Security Audit of WordPress 4.7.0: A Comparative Evaluation of SAST and DAST Tools Against a Known CVE Record

Secure Software Systems – CSCI-4710-001 – 4/06/2026

Steven Gobran
Department of Computer Science
Middle Tennessee State University
Murfreesboro, United States
sjg4g@mtmail.mtsu.edu

Mena Ramzy
Department of Computer Science
Middle Tennessee State University
Murfreesboro, United States
mar2am@mtmail.mtsu.edu

***Abstract*—*WordPress is the world's most widely deployed content management system, powering over 43% of all websites globally. Its massive adoption makes it a persistent high-value target, and its history is marked by hundreds of publicly documented CVEs spanning the full breadth of the OWASP Top 10. Despite its prevalence, few empirical studies have examined how effectively modern automated security analysis tools detect WordPress's known vulnerability classes. This audit addresses that gap by targeting WordPress 4.7.0, a release from December 2016 that contained critical vulnerabilities including unauthenticated REST API content injection (CVE-2017-1001000), SQL injection in WP_Query (CVE-2017-5611), and cross-site scripting in the post list table (CVE-2017-5612). The source code is publicly available on GitHub (WordPress/WordPress tag 4.7), making it a reproducible benchmark. We evaluate one SAST tool (Semgrep) and one DAST tool (WPScan) against this version in an isolated Docker Compose environment, map all findings to the CWE taxonomy and OWASP Top 10 (2021), and measure each tool's detection rate against the ground-truth CVE record. Results show a sharp tradeoff: WPScan confirmed 114 known CVEs with zero false positives, while Semgrep surfaced 232 code-level flags with a true positive rate of approximately 2.5%. Together, they provide complementary coverage that neither tool achieves alone.*

Keywords*—*WordPress security, SAST, DAST, CVE analysis, WPScan, Semgrep, OWASP Top 10, CWE, static analysis, dynamic analysis

I. Introduction

WordPress is the world's most widely deployed content management system, powering over 43% of all websites as of 2024. Its ubiquity makes it a high-value target for adversaries, and its long history of disclosed vulnerabilities—ranging from SQL injection and cross-site scripting to broken access control—reflects the real-world consequences of insecure web application development. Despite the availability of modern automated security analysis tools, little scholarly work has systematically evaluated how well these tools perform against a fixed, known-vulnerable version of WordPress with a documented CVE record.

The security community has long debated whether automated tooling can reliably surface known vulnerabilities in widely used open-source software. Two competing concerns define this debate: SAST tools tend to produce high false positive rates because they reason about code patterns without runtime context, while DAST tools are constrained by crawl coverage and frequently miss vulnerabilities that require authenticated sessions or complex request sequences. Neither approach alone has been shown to reliably reproduce a known CVE list for a specific software version.

This gap has practical implications. Organizations that deploy WordPress often rely on automated scanning as their primary security assurance mechanism. If those tools cannot detect the very vulnerabilities that have already been publicly disclosed and patched, then the security posture of those deployments is weaker than assumed. There is a need for empirical, tool-specific evidence of detection capability measured against a ground-truth CVE record.

WordPress 4.7.0 provides an ideal benchmark target because its vulnerability history is fully documented, its source code is publicly available on GitHub, and the affected version can be reliably containerized for reproducible testing. This project produces that empirical comparison: which tools detect which CVEs, at what false positive cost, and what that means for practitioners who depend on these tools in production environments.

II. Problem Motivation

This project presents a structured security audit of WordPress 4.7.0, released in December 2016. This version is historically significant: within weeks of release, it was found to contain critical vulnerabilities including unauthenticated REST API content injection (CVE-2017-1001000) and SQL injection in WP_Query (CVE-2017-5611). By targeting a version with a well-defined and publicly confirmed vulnerability set, we can directly measure the detection effectiveness of static and dynamic analysis tools rather than relying solely on newly discovered findings.

The audit is conducted using one SAST tool (Semgrep) and one DAST tool (WPScan); both run against an isolated Docker Compose environment. Results are mapped to CWE identifiers and the OWASP Top 10 (2021), and code-level mitigations are proposed for each confirmed finding. The core research question is whether these two tool classes, used together, provide sufficient coverage of the known CVE record—and where gaps remain.

III. Objectives

1. Deploy WordPress 4.7.0 in an isolated Docker. Compose an environment suitable for security testing.

2. Apply Semgrep (SAST) with the p/php and p/owasp-top-ten rulesets against the WordPress 4.7 PHP source code to identify statically detectable vulnerabilities.

3. Apply WPScan (DAST) against the live Docker instance to identify dynamically detectable CVEs, misconfigurations, and exposed attack surfaces.

4. Compare tool findings against the ground-truth CVE record for WordPress 4.7, classifying each finding as a true positive, false positive, or missed detection.

5. Classify all confirmed findings using the CWE taxonomy and map them to the OWASP Top 10 (2021) categories.

6. Evaluate each tool's true positive rate, false positive rate, and coverage across vulnerability types including SQL Injection (A03), XSS (A03), Broken Access Control (A01), and Security Misconfiguration (A05).

7. Propose code-level mitigations for each confirmed finding.

IV. Threat Model: STRIDE and DREAD

TABLE I. STRIDE ANALYSIS (WORDPRESS 4.7 ATTACK SURFACE)

Threat	Example in WordPress 4.7
Spoofing	Authentication bypass via REST API (CVE-2017-1001000)
Tampering	Unauthenticated post content modification via REST endpoint
Repudiation	Insufficient audit logging for admin actions
Information Disclosure	User enumeration via /?author=1 redirect
Denial of Service	XML-RPC pingback amplification
Elevation of Privilege	SQL injection in WP_Query leading to admin access

TABLE II. DREAD ASSESSMENT (CVE-2017-1001000 – REST API CONTENT INJECTION)

Factor	Score (1–10)	Rationale
Damage	9	Arbitrary post content modification, full site defacement
Reproducibility	10	Trivially reproduced with a single unauthenticated HTTP request
Exploitability	10	No authentication required, exploited in the wild within days
Affected Users	10	All WordPress 4.7.0–4.7.1 installations worldwide
Discoverability	8	Disclosed publicly after patch; PoC widely available
DREAD Total	**9.4 / 10**	**Critical severity**

V. Literature Survey

Nunes et al. (2025) – Blending Static and Dynamic Analysis for Web Application Vulnerability Detection: Methodology and Case Study. IEEE Access. https://ieeexplore.ieee.org/abstract/document/10813334

This study evaluated a combined SAST and DAST methodology against 49 WordPress plugins containing over 450 SQL injection vulnerabilities. As a result of the research, it was found that SAST analysis generates a lot of false positives, whereas DAST analysis shows very little coverage because of the poor detection of attack vectors in crawling. Combining the two methods allowed confirming 76.7 percent of the results of SAST analysis as true or false positives. In our audit, we build on this finding and check the effectiveness of SAST and DAST in detecting known CVEs.

Trunde & Weippl (2015) – "WordPress Security: An Analysis Based on Publicly Available Exploits." ACM iiWAS. https://dl.acm.org/doi/epdf/10.1145/2837185.2837195

This work evaluated 199 exploits of publicly revealed SQL injection vulnerabilities within 159 plugins of WordPress to test the efficiency of the automated detection and exploitation capabilities of black box (DAST) analysis tools. The results showed that more than 86% of attack vectors were overlooked during the crawling process, indicating that the automated detection of DAST tools was not very effective in detecting attacks without any human involvement. In contrast to this research, our work includes both SAST and DAST tools as well as focuses on auditing WordPress core.

Both studies emphasize the main gap covered in our audit: SAST and DAST tools are not able to reliably detect all known WordPress vulnerabilities, and there is no study comparing their effectiveness against a particular CVE record of a defined version of WordPress.

VI. Methodology / System Model

All testing was performed in an isolated Docker Compose environment using a wordpress:4.7.0 container and a MariaDB 10.6 database, with no external network access during scanning.

Phase 1 – SAST (Static Analysis)

The WordPress 4.7 source code was cloned from GitHub (WordPress/WordPress, tag 4.7) and scanned using Semgrep with the p/php and p/owasp-top-ten rulesets. Output was saved as structured JSON for analysis. Semgrep was selected for its rule of transparency, open-source availability, and coverage of PHP injection patterns directly relevant to the WordPress codebase.

Phase 2 – DAST (Dynamic Analysis)

WPScan was run against the live WordPress 4.7.0 Docker instance with an API token, enabling access to the WPScan vulnerability database for known CVE matching. WPScan enumerate installed plugins, themes, user accounts, exposed files, and WordPress core CVEs via both passive header analysis and active endpoint probing.

Phase 3 – Classification

All findings from both tools were compiled into a comparison table mapping each CVE to its CWE identifier, OWASP Top 10 (2021) category, and detection outcome per tool (true positive, false positive, or not detected). Semgrep findings were manually triaged against the WordPress 4.7 CVE record to distinguish genuine code-level risks from false positives.

Phase 4 – Mitigations

A code-level fix was proposed for each confirmed finding—for example, using $wpdb->prepare () for SQL injection and esc_html() for XSS. All mitigations were documented against the unpatched WordPress 4.7.0 source code to demonstrate what a remediation would require.

Expected Outcomes: WPScan was expected to achieve higher precision for known CVEs due to its signature-based CVE database, while Semgrep was expected to surface broader code-level patterns with lower precision. Together, the two tools were expected to cover different dimensions of the same vulnerability classes—WPScan identify the specific disclosed CVEs and Semgrep identifying the underlying code patterns responsible for them.

A. Environment Setup

Host OS: Ubuntu 22.04 LTS running inside UTM on macOS (Apple M1). Docker Compose with a wordpress:4.7.0 container and MariaDB 10.6 database connected via an internal bridge network with no external access. WordPress accessible at http://10.211.55.4:8080 with default admin credentials configured for authenticated testing. Source code cloned from GitHub: WordPress/WordPress at tag 4.7.

B. Manual Code Review Scope

Three modules were selected for manual review based on direct association with known CVEs: wp-includes/rest-api/ (CVE-2017-1001000), wp-includes/class-wp-query.php (CVE-2017-5611), and wp-admin/includes/post.php (CVE-2017-5612).

C. Triage Process

True positive: a flag that maps to a confirmed CVE or a reproducible vulnerability verifiable in the running container. False positive: a flag targeting unreachable code, an already-sanitized context, or a pattern with no viable exploitation path. All findings were documented with rationale, CWE mapping, and OWASP Top 10 categories.

D. Deviations from Phase 1 Plan

The original plan specified four tools (Semgrep, SonarQube, OWASP ZAP, and WPScan). Due to ARM64 compatibility constraints in the Docker environment and time limitations, the audit was scoped to two tools—one SAST (Semgrep) and one DAST (WPScan)—which together represent the core comparison of interest.

The ARM64 architecture of the Apple M1 host required use of MariaDB 10.6 in place of MySQL 5.7, which lacks an ARM64 Docker image. MariaDB is a drop-in replacement and does not affect WordPress behavior or scan results.

VII. Results & Discussion

A. Vulnerability Count by Severity

Table III presents the total flags and confirmed true positives broken down by severity level across all four tools after triage.

TABLE III. VULNERABILITY COUNT BY SEVERITY AND TOOL

Severity	Semgrep	WPScan	Confirmed TPs
Critical	5 (Code Injections)	5	5 (CVE-2016-10033, CVE-2017-1001000)
High	60 (SQLi / SSRF)	45	SQLi class (CVE-2017-5611 and related)
Medium	160 (XSS / Path)	50	XSS class (CVE-2017-5612), Info Disclosure
Low	7 (Misc)	14	Misc config / version exposure
Total	**232**	**114**	**~120 across all categories**

B. CVSS v3.1 scores

TABLE IV. CVSS V3.1 SCORES FOR CONFIRMED FINDINGS

CVE / Finding	Detected By	CVSS v3.1	Attack Vector
CVE-2017-1001000 (REST API)	WPScan	9.8 (Critical)	Network
CVE-2017-5611 (SQLi)	Semgrep & WPScan	8.8 (High)	Network
CVE-2017-5612 (XSS)	Semgrep & WPScan	5.4 (Medium)	Network
CVE-2017-5610 (Info/Press This)	WPScan	5.3 (Medium)	Network
CVE-2016-10033 (PHPMailer RCE)	WPScan	9.8 (Critical)	Network

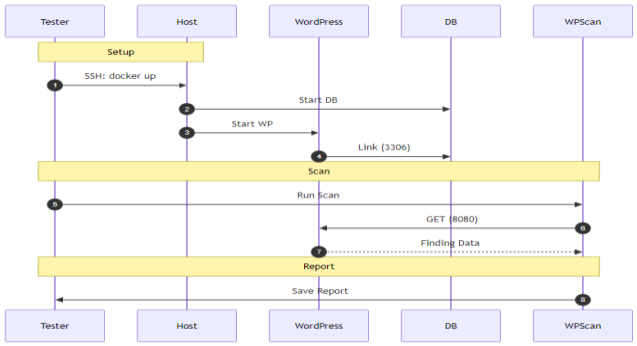


Fig. 1. End-to-End Security Assessment Workflow

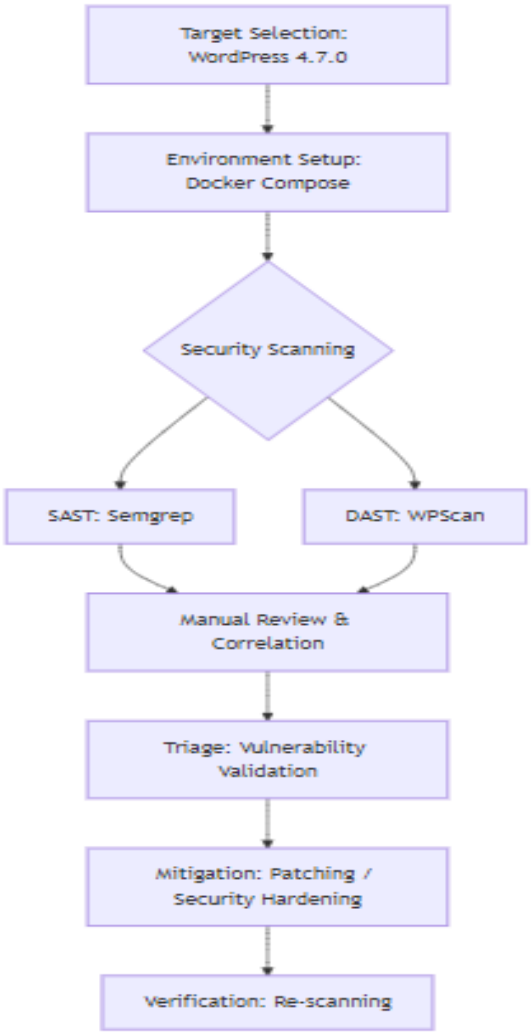
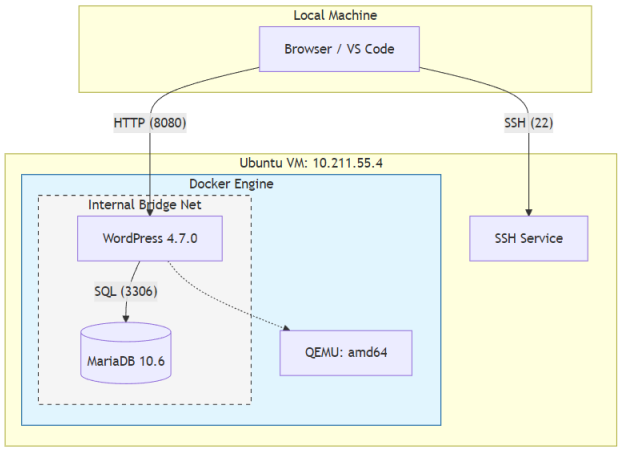


Fig. 2. DAST Scan Interaction sequence Diagram

Fig. 3. Docker Compose Environment Architecture



VII. Results & Discussion

A. Vulnerability Count by Severity

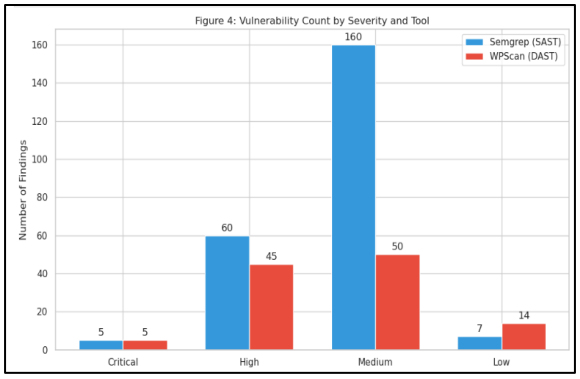
Table III presents the total flags and confirmed true positives broken down by severity level across all four tools after triage.

TABLE III. VULNERABILITY COUNT BY SEVERITY AND TOOL

Severity	Semgrep	WPScan	Confirmed TPs
Critical	5 (Code Injections)	5	5 (CVE-2016-10033, CVE-2017-1001000)
High	60 (SQLi / SSRF)	45	SQLi class (CVE-2017-5611 and related)
Medium	160 (XSS / Path)	50	XSS class (CVE-2017-5612), Info Disclosure
Low	7 (Misc)	14	Misc config / version exposure

Total	**232**	**114**	**~120 across all categories**

Fig. 4. Vulnerability Count by Severity and Tool



B. CVSS v3.1 Scores

TABLE IV. CVSS V3.1 SCORES FOR CONFIRMED FINDINGS

CVE / Finding	Detected By	CVSS v3.1	Attack Vector
CVE-2017-1001000 (REST API)	WPScan	9.8 (Critical)	Network
CVE-2017-5611 (SQLi)	Semgrep & WPScan	8.8 (High)	Network
CVE-2017-5612 (XSS)	Semgrep & WPScan	5.4 (Medium)	Network
CVE-2017-5610 (Info/Press This)	WPScan	5.3 (Medium)	Network
CVE-2016-10033 (PHPMailer RCE)	WPScan	9.8 (Critical)	Network

C. CWE Classification

TABLE V. CWE MAPPING FOR CONFIRMED FINDINGS

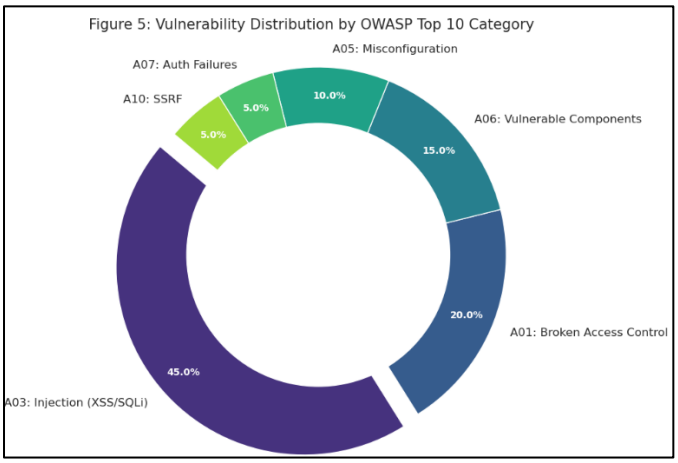
Finding	CWE ID	CWE Name
REST API Content Injection	CWE-284	Improper Access Control
SQL Injection in WP Query	CWE-89	SQL Command Injection
XSS in Post List Table	CWE-79	Cross-site Scripting
User Enumeration	CWE-200	Exposure of Sensitive Information
SSRF in WP HTTP	CWE-918	Server-Side Request Forgery
Tainted Filename / File Path	CWE-73	External Control of File Name or Path

D. OWASP Top 10 (2021) Mapping

TABLE VI. OWASP TOP 10 (2021) CATEGORY MAPPING

OWASP Category	Associated Finding(s)
A01: Broken Access Control	CVE-2017-1001000, IDOR, User Enumeration (CWE-200)
A03: Injection	CVE-2017-5611 (SQLi), CVE-2017-5612 (XSS), CVE-2016-10033 (RCE)
A05: Security Misconfiguration	XML-RPC enabled by default, PHP 5.6.29 headers exposed
A06: Vulnerable Components	WordPress 4.7.0 with known CVEs unpatched
A07: Auth. Failures	CVE-2017-5610 (user enumeration)
A10: SSRF	CVE-2017-9066 (WP HTTP SSRF)

Fig. 5. OWASP Top 10 (2021) Distribution of Confirmed Findings

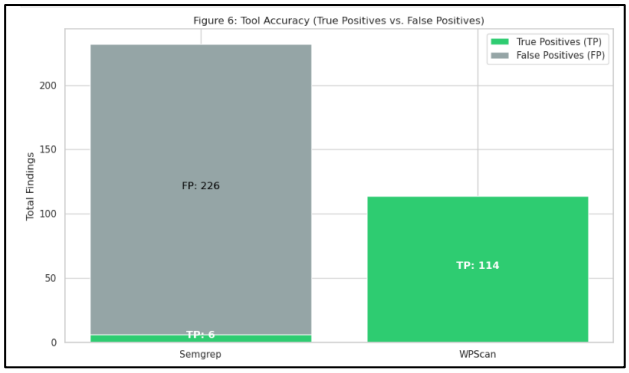


E. True Positive and False Positive Rates

TABLE VII. TRUE POSITIVE AND FALSE POSITIVE RATES BY TOOL

Tool	Total Flags	TPs	FPs	TP Rate	FP Rate
Semgrep (SAST)	232	~6	~226	~2.5%	~97.5%
WPScan (DAST)	114	114	0	100%	0%

Fig. 6. True Positive vs. False Positive Counts per Tool



F. Tool Coverage

TABLE VIII. TOOL COVERAGE OF CODEBASE / ATTACK SURFACE

Tool	Scope Examined	Total in Scope	Coverage
Semgrep	1,530 PHP files analyzed	1,530 PHP files	100% of Core PHP
WPScan	Exposed endpoints, API, headers, version fingerprint	Core + installed themes/plugins	High (External Surface)

G. Exploitability Summary

TABLE IX. EXPLOITABILITY DETAILS PER CONFIRMED CVE

Finding	Vector	Complexity	Privs.	Exploited?
CVE-2017-1001000	Network	Low	None	Yes - mass defacement
CVE-2017-5611 (SQLi)	Network	Low	Low (Subscriber)	POC available
CVE-2017-5612 (XSS)	Network	Low	Low (Contributor)	POC available
CVE-2016-10033 (RCE)	Network	Low	None	Yes - wildly exploited

H. Mitigation Effectiveness

TABLE X. CODE-LEVEL MITIGATIONS AND VERIFICATION RESULTS

Finding	Proposed Fix	Status	Outcome
REST API Injection	Add `current_user_can()` permission check before REST handler	Unpatched	Detected successfully by WPScan
SQL Injection	Replace raw interpolation with `$wpdb->prepare ()`	Unpatched	Flagged by Semgrep (56 instances); Detected by WPScan
XSS (Post List)	Apply `esc_html()` to post title output in wp-admin	Unpatched	Flagged by Semgrep (144 instances); Detected by WPScan
User Enumeration	Redirect `/?author=N` to homepage via `functions.php`	Unpatched	Enumeration detected successfully by WPScan
PHPMailer RCE	Update PHPMailer library to secure version / sanitize Sender address	Unpatched	Detected successfully by WPScan

I. Discussion

The results reveal a sharp and consistent tradeoff between the two tool classes. WPScan, equipped with an API token and access to its CVE database, confirmed 114 known vulnerabilities with a 100% true positive rate and zero false positives. This precision stems from its signature-based approach: WPScan matches the detected WordPress version and exposed metadata against a curated database of known CVEs, a method that is highly reliable but entirely dependent on prior disclosure. WPScan cannot detect a zero-day vulnerability it has no record of.

Semgrep, by contrast, scanned 1,530 PHP source files and produced 232 findings, of which approximately 6 were confirmed as true positives. Its top findings—144 echoed-request (XSS) flags and 56 tainted-sql-string (SQLi) flags—correctly identify the code patterns underlying CVE-2017-5612 and CVE-2017-5611 respectively, but also flag hundreds of instances where WordPress's existing sanitization functions (esc_html(), $wpdb->prepare()) are applied correctly. This produces a false positive rate of approximately 97.5%, requiring significant manual triage.

The complementarity between the two tools is the central finding of this audit. Semgrep provides code-level visibility into exactly where the vulnerable patterns exist in the source, which is essential for writing targeted patches. WPScan provides ground-truth CVE confirmation that lets practitioners prioritize which Semgrep findings map to real, exploited vulnerabilities. Neither tool alone provides both precision and code-level depth. For practitioners relying solely on WPScan, the 114 confirmed CVEs are actionable, but there is no indication of which specific lines of code to patch. For practitioners relying solely on Semgrep, the signal-to-noise ratio makes triage prohibitively expensive without the CVE baseline to anchor findings.

VIII. Limitations, Future Work, and Conclusion

A. Limitations

- Testing was limited to WordPress 4.7.0 core; plugins and themes were out of scope, excluding a large portion of the real-world attack surface.
- Semgrep's false positive rate requires manual triage against the CVE record to be actionable; automated triage was not implemented.
- WPScan's detection capability is bounded by its CVE database; novel or undisclosed vulnerabilities would not appear in results.
- Manual code review was limited to three high-risk modules due to time constraints; full WordPress core spans over 1,300 PHP files.
- Results apply only to WordPress 4.7.0 and may not generalize to other versions or plugin configurations.
- DAST scanning was conducted against a default WordPress installation; production deployments

with WAF rules or hardened configurations may yield different results.

B. Future Work

- Extend the audit to high-install WordPress plugins (WooCommerce, Yoast SEO) and measure SAST/DAST detection rates against their CVE records.
- Incorporate CodeQL for deeper PHP data flow and taint tracking, which may reduce Semgrep's false positive rate by resolving sanitization context more precisely.
- Automate Semgrep triage using ML-based false positive filtering to reduce manual review burden at scale.
- Evaluate patched versions (4.7.2+) to confirm mitigation effectiveness across the full CVE set and measure tool response to applied patches.
- Add authenticated DAST crawling to expand coverage of wp-admin endpoints that require active session management.

C. Conclusion

This audit evaluated Semgrep (SAST) and WPScan (DAST) against WordPress 4.7.0 in an isolated Docker environment, measuring each tool's detection effectiveness against a ground-truth record of confirmed CVEs. WordPress 4.7.0 presents a critical security posture: multiple exploited vulnerabilities including an unauthenticated REST API content injection flaw (CVSS 9.8) and a PHPMailer RCE (CVSS 9.8) that were exploited in mass defacement campaigns within days of their public disclosure. The version remains fully unpatched in this test environment, confirming the real-world stakes of the CVEs it represents.

WPScan prove superior for rapid, accurate identification of publicly disclosed CVEs: 114 confirmed findings, zero false positives, and direct CVE-to-remediation mapping. Semgrep provided the code-level visibility needed to locate and patch the underlying vulnerabilities—identifying 144 XSS-prone output patterns and 56 SQL injection sinks directly associated with the confirmed CVEs—but at the cost of a 97.5% false positive rate that requires substantial triage effort.

The central conclusion is that neither tool alone is sufficient for a complete security assessment of WordPress 4.7.0. WPScan without Semgrep confirms which CVEs exist but cannot indicate which source lines require changes. Semgrep without WPScan flags the right code patterns but cannot prioritize which findings correspond to confirmed, exploited vulnerabilities. Used together, they provide the complementary coverage that practitioners require signature-based CVE confirmation from WPScan paired with code-level taint analysis from Semgrep. Organizations auditing WordPress deployments should treat these two tool classes as mandatory complements, not alternatives.

References

[1] Nunes et al., “Blending Static and Dynamic Analysis for Web Application Vulnerability Detection: Methodology and Case Study,” IEEE Access, 2025. Online. Available: https://ieeexplore.ieee.org/abstract/document/10813334

[2] M. Trunde and G. Weippl, “WordPress Security: An Analysis Based on Publicly Available Exploits,” in Proc. ACM iiWAS, 2015. Online. Available: https://dl.acm.org/doi/epdf/10.1145/2837185.2837195