided that it is integrated and extended appropriately in accordance with the identified constraints [109].

(AI Contribution to the Overall System)

Although the core compliance evaluation relies on deterministic rule execution, the integration of an AI-assisted diff analysis pipeline based on MiniLM semantic embeddings and Google Gemini adds an interpretive layer to the system. This AI subsystem enhances the explanatory quality of compliance reports, automatically highlights meaningful changes between scans, and generates human-readable remediation narratives [92.2]. While it is not part of the core detection engine, these AI-driven capabilities significantly improve usability, interpretability, and the platform's overall alignment with DevSecOps practices.

6.2. Achievement of Research Objectives

Based on the research objectives presented in Chapter 1 and further operationalized through the design, implementation, and evaluation in Chapters 3-5, the thesis assesses the degree of objective fulfillment as follows.

6.2.1. Objective 1 - Design an integrated two-layer compliance testing architecture

The first objective is to design an architecture capable of integrating data from the application layer (WordPress core, plugins, themes, HTTP/HTML behavior) and the infrastructure layer (Apache/Nginx configuration on a Linux environment) into a unified model, ensuring that every finding is mapped to its corresponding reference standard (OWASP/CIS) and technical evidence source [89].

The implementation results confirm that this objective has been achieved:

1. The system implements a modular client-server architecture with dedicated modules for Recon, vulnerability lookup, configuration Agent, and the Rule Engine, ensuring

clear separation of roles, data sources, and responsibilities. Detailed architectural descriptions and data-flow diagrams were presented in Chapter 3 [100].

2. The unified Target Snapshot consolidates data from multiple sources, providing the Rule Engine with a consistent, synchronized view, while the dashboard presents results by standard, severity, rule type, and layer (application/infrastructure), fully aligning with the single scan - multi-standard objective [105].
3. This architecture also satisfies several important non-functional requirements: minimizing privileged access on the target server (via a least-privilege Agent), avoiding interference with production traffic, and reducing dependency on any single scanning tool through abstraction at the Rule Engine layer [108].

Therefore, the proposed model not only fulfills the goal of integrating both layers (application and infrastructure) but also establishes a robust technical foundation for future expansion without requiring a complete architectural redesign.

6.2.2. Objective 2 - Implementing Compliance as Code

The second objective is to formalize security and configuration requirements into a structured rule set (75 YAML rules mapped to OWASP A02/A05/A06 and CIS Level 1 for Apache/Nginx), together with a Rule Engine capable of executing these rules against standardized input data, fully decoupled from the application's source code [107].

Achieved results:

- The YAML rule set has been constructed following a unified schema and is directly mapped to the corresponding clauses in OWASP/CIS, consistent with the specification defined in Chapter 3 (including: rule_id, standard, description, severity, type, target, assertion, rationale, remediation, etc.) [99].
- The Rule Engine processes this rule set independently of the data-collection logic, enabling rules to be modified, extended, or reapplied without touching the system's core backend. When new standards need to be added or checking logic must be updated, modifications occur primarily at the rule layer rather than in source code [105].
- Every Failed finding on the dashboard can be fully traced back to its originating rule_id, its corresponding standard, and the specific fields inside the Target Snapshot (observed header, agent directive, plugin/theme info, etc.), ensuring the level of traceability defined earlier as a mandatory requirement [106].

Successfully implementing this model proves that descriptive compliance requirements (OWASP/CIS texts) can be transformed into machine-readable and machine-executable

rules while still preserving transparency and explainability critical for interactions with Dev, Ops, Security, and Risk Management teams.

6.2.3. Objective 3 - Building a Multi-Source Technical Evidence Collection Mechanism

The third objective focuses on constructing a multi-source data collection mechanism, comprising ba thành phần thiết yếu: HTTP/HTML Recon to gather WordPress fingerprints, security-related HTTP headers, returned HTML content, and other behavioral indicators [101]; External vulnerability lookup for WordPress core, plugins, and themes [116]; and a remote configuration Agent capable of reading and verifying Apache/Nginx configurations and returning normalized Pass/Fail results aligned with CIS Level 1 directives [97].

As demonstrated in Chapters 3, 4, and 5, this objective has been fully achieved. The Recon module and vulnerability lookup module jointly supply data for OWASP rules (A02, A05, A06), enabling checks related to headers (HSTS, X-Frame-Options, CSP), transport configurations, WordPress core version, and plugin/theme vulnerabilities or outdated states [113]. Crucially, the configuration Agent is implemented using the principle of least privilege, returning only normalized Pass/Fail outcomes (or minimal structured values) instead of entire configuration files, ensuring confidentiality while still enabling deep visibility into server configuration [100]. Furthermore, combining these three evidence sources into a single Target Snapshot empowers the Rule Engine to evaluate cross-layer rules. For example, the system can evaluate an A06 rule examining outdated plugins combined with hardening checks at the server layer, offering a more realistic compliance posture rather than viewing risks in isolation.

Thus, the multi-source collection mechanism is not a simple aggregation of tools it is a coordinated design optimized for OWASP/CIS mapping.

6.2.4. Objective 4 - Developing the Compliance Dashboard and Experimental Evaluation Framework

The fourth objective was dual-focused: First, to develop a unified compliance dashboard summarizing results by standard, severity, Pass/Fail state, and individual rule; and second, to establish an empirical evaluation framework based on the Golden Label Dataset, Confusion Matrix, Precision, Recall, and F1-Score for assessing the tool's accuracy.

Regarding the achieved results, the dashboard delivers a consolidated overview of OWASP/CIS compliance, severity distribution, and Passed/Failed rule counts, while supporting drill-down to each rule with detailed evidence (observed headers, agent

directives, plugin/theme status, CVE entries) and actionable remediation. This functionality successfully shortens the gap between technical detection and operational remediation [111].

Furthermore, an empirical evaluation framework has been fully realized. A Golden Label Dataset was constructed with intentionally crafted compliant/non-compliant scenarios for both application and infrastructure layers. When combined with the Confusion Matrix, this allows precise calculation and comparison of Precision, Recall, and F1-Score across rule groups (OWASP and CIS, application vs infrastructure), enabling evaluation not only of functional correctness but also of statistical reliability [109]. Significantly, using these quantitative metrics also helps identify rule groups with higher error rates, providing a data-driven basis for refining rule definitions in future iterations avoiding subjective judgment or guesswork [110].

AI-assisted Interpretation (Supplement to Objective 4)

In addition to traditional reporting, the system incorporates an AI-based analysis pipeline to assist with interpreting scan results. By applying MiniLM embeddings to measure semantic similarity across scans and invoking Gemini to generate concise explanations, the system provides higher-level insights such as identifying partial fixes, regressions, or unchanged vulnerabilities across time. This extends the dashboard from a static reporting tool into a more intelligent assistant for compliance and remediation workflows.

To provide an overall comparison between the research objectives and the achieved results, the following summary can be made:

Research Objective	**Summary of Achieved Results**	**Evaluation**
Design an integrated two-layer compliance testing architecture	Modular client–server architecture implemented, unified ingestion of data from Recon, vulnerability lookup module, and configuration Agent into the Rule Engine and dashboard.	Achieved
Implement Compliance as Code	Developed 75 YAML rules for OWASP A02/A05/A06 and CIS L1, Rule Engine separates rule definitions from application logic.	Achieved

Develop a multi-source technical data collection mechanism	Completed the Recon module, vulnerability lookup module, and configuration Agent, standardized and unified all evidence sources into the Target Snapshot.	Achieved
Develop a compliance reporting interface and an empirical evaluation framework	Implemented a detailed dashboard, established the Golden Label Dataset, measured performance using Confusion Matrix, Precision, Recall, and F1-Score.	Achieved

Table 6.1: Summary of Research Objective Completion

Overall, the initial research objectives have been largely fulfilled, consistent with the project's defined scope, available resources, and intended direction. The outcomes form a solid methodological and technical foundation for future studies, extensions, and real-world deployments following the same architectural approach.

6.3. Limitations

Alongside the results achieved, the thesis still contains several inherent limitations. Identifying these limitations is essential in order to: (i) place the conclusions of the study into their proper context, (ii) avoid overstating the capabilities of the Proof of Concept system, and (iii) establish a foundation for prioritizing future research and development.

The limitations discussed below are derived from the scope defined in Chapter 1, the system architecture and implementation in Chapter 3, and the experimental results and analyses in Chapters 4 and 5.

6.3.1. Limitations in Technology Scope and Supported Standards

The system was intentionally designed to focus on a relatively narrow set of technologies and security standards. Specifically, the system limits its scope to a single application platform: WordPress (single-site), deployed in a typical configuration using WordPress core, official plugins, and themes [104]. This is coupled with only two types of web servers: Apache and Nginx on Linux (Ubuntu), without considering other variants such as Windows/IIS or less common web servers [102]. Furthermore, the project addresses only a subset of security standards: OWASP Top 10 (A02, A05, A06) and CIS Benchmark Level 1 for Apache/Nginx, covering only the requirements that can be automated within the system's technical boundaries [99].

While this constrained scope allows the thesis to focus on depth and avoid unmanageable complexity, it also leads to several consequences. First, other platforms are not directly supported. Although the Rule Engine and Target Snapshot model are reusable, adopting them for other platforms would still require additional analysis and development [116]. Second, other compliance standards (or higher levels of the same standard, such as CIS Level 2) are not implemented in the PoC. As a result, organizational, administrative, or highly complex technical requirements fall outside the system's coverage [97]. Finally, even within the selected standards (OWASP A02/A05/A06 and CIS L1), the thesis only implements requirements that can be automated, leaving out checks that demand manual evaluation, contextual business analysis, or user interaction [101].

Therefore, the current Proof of Concept system remains limited in applicability and is primarily suitable for WordPress deployments on Apache/Nginx in environments similar to the testbed constructed in this study. Extending beyond this scope (both in technology and compliance standards) should be considered follow-up research rather than an implied capability of the current system.

6.3.2. Limitations in Data Sources and Methodological Approach

The system relies on three data sources: HTTP/HTML Reconnaissance, external vulnerability databases, and the on-server Configuration Agent. The thesis prioritizes a snapshot-based model collecting information at one (or several) points in time rather than continuous monitoring. While this approach simplifies deployment and minimizes impact on target systems, it also introduces several limitations.

First, there is a lack of runtime behavioral visibility. Vulnerabilities that only appear during specific business workflows (multi-step interactions, session-dependent states, race conditions, etc.) will not be observed unless they occur within the limited URL set and scenarios covered by the Recon module [113].

Second, the system has a strong dependency on external vulnerability databases. Since the system does not maintain its own vulnerability repository and instead relies on WordPress-specific external sources, two consequences arise: If a vulnerability has not yet been publicly disclosed or has not been updated in the external data source, the system will not be able to flag the corresponding plugin/theme as risky [114]. Conversely, any inaccuracies or delays in the update cycle of the external vulnerability source will directly impact the accuracy of the OWASP A06 rule group [115]. Furthermore, the accuracy of several OWASP A02/A05/A06 rules depends on HTTP fingerprinting and the interpretation of information extracted from headers and HTML content. Exceptional cases

such as customized error pages, reverse proxies masking the origin server, or complex rewrite configurations may cause the system to misinterpret the context, leading to False Positives or False Negatives, as discussed in Chapter 5 [116].

In addition, to balance visibility with the need to protect system confidentiality, the configuration Agent is intentionally designed to return only processed values (Pass/Fail or a minimal set of attributes) instead of the full configuration file contents. While this decision successfully reduces the risk of exposing sensitive information, it also means that the Rule Engine does not possess complete configuration context, thereby limiting its ability to analyze highly complex or abnormal cases.

Limitations of AI-Assisted Interpretation: the AI subsystem (MiniLM + Gemini) enhances interpretability but is not used for primary compliance decisions. Its outputs depend on training data and semantic generalisation, so contextual misunderstandings or imprecise summarisation may still occur. Consequently, AI-generated narratives should be regarded as supportive explanations rather than authoritative compliance judgements. In addition, ensuring consistency across different versions of external AI models remains a challenge for long-term reproducibility [117].

In summary, the snapshot-based approach and the reliance on external data sources are both strengths (simple deployment, low invasiveness) and inherent sources of limitations, which have been reflected in the system's error analysis.

6.3.3. Limitations of the Evaluation Environment and Deployment Scale

The experiments in this thesis were conducted within a controlled laboratory environment, in which WordPress/Apache/Nginx instances were deployed on a virtualization/container platform using configurations designed by the researcher [104]. Compliant and non-compliant configuration scenarios were intentionally constructed to create the Golden Label Dataset used for evaluation. This approach enables tight control over variables, ensures reproducibility, and simplifies result comparison. However, it also introduces several limitations, notably that it does not fully capture the diversity of real-world environments [106]. In production deployments, systems often involve multiple middleware layers, reverse proxies, CDNs, WAFs, load balancers, and various legacy components that are difficult to reproduce in a lab setting, factors that can significantly affect HTTP reconnaissance results, configuration visibility, and runtime application behavior.

Furthermore, the scale and frequency of scanning remain limited. The experiments focused on a relatively small number of targets and did not thoroughly evaluate: How the system behaves when scanning dozens or hundreds of websites within a short timeframe; the

impact of repeated, high-frequency scans on system resources (CPU, RAM, bandwidth) for both the tool and the target systems [110]; or the scalability of the platform when deployed in an enterprise context with many concurrent WordPress sites [111].

In addition, operational challenges of deploying the Agent in production environments were not fully examined. In the lab environment, installing and running the Agent is straightforward because it is not subject to strict access control, change-management processes, or environment segregation (dev/test/prod). In real-world scenarios, installing new components on production servers typically requires risk assessments, approvals, and additional testing-factors that were not replicated in this thesis. These limitations do not diminish the value of the experimental results; however, they underscore that the conclusions regarding system accuracy and performance should be interpreted within the context of a controlled testbed, and must be revalidated when applied to more diverse and complex deployment environments [108].

These limitations do not diminish the value of the experimental results; however, they underscore that the conclusions regarding system accuracy and performance should be interpreted within the context of a controlled testbed and must be revalidated when applied to more diverse and complex deployment environments [108].

6.3.4. Summary of Limitations and Their Linkage to Future Development

To more clearly link the main limitations with the development directions proposed in Section 6.4, the relationship can be summarized as follows:

Main Limitation	Proposed Development Direction (Summary)
Limited technological scope and standards	Expand the rule set to additional relevant standards, apply the methodology to other open-source web platforms with similar architectures.
Dependence on snapshot-based evidence and external vulnerability data	Integrate additional analysis signals at a level appropriate to the system's scope.
Evaluation environment restricted to laboratory settings	Extend the Golden Label Dataset, conduct experiments with configurations closer to real-world deployments, increasing complexity and diversity.

Limited deployment scale and operational breadth	Evaluate performance under multiple targets, optimize data collection and processing pipelines, assess deployment and operational considerations for the Agent in enterprise environments.

Figure 6.2. Summary of Limitations and Corresponding Development Directions.

This consolidated table not only functions as a summary, but also serves as a roadmap for transforming the aspects that the thesis has not yet fully addressed into concrete tasks for future work. The detailed content corresponding to each development direction will be presented in Section 6.4.

6.4. Future Work

Based on the limitations identified in Section 6.3 and the discussions presented in Chapters 4 and 5, several future development directions can be outlined to further enhance the integrated, multi-standard compliance testing model for WordPress on Apache/Nginx. These directions aim both to deepen the current scope and to enable controlled expansion, while preserving the core philosophy of Compliance as Code [105].

6.4.1. Refinement and Deepening Within the Current Scope

Before expanding to additional platforms or standards, the highest priority is to improve the tool within the existing WordPress and Apache/Nginx environments. This strengthens the reliability of the Proof of Concept and avoids premature broadening when the current foundation still holds opportunities for improvement [100].

One essential task is expanding and diversifying the Golden Label Dataset. Rather than relying solely on handcrafted configuration scenarios in a lab environment, future research could incorporate anonymized, real-world configuration samples to reflect practical deployment patterns. This would increase the practical significance of Precision, Recall, and F1-Score metrics and reveal edge cases not covered by the current dataset [109].

Additionally, rules with a high False Positive/False Negative rate identified in the qualitative analysis in Chapter 5 should be reviewed and fine-tuned. Rules in the OWASP A02/A05/A06 groups, which rely heavily on HTTP fingerprinting and external vulnerability APIs, are prime candidates for such refinement. Improvements may include adjusting assertion conditions, adding thresholds, or incorporating additional contextual fields available in the Target Snapshot [113].

Finally, scalability should be further studied as the number of targets increases. Follow-up work may design large-scale scanning scenarios with higher frequency to measure response time, resource consumption, and system load. From this, specific optimizations can be proposed for example: refining Recon strategies to reduce unnecessary requests, caching vulnerability lookups, or batching checks with shared data sources [115].

6.4.2. Integration into DevSecOps and CI/CD Pipelines

Another important direction is to evolve the tool from a standalone scanner into a component of a DevSecOps workflow by integrating it into CI/CD pipelines. Instead of ad-hoc periodic scanning, the tool could be triggered automatically whenever critical changes occur in infrastructure configuration or WordPress components [106].

Future studies may create scripts, plugins, or sample jobs for pipeline integration. Each time configuration files or a new WordPress version is committed, the pipeline may automatically run a compliance scan. If critical rules fail, deployment can be halted and remediation required. This aligns with the shift-left principle in DevSecOps by detecting misconfigurations as early as possible [108].

Moreover, standardizing outputs in structured formats and providing integration hooks or APIs would enable connectivity with existing systems such as issue trackers, change-management platforms, or monitoring systems. Failed findings could automatically translate into actionable tickets with responsibility assignment and deadlines. Over time, such data would help track compliance trends, evaluate configuration hygiene, and support security audits [112].

Integrating the tool into DevSecOps/CI/CD does not change its technical core but changes how it is used from a post-deployment scanner to a continuous quality-control component [106].

6.4.3. Expansion to Related Platforms and Standards

Once reliability is strengthened within the WordPress and Apache/Nginx scope, a natural future direction is expanding the rule set and methodology to related platforms and standards, while retaining the same architectural and Compliance as Code principles [105]

From a standards perspective, future research may incorporate additional groups of requirements from other information-security frameworks provided that such requirements can be automated and mapped to technical checks based on configuration, components, or observable behaviors. For example, several clauses related to TLS configuration, component control, logging, and infrastructure hardening in various security frameworks can be transformed into YAML rules following the same structural format [99]

From a platform perspective, the methodology presented in this thesis can be applied to other open-source web systems that share architectural similarities with WordPress in terms of deployment. However, this process is not merely a matter of copying rules, it requires:

- Re-analyzing differences in deployment and configuration models [103]
- Building or adapting Recon modules, vulnerability lookup sources, and Agent capabilities [116]
- Selecting automatable requirements from the relevant standards [97]
- Constructing a corresponding testbed and Golden Label Dataset [98]

Each such expansion resembles a mini-thesis cycle requiring repetition of the methodological steps demonstrated in this research: requirement analysis, rule modeling, data-collection design, experimental evaluation, and quantitative validation. This ensures methodological rigor and avoids unsupported claims of applicability.

6.4.4. Enhancing Analytical Capabilities and Remediation Support

Finally, a set of future directions focuses on strengthening the depth of analysis and improving remediation assistance to narrow the gaps identified in Section 6.3.

First, additional analytical signals may be incorporated at a reasonable level of complexity, such as:

- Static Application Security Testing (SAST) for WordPress plugins/themes, enabling the detection of insecure coding patterns that can be directly mapped to relevant OWASP rules [113].
- Selective log analysis to identify behavioral patterns associated with insecure configurations or outdated components [115].

These signal sources do not need to be deeply integrated immediately, instead, they may gradually be standardized and added to the Target Snapshot for potential rule integration in future iterations.

Second, the remediation field already attached to each rule can be leveraged to extend operator support during the remediation process. Examples include:

- Generating actionable checklists based on the set of Failed rules in a scan.
- Suggesting configuration commands, files to modify, or relevant documentation for each group of findings [100].
- Grouping findings into action bundles to allow operators to resolve issues more efficiently by thematic category [104].

6.4.5. Advancing AI-Driven Compliance Intelligence

Future research may expand the AI subsystem to support deeper compliance reasoning, anomaly detection and automated remediation assistance in several directions:

Improved semantic diffing and scenario classification.

Enhancing the MiniLM embedding pipeline or adopting domain-specific LLM embeddings could increase the precision of detecting meaningful compliance changes across complex before/after snapshots and deployment scenarios [92.1].

AI-based rule suggestion and automatic rule generation.

Large language models could be leveraged to semantically parse CIS/OWASP requirements and propose candidate YAML rules, reducing manual effort when extending the rule set while keeping humans in the review loop [92.2].

Predictive compliance analytics.

With longitudinal scan data, AI models could predict which configurations or components are most likely to drift into non-compliance, enabling proactive risk mitigation rather than purely reactive fixes [104.1].

AI-assisted remediation automation.

Future work may integrate AI agents capable of generating fix scripts, configuration updates or patching instructions based on failed findings, executed under human supervision and change-management controls [92.2].

Natural-language compliance querying.

Operators could interact with the system through questions such as: Which CIS directives failed this week? or What changed since the last deployment?, using LLM-driven query interpretation on top of structured scan data.

Collectively, these directions move the system towards an Application Security Posture Management (ASPM) model, where rule-based compliance checks and AI-driven intelligence coexist within a unified platform. They remain aligned with the Compliance as Code philosophy: every finding and recommendation is tied to a transparent, traceable and auditable rule, while narrowing the gap between detecting a misconfiguration or vulnerability and actually fixing it in real operational environments [104.2] .

6.5. Closing Remarks

This thesis has focused on a specific but practically meaningful problem: developing a multi-standard integrated security compliance testing tool for WordPress on Apache/Nginx, grounded in the Compliance as Code (CaC) philosophy. Through the standardized rule set, well-defined system architecture, and quantitative evaluation framework, the thesis provides not only a Proof of Concept technical demonstration but also a structured, interpretable, and extensible approach for future research and real-world deployment in security compliance testing [105].

Chapters 1 through 5 sequentially addressed the background, theoretical foundations, architectural design, system implementation, and experimental evaluation. Building on that foundation, Chapter 6 consolidates the key findings, compares them against the research objectives, identifies limitations, and proposes future development directions. Viewed holistically, the thesis demonstrates that transforming the requirements of standards such as OWASP A02/A05/A06 and CIS Benchmark Level 1 into YAML based rules and executing them within a unified Rule Engine is a feasible approach to reducing the gap between the language of standards and the language of technical implementation in web application security practice [99].

Although the system still has limitations and cannot yet cover the entire security landscape of open-source web platforms, its results can be regarded as a systematic starting point laying the groundwork for the development of multi-standard security compliance testing platforms in the future. By emphasizing the role of Compliance as Code, the thesis encourages shifting compliance considerations earlier into the development and operational lifecycle, helping organizations move toward a proactive, consistent, and standards-driven model of information-security governance not only in the context of WordPress on Apache/Nginx, but also as an inspiration for broader future research across diverse technological environments [106].

REFERENCES

[1] NBER. “Open-source software creators: It’s not just about the money.” NBER. https://www.nber.org/be/20241/open-source-software-creators-its-not-just-about-money?page=1&perPage=50 [accessed Oct. 2, 2025].

[2] HIPAA Journal. “Open source security risks.” HIPAA Journal. https://www.hipaajournal.com/open-source-security-risks/ [accessed Oct. 2, 2025].

[3] ISACA. “ISACA Supply Chain Security Report Spotlights Major Vulnerabilities. https://www.isaca.org/resources/news-and-trends/newsletters/atisaca/2022/volume-23/isaca-supply-chain-security-report-spotlights-major-vulnerabilities [accessed Oct. 2, 2025].

[4] Legit Security. “10 top open-source software risks and how to mitigate them.” Application Security Posture Management from Code to Cloud. https://www.legitsecurity.com/aspm-knowledge-base/open-source-software-risks [accessed Oct. 2, 2025].

[5] N. Schäferhoff. “WordPress market share, statisticsand more.” WordPress.com News. https://wordpress.com/blog/2025/04/17/wordpress-market-share/ [accessed Oct. 2, 2025].

[6] D. Knauss. “The 2022 WordPress vulnerability annual report.” SolidWP. https://solidwp.com/blog/the-2022-wordpress-vulnerability-annual-report/ [accessed Oct. 2, 2025].

[7] OWASP Foundation. “OWASP Top 10 Risks for Open Source Software.” OWASP Foundation. https://owasp.org/www-project-open-source-software-top-10/ [accessed Oct. 2, 2025].

[8] Amazon Web Services. “What are CIS benchmarks? – CIS benchmarks explained.” Amazon Web Services. https://aws.amazon.com/what-is/cis-benchmarks/ [accessed Oct. 2, 2025].

[9] Center for Internet Security. “Benchmarks Overview – CIS®.” CIS. https://www.cisecurity.org/cis-benchmarks-overview [accessed Oct. 2, 2025].

[10] R. Severns. “OWASP ZAP: Open source app security testing.” StackHawk. https://www.stackhawk.com/blog/guide-to-zap-application-security-testing/ [accessed Oct. 2, 2025].

[11] ZAP by Checkmarx. “Getting started.” ZAP by Checkmarx. https://www.zaproxy.org/getting-started/ [accessed Oct. 2, 2025].

[12] H. S. Abdullah. “Evaluation of Open Source Web Application Vulnerability Scanners.” Semantic Scholar. https://pdfs.semanticscholar.org/2050/726ef329933cf59e01a2c768e1efda2880bf.pdf [accessed Oct. 2, 2025].

[13] U.-S. Potti, H.-S. Huang, H.-T. Chenand H.-M. Sun. “Security Testing Framework for Web Applications: Benchmarking ZAP v2.12.0 and v2.13.0 by OWASP as an example.” arXiv. https://arxiv.org/abs/2501.05907 [accessed Oct. 2, 2025].

[14] WPScan. “WPScan: The Ultimate In-Depth Guide to WordPress Security Scanning.” Geek Institute Blog. https://blog.geekinstitute.org/2025/09/wpscan-ultimate-in-depth-guide-to-wordpress-security-scanning.html [accessed Oct. 2, 2025].

[15] WPScan. “WPScan User Documentation.” WPScan. https://github.com/wpscanteam/wpscan/wiki/WPScan-User-Documentation [accessed Oct. 2, 2025].

[16] S. Farrell. “Abstraction and automation of WordPress vulnerability scanning.” NCIRL Repository (NORMA). https://norma.ncirl.ie/6515/1/stephenfarrell.pdf [accessed Oct. 2, 2025].

[17] T. Jose. “WPSCAN – discovering the vulnerabilities and enumerating users of WordPress sites with Autotor for IP spoofing.” Academia.edu. https://www.academia.edu/75824296/WPscan_Discovering_the_Vulnerabilities_and_Enumerating_Users_of_WordPress_Sites_with_Autotor_for_IP_Spoofing [accessed Oct. 2, 2025].

[18] freeCodeCamp. “An introduction to web server scanning with Nikto.” freeCodeCamp. https://www.freecodecamp.org/news/an-introduction-to-web-server-scanning-with-nikto/ [accessed Oct. 2, 2025].

[19] KSEC ARK. “Nikto web server scanner.” Ivoid Warranties Tech. https://www.ivoidwarranties.tech/posts/pentesting-tuts/website-pentesting/nikto/ [accessed Oct. 2, 2025].

[20] Hackers4U. “What Are the Limitations of Nikto in Modern Web Security?” 2025. https://www.hackers4u.com/what-are-the-limitations-of-nikto-in-modern-web-security [accessed Oct. 2, 2025].

[21] S. B., S. NRK, T. J.and S. S. “A comparative analysis of Vulnerability Management Tools: Evaluating Nessus, Acunetix and Nikto for risk based Security Solutions.” arXiv. https://arxiv.org/pdf/2411.19123 [accessed Oct. 2, 2025].

[22] ResearchGate. “Vulnerability Scanners – A Proactive Approach to Assess Web Application Security.” ResearchGate. https://www.researchgate.net/publication/261182006_Vulnerability_Scanners-A_Proactive_Approach_To_Assess_Web_Application_Security [accessed Oct. 2, 2025].

[23] MalCare. “12 best WordPress vulnerability scanners 2024: Tested for real detection.” MalCare. https://www.malcare.com/blog/wordpress-vulnerability-scanners/#3-wpscan [accessed Oct. 2, 2025].

[24] OWASP Foundation. “OWASP Top Ten.” OWASP Foundation. https://owasp.org/www-project-top-ten/ [accessed Oct. 2, 2025].

[25] Agile approach for software security projects (rapid responses to changes): OWASP DevSecOps Guideline - OWASP DevSecOps Guideline | OWASP Foundation [accessed Oct. 20, 2025].

[26] Iterative development with Scrum, early validation of modules: What is Agile? | Agile 101 - Agile Alliance - What is Agile? | Agile 101 | Agile Alliance [accessed Oct. 20, 2025].

[27] Client-server architecture for stability, scalability, security: Client-Server Architecture - System Design - GeeksforGeeks - Client-Server Architecture - System Design - GeeksforGeeks [accessed Oct. 20, 2025].

[28] Separation of concerns, maintainable architecture (RESTful APIs, future capabilities): Anatomy of the Client/Server Model - Oracle Documentation - Anatomy of the Client/Server Model [accessed Oct. 20, 2025].

[29] Compliance as Code philosophy: What is Compliance as Code? The Best Way to Automate ... - Puppet - What is Compliance as Code? The Best Way to Automate Compliance Testing + Enforcement | Puppet [accessed Oct. 20, 2025].

[30] DevSecOps approach: OWASP DevSecOps Guideline - OWASP DevSecOps Guideline | OWASP Foundation [accessed Oct. 20, 2025].

[31] Modular monolith architecture: Modular Monolith: Is This the Trend in Software Architecture? - ArXiv - Modular Monolith: Is This the Trend in Software Architecture? [accessed Oct. 20, 2025].

[32] PDCA cycle: PDCA Cycle - What is the Plan-Do-Check-Act Cycle? - ASQ - PDCA Cycle - What is the Plan-Do-Check-Act Cycle? | ASQ [accessed Oct. 20, 2025].

[33] Standards (OWASP Top 10, CIS Benchmarks): OWASP Top Ten - OWASP Top Ten | OWASP Foundation và CIS Apache HTTP Server Benchmarks - cisecurity.org/benchmark/apache_http_server [accessed Oct. 20, 2025].

[34] Non-intrusive data collection (HTTP responses, WPScan API, agent checks): WordPress Vulnerability Database API - WPScan - WordPress Vulnerability Database API | WPScan [accessed Oct. 20, 2025].

[35] Reconnaissance & HTTP Response Analysis: OWASP Top Ten (A05: Security Misconfiguration) - OWASP Top Ten | OWASP Foundation [accessed Oct. 20, 2025].

[36] External Vulnerability API Querying: WPScan API Documentation - WPScan API Documentation [accessed Oct. 20, 2025].

[37] Remote Configuration File Scanning via Agent: CIS Benchmarks (general principles) - CIS Benchmarks® [accessed Oct. 20, 2025].

[38] Standardizing input data: OWASP DevSecOps Guideline (data handling) - OWASP DevSecOps Guideline | OWASP Foundation [accessed Oct. 20, 2025].

[39] Compliance as Code, Rule Engine: What is Compliance as Code? - Puppet - What is Compliance as Code? The Best Way to Automate Compliance Testing + Enforcement | Puppet [accessed Oct. 20, 2025].

[40] Rule Structure, Input Data Constraints, Execution Workflow: OWASP DevSecOps Guideline - OWASP DevSecOps Guideline | OWASP Foundation [accessed Oct. 20, 2025].

[41] Control and Quality Principles, Outputs: PDCA Cycle - ASQ - PDCA Cycle - What is the Plan-Do-Check-Act Cycle? | ASQ [accessed Oct. 20, 2025].

[42] Reliance on fingerprinting, scope of static scanning, external database, agent installation: OWASP Top Ten (A06: Vulnerable Components) - OWASP Top Ten | OWASP Foundation [accessed Oct. 20, 2025].

[44] Back-end (NodeJS, Express): Node.js Documentation - Index | Node.js v25.0.0 Documentation; Express Official - Express - Node.js web application framework [accessed Oct. 20, 2025].

[45] Service (Python): Python Official Documentation - 3.14.0 Documentation [accessed Oct. 20, 2025].

[46] WPScan: WPScan API Documentation - WPScan API Documentation [accessed Oct. 20, 2025].

[47] Database (MongoDB): MongoDB Official - MongoDB: The World's Leading Modern Database | MongoDB [accessed Oct. 20, 2025].

[48] Overview of the system's operational process: Client-Server Architecture - GeeksforGeeks - Client-Server Architecture - System Design - GeeksforGeeks [accessed Oct. 20, 2025].

[51] "Describing the UI – React," react.dev. https://react.dev/learn/describing-the-ui (accessed Oct. 20, 2025).

[52] Magda, "TypeScript - a scalable and reliable language for web applications," *Software House - rozwiązania IT dla Twojego biznesu | UniqueDevs*, May 26, 2025. https://uniquedevs.com/en/blog/typescript-a-scalable-and-secure-choice-for-large-projects/ (accessed Oct. 20, 2025).

[53] Vite, "Getting Started," vitejs, 2019. https://vite.dev/guide/ (accessed Oct. 20, 2025).

[54] Redux, "Redux - A predictable state container for JavaScript apps.," Js.org, 2015. https://redux.js.org/ (accessed Oct. 20, 2025).

[55] GeeksforGeeks, "NodeJS Introduction," GeeksforGeeks, Jan. 21, 2019. https://www.geeksforgeeks.org/node-js/node-js-introduction/ (accessed Oct. 20, 2025).

[56] OpenJS Foundation, "Express - Node.js web application framework," Expressjs.com, 2017. https://expressjs.com/ (accessed Oct. 20, 2025).

[57] Python Software Foundation, “What Is Python? Executive Summary,” Python, 2025. https://www.python.org/doc/essays/blurb/ (accessed Oct. 20, 2025).

[58] “WordPress Vulnerability Database API,” WPScan, Jul. 10, 2023. https://wpscan.com/api/ (accessed Oct. 20, 2025).

[59] IBM, “What Is MongoDB?,” Ibm.com, Oct. 14, 2021. https://www.ibm.com/think/topics/mongodb (accessed Oct. 20, 2025).

[60] “Redis Queue - Redis,” Redis, Aug. 24, 2023. https://redis.io/glossary/redis-queue/ (accessed Oct. 21, 2025).

[61] OWASP Foundation. (2021). *OWASP Top 10: 2021*. https://owasp.org/www-project-top-ten/ [accessed Nov. 25, 2025].

[62] Felderer, M., et al. (2016). Security testing: A survey. *Advances in Computers*, 101, 1-51. https://doi.org/10.1016/bs.adcom.2015.11.003 [accessed Nov. 25, 2025].

[63] Sokolova, M., & Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4), 427-437. https://doi.org/10.1016/j.ipm.2009.03.002 [accessed Nov. 25, 2025].

[64] Merkel, D. (2014). Docker: Lightweight Linux containers for consistent development and deployment. *Linux Journal*, 2014(239), 2. https://www.linuxjournal.com/content/docker-lightweight-linux-containers-consistent-development-and-deployment [accessed Nov. 25, 2025].

[65] Melapress. (2024). Using OWASP Top 10 to improve WordPress security. https://melapress.com/owasp-wordpress-security-top-10/ [accessed Nov. 25, 2025].

[66] CIS. (2023). *CIS Apache HTTP Server Benchmarks*. https://www.cisecurity.org/benchmark/apache_http_server [accessed Nov. 25, 2025].

[67] Bass, L., et al. (2012). *Software Architecture in Practice* (3rd ed.). Addison-Wesley. https://www.pearson.com/us/higher-education/program/Bass-Software-Architecture-in-Practice-3rd-Edition/PGM183302.html [accessed Nov. 25, 2025].

[67.1] Wang, W., Wei, F., Dong, L., Bao, H., Yang, N., & Zhou, M. MiniLM: Deep

Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers. Microsoft Research. 2020.

https://arxiv.org/pdf/2002.10957 [accessed Nov. 25, 2025].

[67.2] DeepMind. Gemini: A Family of Highly Capable Multimodal Models. Google DeepMind Technical Report. 2023.

https://arxiv.org/pdf/2312.11805 [accessed Nov. 25, 2025].

[68] Reaves, B., et al. (2016). The Power of Qualitative Methods: Aha Moments in Exploring Cybersecurity and Trust. NIST Journal of Research. https://csrc.nist.gov/pubs/journal/2016/11/the-power-of-qualitative-methods-cybersecurity-and/final [accessed Nov. 25, 2025].

[69] Wohlin, C., et al. (2012). Experimentation in Software Engineering. Springer. https://doi.org/10.1007/978-3-642-29044-2 [accessed Nov. 25, 2025].

[70] Drata. (2025). Compliance as Code: Tutorial and Best Practices. https://drata.com/grc-central/compliance-as-code/what-is-cac [accessed Nov. 25, 2025].

[71] Practical DevSecOps. (2024). DevSecOps Metrics for Measuring Security Effectiveness. https://www.practical-devsecops.com/devsecops-metrics/ [accessed Nov. 25, 2025].

[72] Hackers4U. (2025). What's the Difference Between Nikto and OWASP ZAP? https://www.hackers4u.com/whats-the-difference-between-nikto-and-owasp-zap [accessed Nov. 25, 2025].

[73] Wiz. (2025). What is Application Security Posture Management (ASPM)? https://www.wiz.io/academy/application-security-posture-management-aspm [accessed Nov. 25, 2025].

[74] OWASP. (2021). A02:2021 - Cryptographic Failures. OWASP Top 10. https://owasp.org/Top10/A02_2021-Cryptographic_Failures/ [accessed Nov. 25, 2025].

[75] MITRE Corporation. (2021). CWE-1346: OWASP Top Ten 2021 Category A02 - Cryptographic Failures. Common Weakness Enumeration. https://cwe.mitre.org/data/definitions/1346.html [accessed Nov. 25, 2025].

[76] Authgear. (2025). Comprehensive Guide to Cryptographic Failures (OWASP Top 10 A02). https://www.authgear.com/post/cryptographic-failures-owasp [accessed Nov. 25, 2025].

[77] Security Journey. (2023). OWASP Top 10 Cryptographic Failures: Definition, Examples, & Solutions.

https://www.securityjourney.com/post/owasp-top-10-cryptographic-failures-explained [accessed Nov. 25, 2025].

[77.1] PointGuard AI. (2025). What Is Application Security Posture Management and Why You May Need It.

https://www.pointguardai.com/blog/what-is-application-security-posture-management-and-why-you-may-need-it [accessed Dec. 12, 2025].

[78] Center for Internet Security. (n.d.). CIS Apache HTTP Server Benchmarks. https://www.cisecurity.org/benchmark/apache_http_server [accessed Nov. 25, 2025].

[79] Center for Internet Security. (n.d.). CIS Benchmarks. https://www.cisecurity.org/cis-benchmarks [accessed Nov. 25, 2025].

[80] Tenable. (2023). CIS Apache HTTP Server 2.4 L1 v2.0.0. https://www.tenable.com/audits/CIS_Apache_HTTP_Server_2.4_Benchmark_v2.0.0_Level_1 [accessed Nov. 25, 2025].

[81] Oezdil, M. (2024). On The Compliance as a Code (CaC) Security. Medium. https://mesutoezdil.medium.com/on-the-compliance-as-a-code-cac-security-46d600b71bfa [accessed Nov. 25, 2025].

[82] RegScale. (n.d.). Shift Left Security with Compliance as Code. https://regscale.com/continuous-compliance-for-devsecops/ [accessed Nov. 25, 2025].

[83] XenonStack. (2024). Compliance as Code - Complete Guide. https://www.xenonstack.com/blog/compliance-as-a-code/ [accessed Nov. 25, 2025].

[84] Drata. (n.d.). DevSecOps Principles: Tutorial and Best Practices. https://drata.com/grc-central/compliance-as-code/devsecops-principles [accessed Nov. 25, 2025].

[85] GitProtect. (2024). Integrating Security as Code: A Necessity for DevSecOps. https://gitprotect.io/blog/integrating-security-as-code-a-necessity-for-devsecops/ [accessed Nov. 25, 2025].

[86] WPScan. (n.d.). WPScan: WordPress Security Scanner. https://wpscan.com/ [accessed Nov. 25, 2025].

[87] WordPress.org. (n.d.). WPScan – WordPress Security Scanner Plugin. https://wordpress.org/plugins/wpscan/ [accessed Nov. 25, 2025].

[88] WPScan Team. (n.d.). wpscanteam/wpscan. GitHub. https://github.com/wpscanteam/wpscan [accessed Nov. 25, 2025].

[89] Center for Internet Security. (n.d.). CIS NGINX Benchmarks. https://www.cisecurity.org/benchmark/nginx [accessed Nov. 25, 2025].

[90] Tenable. (2024). CIS NGINX Benchmark v2.0.1 L1 Webserver. https://www.tenable.com/audits/CIS_NGINX_v2.0.1_Level_1_Webserver [accessed Nov. 25, 2025].

[91] Acunetix. (2020). nginx Security: How To Harden Your Server Configuration. https://www.acunetix.com/blog/web-security-zone/hardening-nginx/ [accessed Nov. 25, 2025].

[92] Upwind Security. (2025). Best Open-Source DevSecOps Tools in 2025. https://www.upwind.io/glossary/13-best-devsecops-tools-2025s-best-open-source-options-sorted-by-use-case [accessed Nov. 25, 2025].

[92.1] Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using Siamese BERT-networks. ACL.

https://arxiv.org/pdf/1908.10084 [accessed Dec. 12, 2025].

[92.2] Paduraru, C., Dumitru, B., & Stefanescu, A. (2025). Automated Generation of Cybersecurity Response Playbooks via Large Language Models. Procedia Computer Science.

https://www.researchgate.net/publication/397358266_Automated_Generation_of_Cybersecurity_Response_Playbooks_via_Large_Language_Models [accessed Dec. 12, 2025].

[93] Anchore. (2021). 7 Tips to Create a DevSecOps Open Source Strategy. https://anchore.com/blog/devsecops-open-source-strategy-7-tips/ [accessed Nov. 25, 2025].

[94] Hackers4U. (2025). What's the Difference Between Nikto and OWASP ZAP? https://www.hackers4u.com/whats-the-difference-between-nikto-and-owasp-zap [accessed Nov. 25, 2025].

[95] Kowalczyk, P., & Gerber, N. (2022). Comparative analysis of the effectiveness of OWASP ZAP, Burp Suite, Nikto and Skipfish in detecting vulnerabilities in web applications. Journal of Computer Science and Information. https://ph.pollub.pl/index.php/jcsi/article/view/2929 [accessed Nov. 25, 2025].

[96] WASP Foundation. (n.d.). Vulnerability Scanning Tools. https://owasp.org/www-community/Vulnerability_Scanning_Tools [accessed Nov. 25, 2025].

[97] CIS benchmarks for securing WordPress. TeamUpdraft. October 6, 2025. https://teamupdraft.com/blog/cis-benchmarks-for-securing-wordpress/ [accessed Nov. 25, 2025].

[98] CIS benchmarks for WordPress: What it means and where to start. Liquid Web. No date. https://www.liquidweb.com/wordpress/security/cis-benchmarks/ [accessed Nov. 25, 2025].

[99] CIS Benchmarks®. CIS Center for Internet Security. No date. https://www.cisecurity.org/cis-benchmarks [accessed Nov. 25, 2025].

[100] Server-Level WordPress Security Hardening. MOSS. November 23, 2025. https://moss.sh/reviews/server-level-wordpress-security-hardening/ [accessed Nov. 25, 2025].

[101] 15-Step Guide to Secure WordPress on Ubuntu and Nginx (OWASP Top 10). Reddit /r/cybersecurity. June 18, 2025. https://www.reddit.com/r/cybersecurity/comments/1lef332/15step_guide_to_secure_wordpress_on_ubuntu_nginx/ [accessed Nov. 25, 2025].

[102] CIS NGINX Benchmarks. CIS Center for Internet Security. No date. https://www.cisecurity.org/benchmark/nginx [accessed Nov. 25, 2025].

[103] Best practices for Apache Server hardening? Stack Exchange. November 11, 2010. https://security.stackexchange.com/questions/77/best-practices-for-apache-server-hardening [accessed Nov. 25, 2025].

[104] The Most Practical WordPress Performance and Security Guide. InstaWP. March 26, 2025. https://instawp.com/wordpress-performance-and-security-guide/ [accessed Nov. 25, 2025].

[104.1] Çakmak, A., & Karabacak, B. (2025). A Novel Approach to Patched Software Lifecycle Management in Cyber Ecosystems.

https://ieeexplore.ieee.org/document/8470942 [accessed Dec. 12, 2025].

[104.2] Wiz Academy. (2025). Application Security Posture Management (ASPM).

https://www.wiz.io/academy/application-security/application-security-posture-management-aspm [accessed Dec. 12, 2025].

[105] Compliance as Code - Complete Guide. XenonStack. August 6, 2024. https://www.xenonstack.com/blog/compliance-as-a-code/ [accessed Nov. 25, 2025].

[106] From Compliance to Continuous Security: Why DevSecOps Needs Continuous Compliance. CloudBees. No date. https://www.cloudbees.com/blog/devsecops-continuous-compliance [accessed Nov. 25, 2025].

[107] On The Compliance as a Code (CaC) Security. Mesut Oezdil. Medium. October 21, 2024. https://mesutoezdil.medium.com/on-the-compliance-as-a-code-cac-security-46d600b71bfa [accessed Nov. 25, 2025].

[108] Security-as-Code: A Key Building Block for DevSecOps. ISACA. May 31, 2024. https://www.isaca.org/resources/news-and-trends/isaca-now-blog/2024/security-as-code-a-key-building-block-for-devsecops [accessed Nov. 25, 2025].

[109] DevSecOps in 2025: Principles, Technologies & Best Practices. Oligo Security. September 29, 2025. https://www.oligo.security/academy/devsecops-in-2025-principles-technologies-best-practices [accessed Nov. 25, 2025].

[110] 6 Best Practices for Implementing DevSecOps. Qovery. No date. https://www.qovery.com/blog/6-best-practices-for-implementing-devsecops [accessed Nov. 25, 2025].

[111] Automating Compliance in DevSecOps to improve AppSec Posture. OpsMx. February 20, 2025. https://www.opsmx.com/blog/transforming-appsec-posture-with-devsecops-compliance-automation/ [accessed Nov. 25, 2025].

[112] DOD Enterprise DevSecOps Activities & Tools Guidebook. U.S. Department of Defense. September 15, 2025. https://dodcio.defense.gov/Portals/0/Documents/Library/DevSecOpsActivitesToolsGuidebook.pdf [accessed Nov. 25, 2025].

[113] Top 10 Application Security Testing Tools (2025 Edition). OX Security. September 11, 2025. https://www.ox.security/blog/application-security-testing-tools/ [accessed Nov. 25, 2025].

[114] Top 10 Dynamic Application Security Testing (DAST) Tools for 2025. Jit. No date. https://www.jit.io/resources/appsec-tools/top-dast-tools-for-2024 [accessed Nov. 25, 2025].

[115] 20 Best Security Assessment Tools in 2025. Qualysec. No date. https://qualysec.com/security-assessment-tools/ [accessed Nov. 25, 2025].

[116] Free for Open Source Application Security Tools. OWASP Foundation. No date. https://owasp.org/www-community/Free_for_Open_Source_Application_Security_Tools [accessed Nov. 25, 2025].

[117] Bommasani, R., et al. (2021). On the Opportunities and Risks of Foundation Models.

https://crfm.stanford.edu/report.html [accessed Dec. 12, 2025].

APPENDIX

Appendix

Appendix A. Ground Truth Assignment for 10 Evaluated Websites

This appendix presents the set of ground truths manually assigned to each website during the system evaluation process.

Ground truths determine the actual status of each Vulnerability ID (TRUE: exists, FALSE: does not exist) for the purpose of measuring TP, FP, and FN.

Vuln_ ID	**Site 1**	**Site 2**	**Site 3**	**Site 4**	**Site 5**	**Site 6**	**Site 7**	**Site 8**	**Site 9**	**Site 10**
V1	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE
V2	...	...	...	...	...	...	...	...	...	...
V3										
V4	...	...	...	...	...	...	...	...	...	...
V5										
V6										
V23	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE

Note:

The full table contains 23 rows and 10 columns.

Detailed data can be referenced when calculating TP, FN, and FP for each website.

Appendix B. SecRuleMap Detection Results for 10 Websites

The table below shows SecRuleMap's predictions for each Vulnerability ID for 10 websites.

The results are used to calculate the TP, FP, and FN of each site.

Vuln_ ID	**Site 1**	**Site 2**	**Site 3**	**Site 4**	**Site 5**	**Site 6**	**Site 7**	**Site 8**	**Site 9**	**Site 10**

V1	1	0	1	1	0	1	0	1	0	1
V2	0	1	0	0	1	0	0	1	0	0
...	...	...	...	...	...	...	...	...	...	...
V23	1	1	0	1	0	0	1	0	1	0

Note:

1 = SecRuleMap detects vulnerability

0 = SecRuleMap does not detect vulnerability

APPENDIX C. Ground Truth Matrix for 10 Kali Apache Servers

This appendix presents the ground truth (TRUE/FALSE) for 28 Apache CIS 2.4 rules tested on 5 Kali Linux servers.

The ground truth was determined manually based on th_e actual configuration of each server.

Vuln ID	**Rule ID**	**Server 1**	**Server 2**	**Server 3**	**Server 4**	**Server 5**
V1	CIS-APACHE-2.4-2.2	TRUE	TRUE	FALSE		
V2	CIS-APACHE-2.4-2.3	TRUE	FALSE	TRUE		
...	...	...	...	...		
V28	CIS-APACHE-2.4-8.2	TRUE	TRUE	FALSE		

Appendix B. SecRuleMap Detection Matrix

Vuln_ID	**Server 1**	**Server 2**	**Server 3**	**Server 4**	**Server 5**
V1	1	0	1	1	0
V2	0	1	0	0	1

...	...	...	...	...	...
V23	1	1	0	1	0