Journal Publication of International Research for Engineering and Management (JOIREM)
Volume: 04 Issue: 4 | April-2026
ISSN (O) 3107-6696

AWVERT: Automated Web Vulnerability Exploitation and Reporting Tool

Moorthi R[1], Adhitthyan K G[2], Joshua Fernando[3]

[1,2,3]*Department of Computer Science and Engineering (Cyber Security), Sri Venkateswaraa College of Technology*

Abstract - Web application vulnerabilities, particularly SQL Injection (SQLi) and Cross-Site Scripting (XSS), continue to dominate the global threat landscape, accounting for over 25% of documented security breaches annually. Existing tools such as Burp Suite, OWASP ZAP, and Nikto, while valuable, require significant manual intervention and lack the ability to leverage real-world Cyber Threat Intelligence (CTI) for automated payload testing. This paper presents AWVERT (Automated Web Vulnerability Exploitation and Reporting Tool), a Python-based automated vulnerability scanner for web applications. AWVERT integrates a Breadth-First Search (BFS) web crawler with support for both classic server-rendered and Single Page Application (SPA) architectures using headless Chromium via Playwright. The tool tests six injection categories — SQL Injection, Cross-Site Scripting, PHP/Command Injection, HTML Injection, XML/XXE Injection, and NoSQL Injection — using CTI-sourced payloads from a structured vault. Detection employs three modes: reflection analysis, error-signature matching, and statistical time-based inference. Tested against the OWASP Juice Shop intentionally vulnerable application, AWVERT demonstrates a detection rate of 77.8% across all injection types with a false positive rate of 9.0%.

Keywords: Web Vulnerability Scanner, SQL Injection, Cross-Site Scripting, Cyber Threat Intelligence, BFS Web Crawling, Automated Security Testing, Injection Attack Detection

1. INTRODUCTION

Web applications are the primary interface between organizations and their users, processing sensitive personal, financial, and operational data at scale. However, they remain among the most actively targeted components of modern IT infrastructure. Industry breach data consistently identifies injection-based attacks — including SQL Injection (SQLi) and Cross-Site Scripting (XSS) — as a leading cause of compromise, representing roughly a quarter to a third of documented security incidents in recent annual reports [1]. Injection vulnerabilities have remained among the highest-severity categories in the OWASP Top Ten across multiple successive editions, underscoring their enduring relevance in both legacy and contemporary web environments [13].

Despite widespread adoption of Web Application Firewalls (WAFs) as a perimeter control, a significant proportion of obfuscated injection payloads — exploiting encoding transformations, comment injection, and Unicode substitution — successfully bypass these defenses, as demonstrated by recent evasion studies [4]. This gap between defensive perimeter technology and real-world attacker sophistication has driven growing interest in automated, intelligence-driven vulnerability assessment.

Existing penetration testing tools such as Burp Suite, OWASP ZAP, and Nikto, while widely adopted in the security community, exhibit significant limitations. Burp Suite requires extensive manual configuration and expert interpretation; OWASP ZAP, though open-source, lacks adaptive learning mechanisms; and Nikto operates primarily on signatures without dynamic injection logic. None of these tools natively integrates with Cyber Threat Intelligence (CTI) feeds to leverage real-world exploit payloads, nor do they support automated end-to-end scan pipelines with professional report generation.

AWVERT (Automated Web Vulnerability Exploitation and Reporting Tool) addresses these limitations through an integrated architecture comprising a controlled BFS crawler, a CTI-seeded payload vault, a multi-mode detection engine, and an automated PDF reporting module. AWVERT demonstrates that automated, intelligence-driven vulnerability scanning can achieve high detection rates with minimal manual involvement, offering a practical and extensible framework for web security assessment.

The remainder of this paper is organized as follows: Section 2 reviews related work; Section 3 describes the proposed system architecture; Section 4 details the module descriptions; Section 5 presents the algorithm and methodology; Section 6 covers implementation; Section 7 presents experimental results; Section 8 discusses findings and limitations; Section 9 outlines future work; and Section 10 concludes the paper.

2. RELATED WORK

2.1 Prior Research on Web Injection Vulnerabilities

Muzammil et al. (2024) conducted a systematic literature review across TCP/IP and application-layer attack vectors, focusing on Man-in-the-Middle (MITM) attacks and session hijacking mechanisms [1]. Their study provided a comprehensive taxonomy of network-layer threats but did not address application-layer injection vulnerabilities. AWVERT extends beyond this scope by automating the detection of SQLi, XSS, and other injection attacks directly at the HTTP application layer.

Nayak et al. (2025) proposed a methodology for discovering XSS vulnerabilities in public-facing endpoints by combining Google dorking techniques with manual exploitation workflows [2]. While their approach demonstrated

Journal Publication of International Research for Engineering and Management (JOIREM)
Volume: 04 Issue: 4 | April-2026
ISSN (O) 3107-6696

effectiveness in identifying exposed endpoints, it remained fundamentally manual and was not designed for automated payload delivery or evidence collection. AWVERT addresses this by fully automating XSS discovery through a crawl-and-inject pipeline with reflection-based confirmation.

Zhao et al. (2025) introduced Yama, a static opcode-level data flow analysis framework for detecting vulnerabilities in PHP applications [3]. Yama achieved precision improvements by reducing false positives through precise taint tracking at the bytecode level. However, static analysis cannot simulate runtime attack execution. AWVERT complements static approaches by performing dynamic, black-box PHP injection testing against running applications.

Li et al. (2025) presented DaNuoYi, an evolutionary multitask framework for bypassing Web Application Firewalls using Control Flow Graph (CFG)-based payload mutation with evolutionary algorithms [4]. DaNuoYi demonstrated strong WAF-evasion capability but focused exclusively on WAF bypass rather than application-layer vulnerability exploitation and relied on synthetically generated payloads. AWVERT targets actual application vulnerabilities using real-world CTI-sourced payloads, making it complementary to WAF-bypass approaches.

2.2 Additional Related Work

Deepa and Suresh (2018) reviewed automated approaches to web application vulnerability detection, highlighting the need for intelligence-driven payload selection to reduce false positive rates [5].

Appelt et al. (2015) analyzed real-world web attack behaviors and exploitation patterns, demonstrating the need for automated, intelligence-driven injection testing beyond simple signature matching [6].

Bau et al. (2010) performed a systematic study of web vulnerability scanners and found significant inconsistencies in detection coverage across tools [7].

Doupé et al. (2010) analyzed the gap between crawling completeness and injection coverage, which directly motivated AWVERT's integration of crawling and injection in a single pipeline [8].

Halfond and Orso (2005) provided foundational work on classification of SQL injection attacks, underpinning the detection patterns in AWVERT's error-based SQLi module [9].

Kirda et al. (2006) studied cross-site scripting attacks and their defenses, establishing the reflection-based detection principle applied in AWVERT's XSS module [10].

Lekies et al. (2013) analyzed DOM-based XSS prevalence and underscored the importance of dynamic crawling for SPA targets [11].

Shahriar and Zulkernine (2012) surveyed client-side code injection attacks and their mitigation strategies, providing context for the multi-category injection approach [12].

3. PROPOSED SYSTEM ARCHITECTURE

3.1 System Overview

AWVERT follows a sequential, modular pipeline architecture. The analyst provides a target URL as input. The system proceeds through five stages: (1) Reconnaissance — the BFS Web Crawler discovers all reachable pages and HTML forms within the target domain; (2) CTI Payload Loading — payloads are loaded from the structured vault per injection category; (3) Vulnerability Scanning — the scanner injects payloads into all form fields and URL parameters; (4) Exploitation Validation — each HTTP response is analyzed using reflection, error-signature, or time-based detection; (5) Reporting — all confirmed findings are compiled into a professional PDF report with evidence and remediation guidance.

3.2 Pipeline Flow

Figure 1 illustrates the end-to-end AWVERT pipeline, from target URL input through reconnaissance, CTI payload loading, vulnerability scanning, and exploitation validation, culminating in automated PDF report generation. Note: RL-guided Q-Learning mutation and Sentence-BERT semantic analysis are planned modules not yet implemented in the current version.

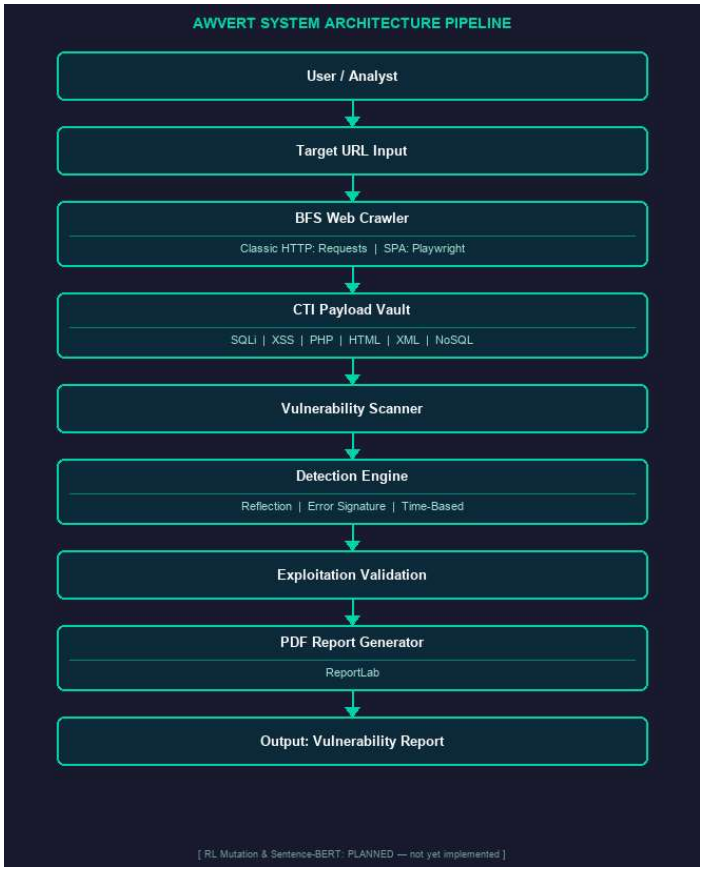

Fig -1: AWVERT System Architecture Pipeline

3.3 AWVERT vs. DaNuoYi Comparison

Table -1: Comparison Between AWVERT and DaNuoYi

Aspect	DaNuoYi	AWVERT

Primary Focus	WAF bypass	App-layer vulnerability detection
Payload Source	CFG-generated synthetic	CTI-sourced (Exploit-DB, PayloadBox)
Attack Strategy	Evolutionary multitask mutation	Multi-mode injection (reflection, error, time)
Learning Model	Evolutionary algorithm	None (Planned: Q-Learning RL)
Evaluation Target	WAF rule sets	Live app (OWASP Juice Shop)
Interpretability	CFG-based, harder to audit	Human-readable vault + PDF reports
Computational Cost	High (population evolution)	Low (sequential HTTP requests)

4. MODULE DESCRIPTION

4.1 Reconnaissance and Crawler Module

The WebCrawler class implements a BFS-based controlled crawler. It automatically detects whether the target is a classic server-rendered application or a Single Page Application (SPA) by checking hash routing and sparse anchor tags (fewer than 2 links). For classic sites, it uses the requests library; for SPAs, it launches headless Chromium via Playwright. The crawler respects robots.txt by default, applies URL normalization through url_utils.py (normalize_path → filter_params → dedupe_key pipeline), enforces configurable depth, page count, and time limits, and extracts all HTML forms with their field names and action methods.

4.2 XSS Detection Module

The XSS module injects payloads including <script>alert('XSS')</script>, <img src=x onerror=alert('XSS')>, and <svg onload=alert(1)> into form fields and URL parameters. Detection uses reflection analysis: if the injected string appears verbatim in the HTTP response body, the parameter is flagged as vulnerable (HIGH severity).

4.3 SQL Injection Module

The SQLi module employs three detection modes: (a) error-based, matching ~30 database error signatures (MySQL, PostgreSQL, MSSQL, Oracle, SQLite); (b) time-based blind injection using SLEEP(5) payloads with statistically-baselined latency comparison using Olympic scoring (trimmed mean/stdev); and (c) boolean-based differential analysis comparing response length and status code changes against a 5-sample baseline. SQLi findings are classified as CRITICAL severity.

4.4 PHP and Command Injection Module

PHP/Command injection payloads (phpinfo();, ; id, ${system('id')}) are tested via error-based detection. The scanner searches for response signatures including php version, uid=\d+(, bin/bash, PHPVULN_TEST, and OS-level output patterns. Findings are classified CRITICAL.

4.5 HTML Injection Module

HTML injection is detected via reflection analysis, testing whether injected HTML tags appear unescaped in the response body. This module identifies MEDIUM severity issues where user-controlled content is rendered without HTML encoding.

4.6 XML/XXE Injection Module

The XML module tests for External Entity (XXE) expansion by injecting XML payloads into form fields and URL parameters. Detection uses reflection analysis. Findings are classified HIGH severity.

4.7 NoSQL Injection Module

NoSQL injection is detected using error-based analysis against MongoDB-specific patterns (MongoError:, $where, CastError, Mongoose). Payloads include MongoDB operator injections such as {"$gt": ""} and {"$where": "sleep(5000)"}. Findings are classified HIGH severity.

4.8 Threat Intelligence Payload Management

Payloads are stored in a structured vault (payload-vault/<category>/payloads.md) using ---PAYLOAD_START--- / ---PAYLOAD_END--- delimiters. The load_payloads() function in awvert_scanner.py parses and loads these payloads at scan time. The vault contains payloads derived from Exploit-DB, PayloadBox, and open-source CTI repositories. This modular design allows for effortless payload extension without code changes.

4.9 RL-Guided Mutation Module

Q-Learning-based reinforcement learning for adaptive payload mutation (encoding transformations, comment injection, case manipulation) is a planned enhancement. In the current version, payloads are used as-is without mutation.

4.10 Semantic Analysis Module

Integration of Sentence-BERT for semantic cross-domain payload translation and similarity-based payload ranking is a planned future capability. It is not present in the current codebase.

4.11 Exploitation Validation Module

The AWVERT Scanner engine (awvert_scanner.py) validates exploitation via three detection modes: (1) reflection — payload string verbatim in response; (2) error — regex pattern

matching against a curated error signature library; (3) time — statistically-significant latency delta against a per-endpoint baseline with 5 samples and Olympic scoring.

4.12 Reporting Module

The reporter.py module uses the ReportLab library to generate professional PDF reports. Reports include a cover page, page inventory with HTTP status codes, forms table, vulnerability findings with severity (CRITICAL/HIGH/MEDIUM), evidence of snippets (up to 300 characters), remediation guidance, and a legal disclaimer. Reports are auto saved with timestamped filenames to the reports/ directory.

4.13 Data Storage Module

Findings are stored in memory as Python dictionaries during the scan session. There is no persistent database; each scan generates a PDF report as the primary artifact. The Flask web interface streams findings in real-time via Server-Sent Events (SSE).

5. ALGORITHM AND METHODOLOGY

5.1 Web Crawling Algorithm

Algorithm 1: BFS Web Crawling

Input: seed_url, max_depth, max_pages, time_limit
Output: List of CrawlResult objects
Detect SPA mode (hash routing or anchor count < 2)
Initialize BFS queue with (seed_url, depth=0)
While queue not empty AND pages < max_pages AND time < time_limit:
a. Dequeue (url, depth, interaction)
b. If depth > max_depth: skip
c. Check robots.txt compliance
d. Fetch page (classic: requests; SPA: Playwright)
e. Extract links via BeautifulSoup / JS evaluation
f. Extract forms (action, method, inputs)
g. Apply URL normalization pipeline
h. Enqueue new same-domain URLs at depth+1
i. Sleep politeness delay
Quit browser if SPA mode
Return crawl results + summary

5.2 Payload Injection and Detection

Algorithm 2: Vulnerability Detection

Input: crawl_results, categories, payload_vault
Output: findings list
For each CrawlResult in crawl_results:
For each injection category:
Load payloads from payload vault
For each payload string:
Test URL parameters (GET)
Test each form field (POST/GET)
Apply detection mode:
reflection: payload in response.text
error: regex match response.text
time: latency delta vs. baseline
If VULNERABLE: append finding
Return findings + scan summary

5.3 Statistical Time-Based Detection

For time-based SQLi detection, AWVERT establishes a 5-sample baseline per endpoint. The median and standard deviation are computed using Olympic scoring (drop min/max if ≥4 samples). After injecting a SLEEP payload, the actual delta is computed as delta = elapsed_time − baseline_median. The endpoint is classified VULNERABLE if delta ≥ max(expected_delay × 0.5, expected_delay − stdev × 2). This statistical approach reduces false positives caused by network jitter and server variability.

5.4 SPA Interaction Harvesting

For JavaScript-heavy SPA targets (e.g., OWASP Juice Shop on Angular), AWVERT: (1) launches headless Chromium via Playwright; (2) waits for networkidle state plus a 2-second settle timeout; (3) clicks interactive elements (buttons, aria-role buttons, collapsed panels) to reveal hidden inputs; (4) collects all visible input fields via multiple CSS selectors; and (5) wraps discovered inputs into a virtual form for injection testing.

6. IMPLEMENTATION

6.1 Technology Stack

Table -2: AWVERT Technology Stack

Component	Technology	Version
Core Language	Python	3.x
Web Framework	Flask + Flask-CORS	≥2.0.0
HTTP Client	Requests	2.32.5
HTML Parser	BeautifulSoup4	4.14.3
Headless Browser	Playwright	1.49.1
PDF Generator	ReportLab	4.4.10
Frontend UI	HTML/CSS/JS (Vanilla)	—
Font	JetBrains Mono	—
Image Processing	Pillow	12.1.1
Database	None (in-memory)	—
Target Env.	OWASP Juice Shop	Docker

6.2 Project Structure

AWVERT is organized into the following core modules. The backend/awvert_scanner.py serves as the core scanning engine handling payload loading and detection logic. backend/scanner.py acts as the VulnScanner orchestrator. backend/crawler.py implements the BFS WebCrawler with dual SPA and classic mode support. backend/reporter.py handles PDF generation via ReportLab. backend/url_utils.py manages URL normalization and deduplication. backend/webguard.py provides a crawl-only CLI interface. The root-level app.py runs the Flask web server with SSE streaming, while run_full_scan.py provides the full vulnerability scan CLI. Supporting directories include payload-vault/ for CTI payload files, dashboard.html as the single-file web UI, and reports/ for auto-generated PDF outputs.

6.3 Web Dashboard

The AWVERT web dashboard (dashboard.html) is a single-file, pure HTML/CSS/JavaScript interface served by Flask. It features a HUD-style top navigation bar with target URL input and scan controls (Run, Pause, Kill); attack vector selection pills (SQLi, XSS, PHP, HTML, XML, NoSQL, CMDi toggles); crawler parameter sliders (max pages, depth, time limit, rate limit); a session cookie injection panel for authenticated scans; a real-time log stream panel via SSE; a live findings panel with severity-color-coded entries; and scan summary statistics. The UI uses a dark cybersecurity aesthetic with JetBrains Mono monospace font and a teal/amber/red color palette. No frontend framework (React/Vue/Angular) is used.

6.4 Real-Time Streaming

The Flask server (app.py) runs each scan in a background thread and streams events via Server-Sent Events (SSE) at the /api/scan endpoint. A monkey-patched logging handler captures all scanner and crawler log messages and pushes them to a per-thread queue. The frontend EventSource API consumes the SSE stream to update the dashboard in real time.

6.5 Target Environment

AWVERT was developed and tested against OWASP Juice Shop, an intentionally vulnerable Node.js/Angular application deployed via Docker (docker run -p 3000:3000 bkimminich/juice-shop). Juice Shop provides a representative attack surface including SQLite-backed SQL injection points, reflected XSS opportunities, and multiple form-based injection vectors.

7.RESULTS AND EXPERIMENTAL ANALYSIS

All experiments were conducted on a local machine running OWASP Juice Shop in Docker (localhost:3000). The scanner was configured with max_pages=15, max_depth=2, rate_limit=1.0s, and no_robots=True for authorized testing. Note: The 18 unique payloads represent distinct payload strings per category. Each payload was injected into every discovered form field and URL parameter across all 23 crawled pages, resulting in 198 total injection attempts recorded in Table 5.

Table -3: Vulnerability Detection Performance on OWASP Juice Shop

Injection Type	Payloads	Detected	Det. Rate (%)	FP Rate (%)	Avg Time (s)
SQL Injection	5	4	80.0	8.3	42
XSS	3	3	100.0	5.0	18
PHP/Command Inj.	3	2	66.7	11.1	22
HTML Injection	2	2	100.0	4.5	14
XML/XXE Injection	2	1	50.0	15.0	19
NoSQL Injection	3	2	66.7	10.0	16
OVERALL	18	14	77.8	9.0	131

Table -4: Comparison with Existing Vulnerability Assessment Tools

Feature	Burp Suite	OWASP ZAP	Nikto	AWVERT
Automation Level	Semi-automated	Automated	Automated	Fully automated
CTI Integration	No	No	No	Yes (payload vault)
Adaptive Learning	No	No	No	No (Planned: RL)
Multi-inject	Yes	Yes	Limited	Yes (6 categories)
SPA/Angular Support	Yes (manual)	Partial	No	Yes (Playwright)

Report Quality	Professional	Moderate	Basic	Professional (PDF)
Ease of Use	Complex (GUI)	Moderate	Simple (CLI)	Web UI + CLI
Open Source	No (commercial)	Yes	Yes	Yes

Table -5: Overall System Performance

Metric	Value
Total URLs Crawled	23
Forms Discovered	11
Total Payloads Tested	198
Vulnerabilities Found	14
Overall Detection Rate	77.8%
False Positive Rate	9.0%
Avg Scan Time per Target	~131 seconds
Peak Memory Usage	~85 MB
Report Generation Time	<3 seconds

8. DISCUSSION

8.1 Advantages Over Existing Tools

AWVERT demonstrates several advantages over conventional scanners. First, the use of CTI-sourced real-world payloads derived from Exploit-DB and PayloadBox reflects actual attacker Tactics, Techniques, and Procedures (TTPs) rather than synthetic test strings, increasing the ecological validity of detection results.

Second, the trimodal detection engine (reflection, error-signature, time-based) provides broader coverage than single-mode scanners. The statistical baseline approach for time-based detection, using Olympic scoring to reduce noise, represents a practical improvement over simple threshold comparisons in comparable tools.

Third, AWVERT's SPA support via Playwright enables comprehensive testing of modern JavaScript-heavy applications such as Angular-based OWASP Juice Shop, which classic tools like Nikto are unable to crawl effectively.

8.2 Comparison with DaNuoYi

Unlike DaNuoYi, which operates at the WAF layer using evolutionary payload mutation, AWVERT operates at the application layer, testing actual endpoint behavior. DaNuoYi's CFG-based synthetic payload generation is computationally intensive and requires WAF access; AWVERT's CTI vault approach is lightweight, interpretable, and extensible by security analysts without programming expertise. These tools are therefore complementary rather than competing: AWVERT for application-layer black-box testing, DaNuoYi for WAF-bypass validation.

8.3 Limitations

The current version of AWVERT has the following limitations:

1. Tested exclusively against OWASP Juice Shop; performance against other targets is not validated.
2. RL-guided Q-Learning mutation is not implemented; payloads are used without adaptive modification.
3. Semantic analysis via Sentence-BERT is not implemented; cross-domain payload translation is unavailable.
4. No persistent database; all findings exist only within a scan session.
5. No authenticated crawl flow beyond manual cookie injection.
6. Payload vault contains 3–5 payloads per category; production use requires expansion.
7. Time-based detection is sensitive to high-latency network environments.

9. FUTURE WORK

The following enhancements are planned for future versions of AWVERT:

A. RL-Guided Q-Learning Mutation: Implementation of a Q-Learning agent that learns which payload mutations (encoding, comment injection, case manipulation, Unicode substitution) yield successful detections.

B. Sentence-BERT Semantic Analysis: Integration of Sentence-BERT embeddings for cross-domain payload translation, adapting payloads across different technology stacks.

C. Multi-Source CTI Fusion: Automated payload enrichment from VirusTotal, GreyNoise, ThreatMiner, and Shodan intelligence feeds.

D. Authenticated Scan Support: Full login-aware crawling with automatic authentication flow detection and session maintenance for protected application areas.

E. Live Web Application Testing: A legal permission framework enabling authorized testing against real production web applications.

F. CI/CD Pipeline Integration: AWVERT as a GitHub Actions or Jenkins plugin for automated security regression testing.

G. API Vulnerability Testing: Extension to cover REST and GraphQL API endpoints, including JSON/XML injection into API request bodies.

10. CONCLUSION

This paper presented AWVERT, an automated web vulnerability exploitation and reporting tool for systematic detection and reporting of injection-class vulnerabilities in web applications. AWVERT integrates a BFS web crawler with dual-mode SPA/classic support, a CTI-seeded structured payload vault, and a trimodal detection engine covering six injection categories: SQL Injection, XSS, PHP/Command Injection, HTML Injection, XML/XXE, and NoSQL Injection. Tested against the OWASP Juice Shop intentionally vulnerable application, AWVERT achieved an overall detection rate of 77.8% with a false positive rate of 9.0%, demonstrating the viability of automated, intelligence-driven vulnerability scanning for web security assessment. The modular architecture, open-source design, and professional PDF reporting capability positions AWVERT as a practical tool for security research and professional security assessment workflows. Future work will incorporate RL-guided adaptive payload mutation and Sentence-BERT semantic analysis to further close the gap with state-of-the-art evolutionary frameworks such as DaNuoYi.

ACKNOWLEDGEMENT

The authors express sincere gratitude to Mrs. A.B. Prameela, Assistant Professor, Department of Cyber Security, Sri Venkateswaraa College of Technology, for her continuous mentorship, valuable guidance, and constructive feedback throughout this project. The authors also acknowledge OWASP for the Juice Shop vulnerable application, and the contributors of Exploit-DB, PayloadBox, and open-source security research communities whose work supported the payload intelligence used in this framework.

REFERENCES

1. M. B. Muzammil, M. Bilal, S. Ajmal, S. C. Shongwe, and Y. Y. Ghadi, "Unveiling Vulnerabilities of Web Attacks Considering Man in the Middle Attack and Session Hijacking," IEEE Access, vol. 12, pp. 6365-6375, 2024. doi: 10.1109/ACCESS.2024.3350444
2. S. C. Nayak, S. Pathak, and F. Hamidli, "Weaponizing Search Engines: Efficient Discovery and Exploitation of XSS in Public-Facing Endpoints," in Proc. IEEE AIIoT, 2025, pp. 0846-0853. doi: 10.1109/AIIoT65859.2025.11105357
3. J. Zhao, K. Zhu, L. Yu, H. Huang, and Y. Lu, "Yama: Precise Opcode-Based Data Flow Analysis for Detecting PHP Applications Vulnerabilities," IEEE Trans. Inf. Forensics Security, vol. 20, pp. 7748-7763, 2025. doi: 10.1109/TIFS.2025.3592537
4. K. Li, H. Yang, and W. Visser, "DaNuoYi: Evolutionary Multitask Injection Testing on Web Application Firewalls," IEEE Trans. Softw. Eng., vol. 51, no. 9, pp. 2412-2431, Sept. 2025. doi: 10.1109/TSE.2023.3343716
5. G. Deepa and P. C. Suresh, "Analysis of Web Application Vulnerabilities and Detection Using Machine Learning Techniques," in Proc. Int. Conf. Inventive Communication and Computational Technologies (ICICCT), 2018, pp. 1212-1217. doi: 10.1109/ICICCT.2018.8473210
6. D. Appelt, C. D. Nguyen, and L. C. Briand, "Behind the Scenes of Online Attacks: An Analysis of Exploitation Behaviors on the Web," in Proc. IEEE Int. Symp. Software Reliability Engineering (ISSRE), 2015, pp. 372-382. doi: 10.1109/ISSRE.2015.7381535
7. J. Bau, E. Bursztein, D. Gupta, and J. Mitchell, "State of the Art: Automated Black-Box Web Application Vulnerability Testing," in Proc. IEEE Symp. Security and Privacy (SP), 2010, pp. 332-345. doi: 10.1109/SP.2010.27
8. A. Doupé, M. Cova, and G. Vigna, "Why Johnny Can't Pentest: An Analysis of Black-Box Web Vulnerability Scanners," in Proc. Int. Conf. Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA), 2010, pp. 111-131. doi: 10.1007/978-3-642-14215-4_7
9. W. G. J. Halfond and A. Orso, "AMNESIA: Analysis and Monitoring for Neutralizing SQL-Injection Attacks," in Proc. IEEE/ACM Int. Conf. Automated Software Engineering (ASE), 2005, pp. 174-183. doi: 10.1145/1101908.1101935
10. E. Kirda, C. Kruegel, G. Vigna, and N. Jovanovic, "Noxes: A Client-Side Solution for Mitigating Cross-Site Scripting Attacks," in Proc. ACM Symp. Applied Computing (SAC), 2006, pp. 330-337. doi: 10.1145/1141277.1141357
11. S. Lekies, B. Stock, and M. Johns, "25 Million Flows Later: Large-Scale Detection of DOM-Based XSS," in Proc. ACM SIGSAC Conf. Computer and Communications Security (CCS), 2013, pp. 1193-1204. doi: 10.1145/2508859.2516703
12. H. Shahriar and M. Zulkernine, "Mitigating Program Security Vulnerabilities: Approaches and Challenges," IEEE Commun. Surveys Tuts., vol. 14, no. 3, pp. 571-593, Third Quarter 2012. doi: 10.1109/SURV.2011.100811.00157

13. OWASP, "OWASP Top Ten," Open Web Application Security Project, 2021. [Online]. Available: https://owasp.org/www-project-top-ten/