IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.151070

AutoSecAudit: An Extensible Multi-Scanner Orchestration Framework with AI-Assisted Remediation Guidance, Precision Fuzzing, and Delta Auditing in DevSecOps

Kodi Vishruth Aithal[1], Thilak N[2], Vinaya Kumar B[3], Vishal Gowda B[4], Ranjitha R*[5]

Department of Computer Science and Engineering, Don Bosco Institute of Technology, Bangalore, Karnataka, India[1-5]

*Corresponding Author

Abstract: Automating security verification in fast-paced continuous integration and continuous deployment (CI/CD) pipelines creates operational friction: studies of developer tool adoption report that noisy output and missing actionable guidance discourage the use of automated analysis tools [13]. This paper presents AutoSecAudit, an open, extensible dynamic application security testing (DAST) orchestration framework with remediation guidance, designed for CI/CD pipelines and AI developer tooling. It combines (1) attack-surface discovery through a depth-limited recursive crawler, JavaScript route extraction, OpenAPI import, and a scope guard that checks every resolved address; (2) concurrent execution of 19 scanner plugins with heuristic false-positive reduction (baseline-response comparison, a two-phase statistical timing check, single-page-application filtering, and adaptive politeness with retry on HTTP 429/503); (3) an intelligence layer that merges duplicate findings on a canonical key, enriches CVE-tagged findings with CVSS data, maps findings to OWASP Top 10 (2021), CWE, and PCI DSS v4.0 by keyword rules, and assigns plain-English business-risk categories and a five-level posture grade; and (4) delivery features: predefined multi-framework fix recipes, fingerprint-based delta comparison between scans, severity gating through exit codes, and a Model Context Protocol (MCP) server with four tools for AI coding assistants. Applying patches is left to the assistant or developer, and closed-loop remediation is not evaluated. We evaluate the tool on two purpose-built test applications with 17 and 8 planted vulnerabilities, using 5 repetitions per setting on one Windows machine. Testbed 2 was fully detected with no false positive in all 5 runs, whereas testbed 1 was fully detected in 3 of 5 runs (mean recall 81%) because some scans reached the 300 s scan deadline. None of these measured runs reported a false positive, but one further testbed 2 run inside the test suite did (a missing-SameSite cookie finding that matched no planted item). Merging reduced raw plugin alerts by 21% and 22% on complete scans. Increasing worker threads from 1 to 16 did not shorten scans (median 53 s at k=1 and 53 s at k=16), so an earlier speedup claim is withdrawn, and 10 of 25 scans ran longer than the 56 s floor. Delta comparison was exact on two complete scans but produced spurious changes when scans were incomplete. These results come from one machine and small applications written alongside the tool; they say nothing about performance on production applications.

Keywords: Vulnerability Assessment, DevSecOps, Scanner Orchestration, Model Context Protocol (MCP), AI-Assisted Remediation, Statistical Timing Verification, Compliance Mapping, Delta Auditing, Web Security, API Security.

I. INTRODUCTION

Modern web applications, cloud-native services, and microservice architectures represent heterogeneous software systems that undergo continuous integration and automated deployment multiple times daily [9]. Ensuring reliable software security within these accelerated delivery cadences motivates 'shift-left' verification, wherein security analysis is executed during automated build and pull-request gating steps [9]. Dynamic Application Security Testing (DAST) evaluates target web services from an external attacker perspective without requiring underlying source code, validating whether identified vulnerabilities are exploitable over active transport protocols.

Embedding DAST in automated build pipelines can be difficult in practice. Johnson et al. [13] report that false positives and poor integration deter developers from using static analysis tools, and Bau et al. [10] found that commercial black-box scanners differ widely in coverage and accuracy. We treat similar friction as a plausible risk for DAST in CI/CD pipelines; this paper does not itself measure developer behavior. We identify five operational challenges that motivate the design:

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.151070

1) Heterogeneous Tool Fragmentation and Format Sprawl: A thorough security audit requires executing multiple specialized utilities—such as port discovery tools (Nmap [1]), web server misconfiguration checkers (Nikto [2]), and API parameter fuzzers. Each utility outputs disparate formats (XML, unstructured console text, non-uniform JSON), forcing engineering teams to write and maintain fragile parsing wrappers.

2) Duplicate Alert Sprawl and Developer Fatigue: Independent scanner modules frequently flag the exact same root vulnerability on an endpoint under slightly varied diagnostic titles. Without cross-plugin deduplication, engineers may triage several notifications for a single bug, which contributes to alert fatigue (cf. [13] for static analysis tools).

3) Executive Communication Gap: Scanner output is typically expressed as Common Vulnerabilities and Exposures (CVE) identifiers, Common Vulnerability Scoring System (CVSS) vectors, and raw request traces. These technical representations can be hard for executives and product owners to translate into business impact.

4) Missing Code Fixes and AI Workflow Disconnection: Scanners often report what is broken but leave developers to write framework-specific patches, input sanitization routines, or Web Application Firewall (WAF) virtual patching rules themselves. Furthermore, developers increasingly work inside AI-assisted environments (Cursor, Claude Code, Antigravity). Most established scanners were designed around CLI, GUI, or REST interfaces rather than an agent protocol, although Model Context Protocol (MCP) [14] integrations are now emerging for some tools [17], [18].

5) Stateless Scans Lacking Regression Awareness: Release gating benefits from knowing what changed between builds. A single scan does not by itself distinguish newly introduced vulnerabilities from previously known ones or show whether a committed patch resolved an open issue; findings-management platforms [19] and scan-comparison features address this in different ways.

To address these challenges, we present AutoSecAudit, an extensible multi-scanner security orchestration framework with remediation guidance for AI-assisted workflows. AutoSecAudit combines automated attack surface discovery, concurrent execution of 19 scanner plugins, false-positive reduction mechanisms, plain-English business danger translation, predefined multi-framework fix recipes, fingerprint-based delta comparison between scans, and an MCP tool server in one DevSecOps engine. The contributions are the integrated design and its open implementation; the evaluation in Section IV is small-scale, uses two built-in test applications, and is reproducible from the repository.

II. RELATED WORK AND COMPARATIVE ANALYSIS

Security testing tools generally fit into four architectural paradigms: standalone point scanners, interactive DAST proxies, template-driven matchers, and centralized vulnerability management platforms. Table I summarizes built-in capabilities as described in each tool's public documentation (accessed October 2026) and compares them with AutoSecAudit. Capabilities available only through add-ons, integrations, or configuration are noted as such, and the table is not a head-to-head benchmark.

A. *Point Scanners and Specialized Utilities*

Network scanners like Nmap [1] handle port discovery and service fingerprinting, while web server checkers like Nikto [2] scan for server misconfigurations and outdated defaults. Template-driven scanners like Nuclei [3] run YAML-defined checks over HTTP and other protocols. These tools are focused scanners: each runs as a standalone binary with its own output format, and none of them by itself correlates results across tools, compares scans across builds, or translates findings into business risk. Nuclei templates do carry severity and classification metadata, and all three tools can emit machine-readable output that external pipelines consume.

B. *Stateful DAST Proxies and API Fuzzers*

State-aware scanners, explored by Doupé et al. [11], track application navigation states to uncover deeper injection points. In API fuzzing, Atlidakis et al. [12] demonstrated with RESTler that effective testing requires parsing OpenAPI structures. Intercepting proxies and scanners such as OWASP ZAP [4] and PortSwigger Burp Suite provide broad coverage and are considerably more mature than AutoSecAudit. ZAP ships packaged baseline, API, and full scans and an Automation Framework intended for CI use [16]; full active scans with proxy-based tools can be long-running depending on configuration and target, which is why time-boxed or passive modes are common in merge checks. We did not benchmark AutoSecAudit against these tools, so this paper makes no claim of superior coverage or accuracy.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.151070

C. *Vulnerability Management Aggregation Platforms*

Platforms like OWASP DefectDojo [5] aggregate findings from many third-party scanners, deduplicate them, and track their lifecycle, including re-import workflows that close findings absent from a newer report [19]. DefectDojo does not itself execute scans, so teams still need pipeline jobs that run the individual tools and upload their reports. AutoSecAudit instead bundles execution with a lighter-weight post-processing layer; it does not replace a findings-management platform.

D. *AI Coding Assistants and Agentic Protocols*

Recent advances in AI-assisted software engineering have produced agentic developer tools (such as Cursor, Claude Code, and Antigravity). The Model Context Protocol (MCP) [14] standardizes how such agents call external tools over JSON-RPC 2.0, and MCP integrations now exist for other security tools, including a Burp Suite extension [17] and a ZAP MCP integration add-on [18]. AutoSecAudit implements an MCP stdio server that lets an agent initiate scans, inspect findings with business-risk annotations, and retrieve predefined fix recipes. The protocol is not a contribution of this work. The server does not edit source code itself, and the agent-driven loop of patching, re-scanning, and checking for regressions is not evaluated in this paper.

TABLE I COMPARISON OF BUILT-IN CAPABILITIES ACROSS SECURITY TESTING PARADIGMS

Dimension / Capability	Nmap / Nikto / Nuclei [1-3]	OWASP ZAP [4]	DefectDojo [5]	AutoSecAudit (Ours)
Primary role	Focused scanners (ports, web server, templates)	Intercepting proxy and scanner	Findings-management platform; does not run scans	**Plugin-based DAST orchestrator**
Parallelism	Tool-specific	Configurable scanner threads	Not applicable	**Plugin-level ThreadPool (default 4, configurable)**
Surface discovery	Tool-specific; limited crawling	Spider; OpenAPI import via add-on [16]	Not applicable (imports reports)	**Recursive HTML crawl, regex JS routes; OpenAPI import feeds the plugins (see Sec. III-A)**
Scanner breadth	Single purpose or template-defined	Built-in rules plus add-ons	None built in; imports third-party reports	**19 built-in plugins (compact)**
False-positive handling	Tool-specific matchers	Rule thresholds and alert filters	Deduplication and triage statuses	**Baseline comparison; 2-phase timing check; SPA filter**
Vulnerability intelligence	Template metadata (Nuclei)	Alert references (CWE, OWASP)	Stores imported CVE/CWE/CVSS data	**NVD/CIRCL CVSS lookup for CVE-tagged findings**
Compliance mapping	Template tags (Nuclei)	Alert references	Imported metadata and reports	**Keyword rules to OWASP 2021, CWE, PCI DSS v4.0**
Business-risk view	Not provided	Risk levels only	Dashboards and metrics	**Plain-English danger categories, 5-level grade (heuristic)**
Fix guidance	Tool-specific advice	Per-alert solution text	Imported guidance; Jira integration	**Predefined recipes (Node, Python, Nginx, WAF) and Markdown tickets**
AI-agent interface	Not assessed	MCP integration add-on [18]	REST API	**MCP stdio server, 4 tools**
Cross-build comparison	Not built in	Not assessed	Re-import closes and reopens findings [19]	**Fingerprint-based delta (Sec. III-C)**
CI/CD gating	Exit status or output parsed by pipeline scripts	Automation Framework exit-status job [16]	Through API and integrations	**--fail-on severity threshold, exit code 1**

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.151070

III. SYSTEM ARCHITECTURE AND METHODOLOGY

AutoSecAudit is structured into four decoupled subsystems (Fig. 1): (1) Input Ingestion, Surface Discovery & Pre-flight Validation; (2) Concurrent Multi-Scanner Engine & Precision Subsystem; (3) Post-Processing Intelligence Layer; and (4) Delivery, CI/CD Gating, and AI-Assistant Integration.

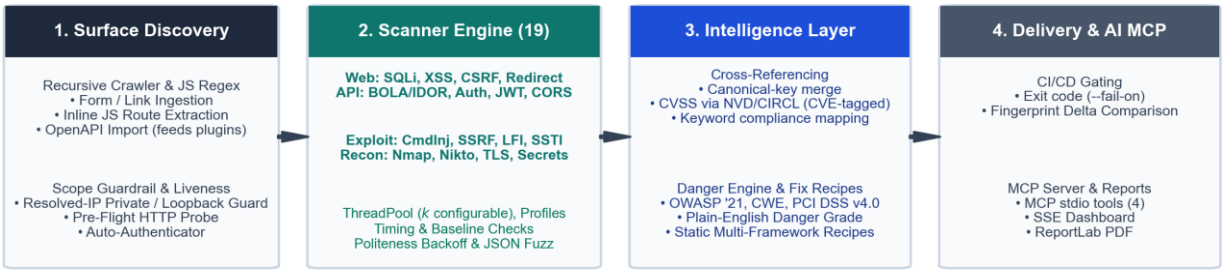


Fig. 1. AutoSecAudit system architecture: four-stage pipeline from surface discovery and concurrent 19-plugin execution through cross-referencing, compliance mapping, danger translation, delta comparison, and MCP integration.

A. *Attack Surface Discovery and Pre-flight Validation*

Given a target hostname, IP address, or URL, the surface discovery subsystem seeds an endpoint pool E before initiating active scans. The design is motivated by [11], [12] but does not implement state-aware crawling or RESTler-style stateful fuzzing:

1) Target Scope Guardrail and Pre-Flight Liveness: Before a scan starts, is_target_allowed() parses the target host. An IP literal is accepted only if it is private (RFC 1918 / RFC 4193) or loopback; a hostname is resolved through DNS and accepted only if every address it resolves to is private or loopback (IPv4 and IPv6). The check is enforced when the engine sets its target, so the CLI, the dashboard, and the MCP server share it, and a name that merely looks private (for example 10.example.com) is rejected when it resolves to a public address. This is a guardrail against accidental scanning of public systems, not an authorization mechanism: it does not verify that the operator may test the target, and a DNS answer could change between the check and the scan. HTTP redirects are followed manually and only to same-origin, in-scope locations, and each response is bounded in size (5 MiB by default) and in time (30 s per request by default). In real-scan mode, an HTTP reachability probe (HEAD, falling back to GET, with a raw TCP fallback) confirms that the server responds before plugins are scheduled; the probe is skipped in mock mode.

2) Recursive Web Crawler and Client-Side JS Route Extraction: The WebCrawler follows same-host links starting from the root URL, recursing depth-first (not breadth-first) up to a depth limit D_max (default 3) and a page budget (default 20). It extracts HTML form structures (<form action method>), input fields, query parameters, and hyperlinks. Regular-expression extractors are applied to the HTML of each visited page, including inline <script> content, to find client-side REST API routes (/api/v1/...) that lack standard HTML links. Externally referenced JavaScript bundles are not fetched or analyzed in the current implementation, so routes that exist only in such bundles are missed.

3) OpenAPI / Swagger Ingestion: For headless REST backends, the OpenAPIImporter reads an OpenAPI 2.0 or 3.x specification supplied as a JSON or YAML file or URL and extracts paths, HTTP methods (GET, POST, PUT, DELETE, PATCH), declared parameters (including path variables), local $ref targets, and top-level JSON request-body properties. The imported endpoints are handed to the scanner plugins, and when no specification is supplied the engine also looks for one at common locations (for example /swagger.json, /v2/api-docs, /v3/api-docs, and /api-docs) and imports it automatically. Remote $ref targets and full request-body schema validation are not handled.

4) Automated Multi-Strategy Authenticator: To audit authenticated application surfaces, the AutoAuthenticator probes common login paths, tries HTML-form, JSON-API, and blind-form login strategies with user-supplied credentials, extracts anti-CSRF tokens, and captures session cookies and bearer tokens. The engine then passes the captured headers and cookies to the crawler and, through set_auth(), to the plugins; we did not audit how much each plugin uses the session. Authentication runs only in real-scan mode.

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.151070

Algorithm 1: Attack Surface Discovery (as implemented: depth-first, bounded)

Input: Target URL u0, Max Depth D_max, Page Budget P_max, Auth Credentials A (optional)
Output: Discovered Endpoint Pool E (forms, links, parameters, JS routes)
1: If not IsTargetAllowed(u0) [every resolved address]: Abort('Target not permitted')
2: If real mode and not CheckLiveness(u0): Abort('Host unreachable')
3: Session S = InitializeSession()
4: If A is provided: S = AutoAuthenticate(u0, A)
5: Visited Set V = {}, E = {}
6: Crawl(u0, 0)
7: Procedure Crawl(u, d):
8: If d > D_max or |V| >= P_max or Normalize(u) in V: Return
9: V = V U {Normalize(u)}; Response R = S.GET(u)
10: If R is not HTML: E = E U {ResourceEndpoint(u)}; Return
11: E = E U ParseHTMLForms(R.body) U ExtractJSRoutes(R.body)
12: For each same-host link l in ExtractHyperlinks(R.body):
13: E = E U {LinkEndpoint(l)} (with query parameters, if any)
14: Crawl(l, d+1)
15: Return E

Note: endpoints imported from an OpenAPI document (supplied or auto-discovered) are added to E before the plugins run.

B. *Core Engine, 19 Scanner Plugins, and Precision Mechanisms*

Each scanner module extends the abstract BaseScanner interface. Standardized findings use the tuple model: F = <id, title, severity, host, port, description, raw_output, c, M, t, e, m, q>, where c in {low, medium, high} is a heuristic confidence label assigned by the plugin, M holds optional CVE/CVSS, OWASP, CWE, PCI DSS, and remediation fields, and t, e, m, and q are the vulnerability type, normalized endpoint path, HTTP method, and parameter that identify the finding across tools and across scans.

AutoSecAudit organizes 19 scanner plugins across four operational domains:

- Web Application Injections: SQLiScanner (error-based and tautology payloads, plus time-based blind probes verified by the timing check), XSSScanner (reflected XSS only), CSRFScanner (anti-CSRF tokens and SameSite cookie attributes), OpenRedirectScanner (unvalidated redirect parameters), MisconfigScanner (security headers, cookie flags, exposed files, HTTP methods).
- API & Access Control: BOLAIdorScanner (object-identifier manipulation), APIAbuseScanner (excessive data exposure, mass assignment, missing pagination, GraphQL introspection, API versioning, broken authentication), AuthScanner (default credentials, IDOR, weak-password registration, exposed admin panels, user enumeration, login rate limiting), JWTScanner (passive inspection of JWTs observed in responses for alg=none headers and missing or expired exp claims; it does not test weak HMAC secrets or forged-token acceptance), CORSScanner (wildcard origin, origin reflection, null origin).
- Advanced Exploit Vectors: CommandInjectionScanner (OS command injection, including timing-verified blind probes), SSRFScanner (internal-URL probes; cloud-metadata probes to 169.254.169.254 are disabled unless AUTOSEC_ENABLE_METADATA_PROBES is set), PathTraversalScanner (local file inclusion, e.g., /etc/passwd), SSTIScanner (arithmetic expression probes such as {{7*7}}, ${7*7}, and <%= 7*7 %>).
- Reconnaissance & Infrastructure: SecretExposureScanner (sensitive paths such as /.env and /.git/config), SSLTLSScanner (certificate validity and expiry, HSTS, obsolete TLS versions), DirBruteScanner (a 50-entry curated path list that includes several Juice Shop-specific entries), NiktoPlugin and NmapPlugin (wrappers around the external nikto and nmap executables, which must be installed).

Scan Profiles: The engine supports four scan profiles, defined in config.py and verified by loading each profile: 'full' (all 19 plugins), 'owasp' (11 plugins: SQLi, XSS, CSRF, SSRF, Auth, Misconfig, CommandInjection, PathTraversal, SSTI, SecretExposure, OpenRedirect), 'api' (7 plugins: BOLAIdor, JWT, CORS, APIAbuse, Auth, SQLi, XSS), and 'recon' (6 plugins: SSLTLS, Misconfig, DirBrute, Nikto, Nmap, SecretExposure). The recon profile is described in the configuration as reconnaissance with low-impact active probing: Nmap performs port scanning, and Nikto, DirBrute, Misconfig, and SecretExposure issue active HTTP requests, so it is non-exploitative but not passive. Plugins run concurrently through a thread pool whose size is set by AUTOSEC_THREAD_COUNT (default 4); the evaluation in Sec. IV sweeps it from 1 to 16.

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.151070

Precision Mechanisms: AutoSecAudit includes four mechanisms intended to reduce false positives and target load. They lower, but do not eliminate, false-positive risk:

1) Two-Phase Statistical Timing Check (core/timing.py): Blind time-based probes (for example SLEEP(3) or sleep are prone to false positives from latency spikes. BaseScanner measures m = 2 benign requests to estimate a baseline mean mu_base and sample standard deviation sigma_base. For a probe with injected delay tau, the delay must reach mu_base + max(3*sigma_base, 0.70*tau) on a first probe and again on a second, independent delay probe. If a zero-delay control payload is supplied, phase 2 requires it to return within mu_base + 0.35*tau, and the finding is then graded high confidence; without a control probe a reproduced delay is graded medium, and a slow control probe rejects the finding. With m = 2 the variance estimate is coarse and the check remains a heuristic. Timing probes are serialized across plugins so that concurrent scanning does not contaminate the measurements, and the check is applied by the SQLi and command-injection plugins only.

2) Baseline Response Comparison: The SQLi plugin requests a benign baseline response and compares each probe response against it; a change in HTTP status or a response-length change above 20% is treated as evidence, graded high, medium, or low by the magnitude of the change. Other plugins use their own matching logic.

3) SPA and Development-Server Filtering (core/spa_detector.py, core/dev_detector.py): Responses that look like single-page-application catch-all pages are filtered to avoid findings caused by soft-404 behavior, and development-server artifacts are tagged.

4) Adaptive Politeness (core/politeness.py): Each plugin adds randomized 0-20 ms jitter between requests. All plugins share one HTTP client; on HTTP 429 or 503 it honors Retry-After (capped at 10 s) or backs off exponentially for min(2^(n-1), 8) seconds and retries up to twice, and a response that is still throttled is withheld from detection logic so that a rate-limit page is not reported as evidence. The client also enforces the scope, size, and time limits described in Sec. III-A.

A recursive JSON body fuzzer (core/fuzzer.py) enumerates every leaf of a nested JSON request body and replaces one leaf at a time with a payload while preserving the rest of the structure; the SQLi and command-injection plugins use it. Object keys that contain a period are not addressed correctly by its dotted-path scheme, and empty containers have no leaf to mutate.

C. *Post-Processing Intelligence Layer*

Following raw scanner execution, the shared pipeline (core/pipeline.py, used by the CLI, the dashboard, and the MCP server) post-processes findings in the steps of Algorithm 2. Findings from different plugins that describe the same weakness are merged on a canonical key <vulnerability type, HTTP method, normalized endpoint, parameter>, in which numeric path segments are replaced by a placeholder and the reporting tool is ignored; the merged finding keeps the highest severity and confidence, the union of the reporting tools, and the combined evidence and references. Remaining findings that share an endpoint and parameter are cross-referenced but not merged. Compliance mapping applies keyword rules to finding titles to assign OWASP Top 10 (2021) categories [6] and uses a built-in table for CWE identifiers and PCI DSS v4.0 requirements [7]; CVSS data are looked up from the NVD [8] or CIRCL for findings that carry a CVE identifier.

Algorithm 2: Intelligence Post-Processing and Delta Comparison (as implemented)

Input: Raw Findings F_raw, optional Previous Report S_{t-1}
Output: Annotated Report S_t, Delta Partition <Delta_NEW, Delta_FIXED, Delta_UNCHANGED>
1: Group F_raw by canonical key k(f) = <f.type, f.method, norm(f.endpoint), f.param>
2: For each group g: S_t = S_t U {Merge(g)} [max severity, max confidence, union of tools and evidence]
3: For merged findings with equal <endpoint, param>: append 'Related findings: ids' to f.references
4: For each f in S_t: fp(f) = SHA-256(f.tool | f.type | f.method | f.endpoint | f.param)
5: For each f in S_t with f.cve_id: query NVD (2 s timeout), fall back to CIRCL;
6: set f.cvss_score if unset and merge references; skip silently on failure
7: For each f in S_t: (f.owasp_tag, f.cwe_id, f.pci_dss) = MapCompliance(f) [keyword rules]
8: Danger category/grade and fix recipes are derived from f when a report or MCP response is built
9: If S_{t-1} is provided: T = scanners that completed successfully in both scans;
10: keep only findings reported by a scanner in T
11: Delta_NEW = { f in S_t | fp(f) not in fp(S_{t-1}) }
12: Delta_FIXED = { f in S_{t-1} | fp(f) not in fp(S_t) }
13: Delta_UNCHANGED = { f in S_t | fp(f) in fp(S_{t-1}) }
14: Return S_t, <Delta_NEW, Delta_FIXED, Delta_UNCHANGED>

 This work is licensed under a Creative Commons Attribution 4.0 International License



International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026
DOI: 10.17148/IJARCCE.2026.151070

Plain-English Danger Engine and Executive Posture: The DangerEngine assigns each finding to one of four business threat categories using keyword rules over the finding title and tool name, with an OWASP-tag fallback: (i) Customer & Database Theft (SQL injection, exposed repositories and environment files, directory listing); (ii) Account Hijacking & Auth Bypass (authentication, JWT, session, cookie, IDOR, and brute-force findings); (iii) Phishing, XSS & Brand Defacement (CORS, XSS, security headers, open redirects); and (iv) Full Server Takeover & RCE (command injection and remote code execution). Unmatched findings default to category (i). The posture score is 100 - (25*C + 12*H + 5*M + 1*L), where C, H, M, and L are the counts of critical, high, medium, and low findings, clamped to the range [10, 100]; together with count thresholds it yields one of five grades (A+, B, C, D, F). The grade and the plain-English explanations (what is broken, what an attacker can do, business impact, estimated fix time) are rule-based heuristics; their usefulness to non-technical readers was not evaluated.

Fix Recipes and Dev Tickets: The RecipeEngine returns predefined, static code snippets selected by keywords in the finding title: Node.js/Express (parameterized queries, Helmet, express-validator), Python (SQLAlchemy and Django ORM, Pydantic, Flask/FastAPI middleware, Flask-Talisman), Nginx directives, and a generic WAF rule in JSON form. Recipes exist for seven finding classes (SQL injection, CORS, security headers, directory and secret exposure, CSRF, open redirect, BOLA/IDOR) plus a generic fallback; there are none for XSS, command injection, SSRF, SSTI, JWT, or path traversal. The recipes are templates: they are not generated per codebase, and this paper does not test that applying them removes the corresponding vulnerability. A Markdown ticket template is also provided that can be pasted into issue trackers such as Jira, Linear, or GitHub.

Fingerprint-Based Delta Comparison: The DeltaAnalyzer compares two reports by a content fingerprint: the SHA-256 of the reporting tool, vulnerability type, HTTP method, normalized endpoint, and parameter (Algorithm 2, step 4), so the key does not depend on the order in which a plugin numbers its findings. With fp(S) denoting the set of fingerprints in report S, findings are partitioned into three disjoint sets: Delta_NEW = { f in S_t | fp(f) not in fp(S_{t-1}) }, Delta_FIXED = { f in S_{t-1} | fp(f) not in fp(S_t) }, and Delta_UNCHANGED = { f in S_t | fp(f) in fp(S_{t-1}) }. Only scanners that completed successfully in both scans are compared, so a narrower profile does not report every finding of the missing scanners as fixed. Because the reporting tool is part of the key, a finding that is reported by a different set of plugins in two scans changes fingerprint; Sec. IV-C measures how often this happens.

D. *AI-Assistant Integration via Model Context Protocol (MCP) and CI/CD Delivery*

AutoSecAudit includes a JSON-RPC 2.0 stdio MCP server (mcp_server.py; protocol revision 2024-11-05) that lets AI developer tools such as Cursor, Claude Code, and Antigravity call the scanner [14]. It implements the initialize, tools/list, tools/call, and ping methods and exposes four tools (Table II). We distinguish four levels of remediation support: (a) generating predefined fix recipes, which is implemented; (b) exposing those recipes to an agent through MCP, which is implemented; (c) an agent applying a change to source code, which is performed by the agent's own editing tools and is outside the server; and (d) re-scanning to show that a change fixed the issue, which autosec_verify_fix does against the live target by re-running only the scanners relevant to the vulnerability type; it does not check for regressions elsewhere. Levels (a), (b), and (d) are implemented; none of them is evaluated against agent-applied patches in this paper.

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.151070

TABLE II AUTOSECAUDIT MCP TOOLS AND THEIR CURRENT LIMITATIONS

MCP Tool Name	Arguments	Behavior as implemented	Limitation in this release
autosec_scan	target (required), profile, mock	Runs a scan and returns grade, severity counts, and the threat matrix.	mock defaults to true, so a call that omits it returns simulated findings; real scans must request mock=false.
autosec_get_findings	report_id, view	Reads a saved scan report and returns findings with business-danger text and fix recipes.	Reads the saved report key 'all_findings' (an earlier release read 'findings' and returned no findings; fixed and covered by tests).
autosec_get_fix_recipe	finding_title (required), framework	Returns static recipes (nodejs, python, nginx, waf) and a Markdown ticket for a title keyword.	Keyword lookup; seven finding classes plus a generic fallback.
autosec_verify_fix	target (required), vulnerability_type (required)	Re-runs only the scanners for the vulnerability type against the target and reports RESOLVED or STILL_VULNERABLE; scan errors or an unreachable target are reported as errors, not as RESOLVED.	Real by default (mock=false); checks one vulnerability type only and does not detect regressions elsewhere.

The server therefore supports recipe delivery, scan access, and targeted re-testing for an AI assistant. It does not edit source files, and this paper does not demonstrate that a recipe or an agent-applied patch removes a vulnerability or avoids regressions; no patch-success rate is reported.

For CI pipelines, the command-line scan accepts --fail-on with the values low, medium, high, or critical. If any finding meets or exceeds the threshold, the process returns exit code 1; it also returns 1 when the target is invalid or unreachable, and 0 otherwise. Progress is streamed to the web dashboard over Server-Sent Events (SSE), and reports can be exported as HTML, JSON, and ReportLab-generated PDF files.

IV. EXPERIMENTAL EVALUATION AND RESULTS

Evaluation setup. All measurements were taken on one machine (Windows-11-10.0.26200-SP0, 8 logical CPUs, 15.4 GB RAM, Python 3.14.6) that was also running ordinary desktop processes, so it was not an isolated or idle host. The targets are the two built-in testbeds, started on 127.0.0.1 with an ephemeral port for each scan. Testbed 1 is a Flask application with 17 planted, detectable vulnerabilities (SQL injection, reflected XSS, SSTI, command injection, SSRF, path traversal, open redirect, IDOR, CSRF, insecure cookie, exposed .env file, and an unauthenticated admin page). Testbed 2 requires a login and has 8 planted vulnerabilities plus two decoys that look vulnerable but are safe. Each scan used the full profile with the Nmap and Nikto wrappers excluded, so 17 of the 19 plugins ran. The ground truth for each testbed is a JSON file in benchmark/ground_truth; a finding is a true positive when its vulnerability type and normalized endpoint match a planted item, and a finding that matches a decoy or nothing is a false positive, except for informational findings such as missing security headers. The authors wrote the testbeds, the ground truth, and the plugins, so the accuracy results measure how well known cases are handled, not how the tool generalizes. Earlier versions of this paper reported results on OWASP Juice Shop; those figures had no retained raw data and relied on a consolidation step that the released code did not implement, so they are withdrawn and replaced by the measurements below. The raw per-run results are stored in benchmark/results/paper and can be regenerated with python -m benchmark.paper_experiments all. The evaluation addresses four research questions: RQ1 (Latency & Concurrency), RQ2 (Accuracy & Alert Consolidation), RQ3 (Delta Comparison Behavior), and RQ4 (Test Suite and Safety Checks).

Evidence status. Each setting was run 5 times. We report means, standard deviations, medians, and ranges; with this few runs no confidence interval or significance test is claimed, and the runs were not randomized beyond rotating the thread count within each repetition.

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.151070

A. *Concurrency and Execution Latency (RQ1)*

Total scan time of testbed 1 was measured for worker thread counts k in {1, 2, 4, 8, 16}, 5 times each (Table III, Fig. 2). The median time was 53.3 s at k=1 and 53.2 s at k=16, and the fastest scan at every k was 53.0 s to 53.2 s, so on this target more threads did not make the scan faster. Our earlier report of a 2.84x speedup is therefore not supported and is withdrawn. A plausible reason is that the scan contains a fixed amount of serial work: timing probes are deliberately serialized across plugins (Sec. III-B), and the testbed contains endpoints that sleep on purpose. We did not profile the scan, so this explanation is untested. The timing was also unstable: 10 of 25 scans took longer than 56 s, 2 reached the 300 s scan deadline, and 4 returned fewer than the usual 27 findings. These slow scans were spread across thread counts, and we did not isolate the cause (candidates are contention on the desktop, the serialized timing probes, and the plugin time limits). The CPU column reports process CPU time, which includes the in-process test server and is not a utilization percentage.

TABLE III SCAN TIME OF TESTBED 1 BY WORKER THREADS (N = 5 SCANS PER ROW)

Worker Threads (k)	Mean +/- SD (s)	Median (s)	Min-Max (s)	Speedup (median)	Process CPU (s)
1 (sequential)	103.0 +/- 110.3	53.3	53.1-300.3	1.00x	16.9
2	66.6 +/- 18.5	53.2	53.1-87.8	1.00x	21.1
4	136.6 +/- 108.7	81.2	53.2-300.3	0.66x	19.8
8	77.6 +/- 35.9	57.4	53.0-136.3	0.93x	18.9
16	66.5 +/- 28.9	53.2	53.1-118.2	1.00x	18.5

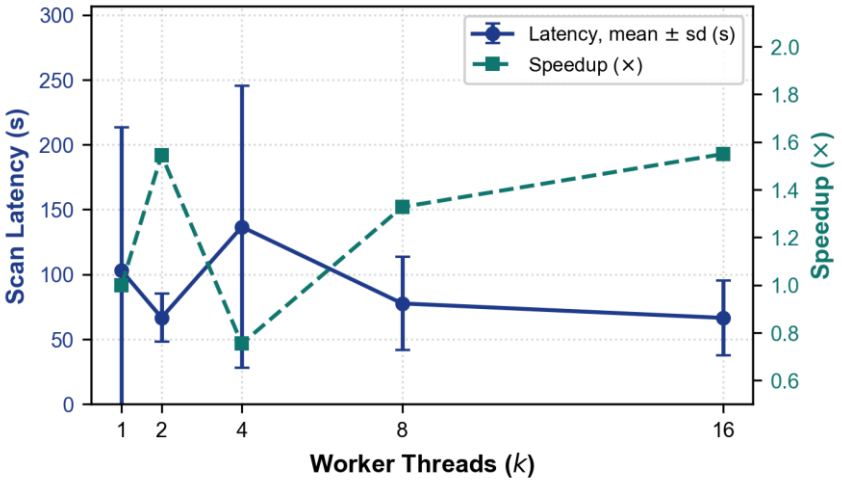


Fig. 2. Scan time (mean and standard deviation of 5 scans) and speedup of the mean relative to k=1, by worker thread count.

B. *Detection Accuracy and Alert Merging (RQ2)*

A true positive (TP) is a planted vulnerability matched by at least one finding, a false positive (FP) is a finding that matches a decoy or nothing in the ground truth, and a false negative (FN) is a planted vulnerability with no matching finding. Precision = TP/(TP+FP), recall = TP/(TP+FN), and F1 = 2PR/(P+R). Table IV gives the results per testbed. Testbed 2 was fully detected in all 5 runs with no false positive, and both decoys were suppressed. Testbed 1 was fully detected in 3 of 5 runs; the other 2 runs found 8, 10 of 17 because plugins hit the 300 s scan deadline, which the engine reports as a degraded scan. None of the ten measured runs reported a false positive. However, an eleventh run of testbed 2, executed by the benchmark regression test during the test-suite run (Sec. IV-D), reported one: a missing-SameSite finding for the session cookie, labelled as CSRF at the root path, which matched no planted item and failed that test's zero-false-positive assertion. So the false-positive count is not stable at zero even here, and with applications whose planted items and decoys we chose ourselves it should not be read as a false-positive rate for real applications.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.151070

TABLE IV DETECTION ACCURACY ON THE BUILT-IN TESTBEDS (* COMPLETE SCANS ONLY)

Testbed	Runs	Planted	TP mean (range)	FP	Precision	Recall (mean)	Raw alerts to merged findings*
testbed1	5	17	13.8 (8-17)	0	1.00	0.81	34 to 27 (21% fewer)
testbed2 (login, 2 decoys)	5	8	8.0 (8-8)	0	1.00	1.00	20 to 16 (22% fewer)

Alert merging. The pipeline merges findings that several plugins report for the same weakness on a canonical key (Sec. III-C). On complete scans, the 34 raw plugin alerts of testbed 1 became 27 findings (21% fewer) and the 20 alerts of testbed 2 became 16 (22% fewer; Fig. 3). Because the merge key was designed with these applications in view, the reduction is specific to them, and we did not measure whether any merge hid a distinct vulnerability beyond checking that all planted items remained matched.

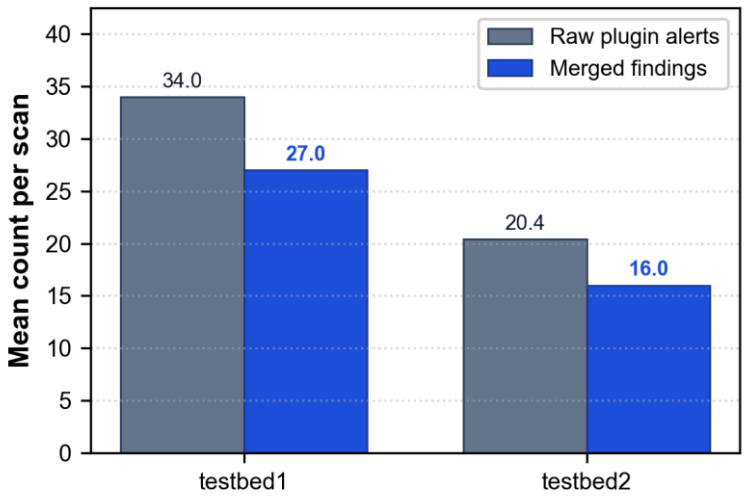


Fig. 3. Mean raw plugin alerts and merged findings per scan on complete scans of each testbed.

C. *Delta Comparison Behavior (RQ3)*

Delta comparison was tested on three consecutive real scans of testbed 1 whose application build was changed on purpose: build v1 hides the /template route, build v2 is the full application, and build v3 hides the /view route. From the ground truth, the expected result of v1 to v2 is 2 new findings (SSTI and XSS at /template) and none fixed, and of v2 to v3 is 2 fixed findings (path traversal and XSS at /view) and none new. The experiment was executed twice. In the second execution all three scans were complete (25, 27, and 25 findings), and the delta was exact: v1 to v2 gave 2 new, 0 fixed, and 25 unchanged findings, and v2 to v3 gave 0 new, 2 fixed, and 25 unchanged (Fig. 4). In the first execution the scans were incomplete (they returned fewer findings than a complete scan; the engine's degraded flag was not logged) and returned 22, 19, and 25 findings; the delta then reported 4 new and 3 fixed findings for v1 to v2 and 3 new and 4 fixed for v2 to v3, which includes findings that had not changed (the admin page and the exposed .env file each appeared as both new and fixed, and the IDOR finding was absent from one scan). In the complete scans the fingerprints of identical findings were identical. The first execution did not log fingerprints, so the cause of the paired new/fixed entries is untested; it is consistent with a change in the set of plugins that reported the finding, because the reporting tool is part of the fingerprint (Sec. III-C). A delta is therefore only as reliable as the scans it compares, and a release gate should refuse to compare scans that the engine marks as degraded or that return fewer findings than expected. This check covers one application and two build changes and is not a general fidelity figure.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.151070

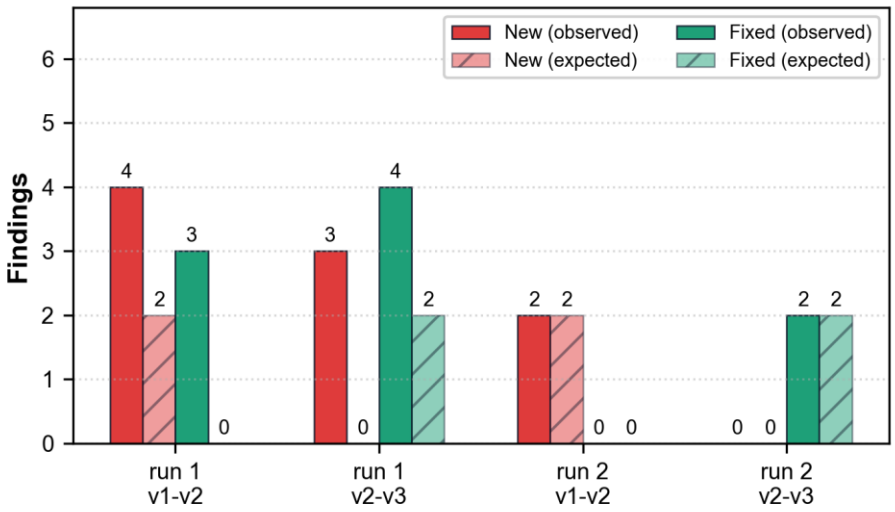


Fig. 4. Delta results per build transition in two executions, compared with the expected counts (hatched).

D. *Test Suite and Safety Checks (RQ4)*

The project's runner (run_tests.py) executed 578 tests in our verification run (parallel worker processes, with timing-sensitive modules re-run alone); 577 passed and 1 failed. The failure is the benchmark regression test test_benchmark_builtin_testbed1_and_testbed2, which asserts zero false positives and failed because testbed 2 reported the cookie finding described in Sec. IV-B; it failed again when re-run alone, so the suite is not fully green. The tests are isolated from the persistent user database by a per-worker temporary data directory, which fixes the non-idempotent authentication test noted in an earlier version. They cover plugin behavior, the timing engine, the JSON fuzzer, the danger engine and recipes, MCP message handling, scan profiles, target reachability, report escaping, and the --fail-on exit code, and several run real scans against local fixture servers. Regression tests also check safety properties: logout and other state-changing paths are never probed (test_probe_safety.py), redirects to out-of-scope hosts are not followed (test_b09_shared_http_client.py), the dashboard rejects POST requests without a CSRF token, throttles repeated failed logins, sets session cookie attributes, requires a POST with a CSRF token to log out, and hides one user's reports from another (test_b05_web_hardening.py and test_b20_dashboard_ownership_and_session.py), and the dashboard binds to 127.0.0.1 by default. These are regression checks written by the developers, not an independent security assessment, and they do not show that no vulnerability remains in the tool itself.

V. DISCUSSION, LIMITATIONS, AND DEPLOYMENT CONSIDERATIONS

Deployment considerations. (1) Single-page applications: regular-expression extraction finds API routes embedded in the HTML of visited pages, but externally loaded JavaScript bundles are not analyzed; pairing the crawler with a headless browser (for example Playwright) would be needed for full DOM-driven discovery. (2) Offline operation: CVSS enrichment applies only to findings that carry a CVE identifier, uses 2 s network timeouts, and is skipped silently when NVD and CIRCL are unreachable; there is no embedded offline CVE dictionary. The CWE and PCI DSS tables used by the compliance mapper are built in. (3) Enterprise authentication: SAML 2.0 and mTLS flows are not supported by the automatic authenticator; pre-authenticated headers can be supplied with the --headers option. (4) Default mode: the configuration default is mock mode (AUTOSEC_MOCK_MODE=true), and the MCP autosec_scan tool also defaults to mock, so real scans must be requested explicitly. (5) Transport: scanner requests disable TLS certificate verification. (6) Scan completeness: a scan that reaches its deadline is marked degraded, and such a scan should not be used for release gating or delta comparison (Sec. IV-A and IV-C).

Limitations and threats to validity. The evaluation uses two small applications that the authors wrote together with the tool and its ground truth, one machine that was not isolated, and five repetitions per setting, so the accuracy figures describe known cases and not real-world performance, and no statistical inference is made. The parallel speedup reported in earlier versions is not reproduced: scan time did not fall with more threads, the cause was not profiled, and scans were intermittently slow or truncated by the 300 s deadline. Only 17 of the 19 plugins ran in the evaluation because the Nmap and Nikto wrappers need external binaries. The merge key and the delta fingerprint were designed with these testbeds in view, delta behavior was checked on one application and two build changes, and the scope guard resolves names once, so a changing DNS answer is not detected. The JWT plugin inspects tokens passively, and the fix recipes cover seven finding classes and were not applied to a vulnerable application and re-scanned. No head-to-head comparison with ZAP,

DOI: 10.17148/IJARCCE.2026.151070

Burp Suite, Nuclei, or other tools was performed, and the fix recipes and danger grades were not validated with developers or executives.

Recommended follow-up work. (a) Profile a scan to find the serial work that sets the 53 s floor and the cause of the intermittent slow and truncated scans, then repeat the thread sweep on an idle machine. (b) Evaluate on independent intentionally vulnerable applications such as VAmPI, PyGoat, Vulpy, and a pinned OWASP Juice Shop image [15], using the ground-truth files and harness in the benchmark directory, with two independent annotators. (c) Make delta comparison refuse, or flag, scans that the engine marks as degraded. (d) Run an agent-in-the-loop study that applies recipes, re-scans, and checks for regressions. (e) Compare against ZAP baseline and API scans on the same targets. (f) Add recipes for the finding classes that have none.

VI. CONCLUSION AND FUTURE WORK

AutoSecAudit is an open, extensible DAST orchestration framework that runs 19 scanner plugins concurrently through one shared HTTP client, applies heuristic false-positive reduction, merges duplicate findings, annotates them with compliance and business-risk context, offers predefined fix recipes, and exposes scanning, recipe lookup, and targeted re-testing to AI coding assistants through an MCP server. On two small purpose-built applications it found all planted vulnerabilities when its scans completed (3 of 5 runs on testbed 1 and all 5 on testbed 2), and merging reduced raw alerts by 21% and 22%; false positives were absent in ten measured runs but one appeared in a further run. The evaluation also exposed weaknesses: more worker threads did not shorten scans, some scans were slow or reached the scan deadline and lost findings, and delta comparison gave spurious changes when scans were incomplete. These results are small-scale, come from applications the authors wrote, and replace earlier figures that could not be reproduced. Closed-loop AI remediation, in which an agent applies a patch and a live re-scan confirms the fix without regressions, remains future work. Further work will address the limitations in Sec. V, evaluate independent vulnerable applications, and explore distributed multi-node scanning.

ACKNOWLEDGMENT

This research was conducted as part of the AutoSec Security Engineering Initiative in the Department of Computer Science and Engineering at Don Bosco Institute of Technology, Bangalore. The authors express sincere gratitude to faculty mentor Ms. Ranjitha R, department leadership, and the open-source security engineering community for foundational tools, benchmark platforms, and ongoing support.

REFERENCES

[1] G. F. Lyon, Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Vulnerability Scanning. Insecure.Com LLC, 2009.
[2] C. Sullo, "Nikto web server scanner," open-source software, version 2.6. Available: https://github.com/sullo/nikto (accessed Oct. 2026)
[3] ProjectDiscovery, "Nuclei: Fast and Customizable Vulnerability Scanner Based on Simple YAML DSL," 2024. Available: https://github.com/projectdiscovery/nuclei
[4] OWASP Foundation, "OWASP Zed Attack Proxy (ZAP)," 2024. Available: https://www.zaproxy.org/
[5] OWASP Foundation, "OWASP DefectDojo: Vulnerability Management Orchestration," 2024. Available: https://www.defectdojo.org/
[6] OWASP Top 10 Team, "OWASP Top 10: 2021 The Ten Most Critical Web Application Security Risks," OWASP Foundation, 2021. Available: https://owasp.org/Top10/
[7] PCI Security Standards Council, Payment Card Industry Data Security Standard (PCI DSS): Requirements and Testing Procedures, Version 4.0, 2022. Available: https://www.pcisecuritystandards.org/document_library/ (accessed Oct. 2026)
[8] National Institute of Standards and Technology (NIST), "National Vulnerability Database (NVD) REST API v2.0," U.S. Department of Commerce, 2023. Available: https://nvd.nist.gov/developers/vulnerabilities (accessed Oct. 2026)
[9] M. Myrbakken and R. Colomo-Palacios, "DevSecOps: A Multivocal Literature Review," in Communications in Computer and Information Science, vol. 770, Springer, 2017, pp. 17–29.
[10] J. Bau, E. Bursztein, D. Gupta, and J. C. Mitchell, "State of the Art: Automated Black-Box Web Application Vulnerability Testing," in Proc. 31st IEEE Symposium on Security and Privacy (S&P '10), 2010, pp. 332–345.
[11] A. Doupé, M. Cova, and G. Vigna, "Enemy of the State: A State-Aware Black-Box Web Vulnerability Scanner," in Proc. 21st USENIX Security Symposium (USENIX Security '12), 2012, pp. 523–538.



International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026
DOI: 10.17148/IJARCCE.2026.151070

[12] V. Atlidakis, P. Godefroid, and M. Polishchuk, "RESTler: Stateful REST API Fuzzing," in Proc. 41st IEEE/ACM International Conference on Software Engineering (ICSE '19), 2019, pp. 748–758.
[13] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, "Why Don't Software Developers Use Static Analysis Tools to Find Bugs?" in Proc. 35th IEEE/ACM International Conference on Software Engineering (ICSE '13), 2013, pp. 672–681.
[14] Anthropic, "Model Context Protocol (MCP) Specification," 2024. Available: https://modelcontextprotocol.io
[15] OWASP Foundation, "OWASP Juice Shop," project page. Available: https://owasp.org/www-project-juice-shop/
[16] ZAP Project, "Automation Framework," ZAP documentation. Available: https://www.zaproxy.org/docs/automate/automation-framework/
[17] PortSwigger, "Burp Suite MCP Server extension," GitHub repository. Available: https://github.com/PortSwigger/mcp-server
[18] ZAP Project, "MCP Integration add-on," ZAP documentation. Available: https://www.zaproxy.org/docs/desktop/addons/mcp-integration/
[19] DefectDojo, "About Deduplication" and "Reimport," DefectDojo documentation. Available: https://docs.defectdojo.com/

BIOGRAPHY

The authors are undergraduate research scholars and faculty mentors in the Department of Computer Science and Engineering at Don Bosco Institute of Technology (DBIT), Bangalore, affiliated with Visvesvaraya Technological University (VTU), Belagavi, Karnataka, India. Their research focuses on automated vulnerability assessment, DevSecOps engineering, intelligent security orchestration, and agentic AI remediation systems.