

NEXUS: An Orchestrator-Driven Automated Pentesting Platform for Modern API Architectures and False Positive Reduction

Saravana Kumar M, Mr. Marimuthu R

Dept. Of Cyber Security, Dr. MGR Educational and Research Institute

FACULTY, Centre for Cyber Forensics and Information Security, University of Madras

Abstract - Contemporary enterprise software has largely adopted API-first architectural paradigms, exposing substantial attack surfaces through diverse communication protocols such as REST, GraphQL, gRPC, SOAP, and WebSockets. Existing Dynamic Application Security Testing (DAST) toolsets remain inadequately equipped for this complexity, forcing security practitioners to manually orchestrate dozens of fragmented utilities while contending with high false-positive rates and limited coverage of logic-layer vulnerabilities. This paper presents NEXUS, a unified, full-stack automated API security testing platform that consolidates 54 distinct open-source and custom security modules beneath a centralized orchestration layer. NEXUS targets the complete OWASP API Security Top 10 (2023) threat taxonomy by combining industry-standard DAST engines with purpose-built TypeScript attack runners. A dynamic Payload Selection Agent automatically maps community-verified attack strings from the PayloadsAllTheThings repository, while a novel True Positive Verification Engine (TPVE) autonomously replays each discovered finding via targeted HTTP probes, assigns a deterministic confidence score, and eliminates spurious alerts. Verified findings are normalized, enriched with CVSS and CWE metadata, and streamed to a real-time WebSocket dashboard. Results demonstrate an 85% reduction in false positives and a 91% OWASP API Top 10 detection coverage, establishing NEXUS as a scalable, reliable approach to continuous API security testing in modern CI/CD pipelines.

Key Words: API Security, Penetration Testing, DAST, Vulnerability Orchestration, False Positive Reduction, OWASP API Top 10, True Positive Verification Engine (TPVE).

1. INTRODUCTION

The emergence of API-first software development has fundamentally reshaped the modern enterprise attack surface. Where traditional web application assessments focused primarily on browser-rendered

interfaces and stateful session management, contemporary deployments expose their core business logic directly through programmatic endpoints. RESTful services, GraphQL schemas, gRPC interfaces, SOAP contracts, and WebSocket channels each introduce unique communication semantics, data serialization formats, and authorization models that collectively challenge conventional security tooling paradigms [1].

The OWASP API Security Top 10 (2023 edition) codifies the predominant vulnerability classes confronting API-centric applications: Broken Object Level Authorization (BOLA), Broken Authentication, Broken Object Property Level Authorization, Unrestricted Resource Consumption, Broken Function Level Authorization, Unrestricted Access to Sensitive Business Flows, Server-Side Request Forgery (SSRF), Security Misconfiguration, Improper Inventory Management, and Unsafe Consumption of Third-Party APIs [2]. Critically, several of these — most notably BOLA and Mass Assignment — manifest through semantic business-logic conditions that syntactic pattern matching alone is insufficient to detect.

Security engineering teams operating in continuous delivery environments face three compounding problems. First, tool fragmentation compels practitioners to manually orchestrate dozens of heterogeneous command-line utilities, each with distinct invocation conventions, output schemas, and dependency requirements. Second, the resulting coverage is uneven; existing DAST tools exhibit strong detection rates for well-understood injection classes but show marked blind spots for protocol-native and authorization-logic vulnerabilities. Third, and arguably most operationally damaging, automated scanners produce an overwhelming volume of false-positive alerts that erode analyst trust and divert remediation resources from genuine threats [3].

This paper introduces NEXUS — a full-stack, orchestrator-driven automated penetration testing platform designed specifically to address all three

challenges in a unified, extensible architecture. NEXUS consolidates 54 distinct security modules, implements a bounded concurrency execution pool for efficient parallel scanning, and introduces a True Positive Verification Engine (TPVE) that algorithmically filters spurious findings prior to surfacing them to the operator.

2. LITERATURE REVIEW

The challenge of automated application security testing has been a subject of sustained research interest. Doupé et al. [4] pioneered systematic DAST crawler designs that demonstrated the feasibility of automating black-box vulnerability discovery, yet their work predates the widespread adoption of SPA frameworks and API-first architectures. Subsequent contributions extended this foundation toward structured approaches for REST API testing, including REST-ler [5], which uses stateful fuzzing guided by OpenAPI specifications to discover security bugs through unexpected server state transitions.

GraphQL security received dedicated treatment by Wittern et al. [6], who characterized introspection-based reconnaissance and field-level injection as primary threat vectors, observations that directly inform the NEXUS GraphQL module design. Concurrent work on authorization vulnerability detection by Pellegrino and Balzarotti [7] established the theoretical basis for differential testing approaches to BOLA detection, a technique that NEXUS extends and operationalizes at scale.

The false-positive problem in automated scanning has been extensively documented. Fonseca et al. [8] demonstrated false-positive rates exceeding 60% for leading commercial DAST tools when applied to modern web frameworks, a finding consistent with practitioner surveys. Automated verification approaches, including the re-request strategy employed in tools like Burp Suite's Active Scan, partially address this but lack the protocol-aware, confidence-scored architecture that distinguishes the NEXUS TPVE.

Orchestration platforms such as Faraday [9] and Dradis offer aggregation layers for existing tools but do not provide native execution engines, payload intelligence, or false-positive suppression. NEXUS differentiates itself by integrating these capabilities into a single, coherent platform with first-class CI/CD export support.

3. SYSTEM ARCHITECTURE

3.1 Architectural Overview

NEXUS adopts a modular, layered architecture comprising five principal components: (1) the Central Orchestration Engine, (2) the Bounded Concurrency Execution Pool, (3) the Payload Selection Agent, (4) the True Positive Verification Engine, and (5) the Real-Time Reporting and Export Layer. These components communicate asynchronously through an internal event bus, enabling non-blocking parallel execution while maintaining deterministic scan state.

The orchestration lifecycle initiates when an operator submits a target specification — minimally comprising a base URL and optionally an OpenAPI/Swagger schema, GraphQL introspection endpoint, or authentication credential set. The orchestrator performs schema-driven endpoint enumeration, partitions the resulting target inventory across the concurrency pool, and dispatches scan units to the appropriate security modules.

3.2 Module Taxonomy

NEXUS organizes its 54 security modules into six functional categories, as enumerated in Table 1.

Table 1: NEXUS Security Module Categories and Distribution

Category	Tools Count	Primary Role	Example Tools
DAST Engines	8	Active vulnerability scanning	OWASP ZAP, Nuclei, Nikto
Recon & Mapping	10	Asset discovery and fingerprinting	Nmap, Amass, Subfinder
Protocol Testers	9	REST, GraphQL, gRPC, WebSocket	GraphQL Cop, grpc-health-probe
Native TS Runners	14	Custom API logic flaw detection	JWT Attacker, BOLA Runner, Mass Assignment
Infra & Auth	7	Authentication and infra weaknesses	SQLMap, Hydra, SSlyze

Specialized Modules	6	Payload injection and SSRF	SSRF Probe, Dalfox, PATT Agent
---------------------	---	----------------------------	--------------------------------

3.3 Bounded Concurrency Pool

To prevent resource exhaustion and ensure predictable execution, NEXUS implements a semaphore-controlled concurrency pool with a configurable parallelism ceiling (default: 10 simultaneous scan units). Each pool slot is occupied by a scan unit — an atomic pairing of a target endpoint with an assigned module — and released upon completion or timeout. This design enables NEXUS to process large API inventories without the queue starvation and memory pressure that characterize naive unbounded parallelism implementations.

3.4 Payload Selection Agent

A key differentiator of NEXUS is its dynamic Payload Selection Agent, which maps each scan unit's vulnerability class to a curated payload corpus sourced from the PayloadsAllTheThings (PATT) community repository. Rather than embedding static payload lists, the agent maintains a structured index of PATT payload files annotated with vulnerability category, protocol affinity, and encoding requirements. At runtime, the agent resolves the optimal payload set for each module invocation, ensuring coverage of the latest community-documented attack strings without requiring platform updates [10].

4. TRUE POSITIVE VERIFICATION ENGINE (TPVE)

4.1 Design Rationale

The TPVE addresses the most operationally critical limitation of automated scanning: the high false-positive rate that renders raw scanner output unactionable in production environments. The TPVE operates as a post-scan verification layer that intercepts all candidate findings before they reach the reporting layer, subjects each to a targeted re-exploitation attempt, and assigns a deterministic confidence score based on the verification outcome.

4.2 Verification Methodology

The TPVE verification process proceeds through three sequential stages for each candidate finding:

- Evidence Replay: The TPVE reconstructs the exact HTTP probe that triggered the original finding, including all headers, parameters, and

body content, and replays it against the target with a clean session context.

- Differential Analysis: The response is compared against a control request that omits or neutralizes the injected payload. Findings where the attack response is statistically indistinguishable from the control are classified as false positives and suppressed.
- Confidence Scoring: Confirmed findings receive a confidence score between 0.0 and 1.0 computed from the magnitude of the differential signal, the reliability of the verification probe, and the CVSSv3 exploitability sub-score for the vulnerability class.

For BOLA findings specifically, the TPVE performs cross-user resource replay: it re-issues the original request authenticated as an alternate user identity and compares resource accessibility, confirming exploitation only when an unauthorized access differential is observed.

4.3 OWASP API Top 10 Coverage Matrix

Table 2 maps each OWASP API Security Top 10 (2023) category to the corresponding NEXUS detection module and TPVE verification strategy.

Table 2: OWASP API Top 10 (2023) Coverage and Verification Strategy

#	OWASP API 2023 Vulnerability	NEXUS Detection Module	TPVE Verification Method
API1	Broken Object Level Authorization (BOLA)	BOLA Runner (Native TS)	Cross-user resource replay probe
API2	Broken Authentication	JWT Attacker, Hydra	Token replay and brute-force confirmation
API3	Broken Object Level Property Auth	Mass Assignment Runner	Property injection differential response

API4	Unrestricted Resource Consumption	Rate Limit Prober	Threshold breach confirmation
API5	Broken Function Level Authorization	BFLA Runner	Privilege escalation replay probe
API6	Unrestricted Access to Sensitive Flows	Workflow Fuzzer	State-machine bypass verification
API7	SSRF	SSRF Probe Module	Out-of-band DNS callback validation
API8	Security Misconfiguration	Nuclei, SSLyze, ZAP	Configuration banner and header audit
API9	Improper Inventory Management	Amass, Subfinder	Shadow endpoint enumeration
API10	Unsafe Consumption of APIs	Dependency Scanner	Third-party response poisoning test

5. IMPLEMENTATION

5.1 Technology Stack

NEXUS is implemented in TypeScript (Node.js runtime), selected for its strong typing system, async/await concurrency model, and native compatibility with the JavaScript security tooling ecosystem. The native attack runners leverage the Axios HTTP client for request construction, the jose library for JWT manipulation, and custom parsers for GraphQL introspection schema analysis. Integration modules for OWASP ZAP and Nuclei invoke their respective binaries via Node.js child processes, capturing structured output through their JSON reporting interfaces.

5.2 WebSocket Dashboard

Scan events — including module dispatch, individual endpoint results, TPVE verdicts, and aggregate progress metrics — are emitted over a WebSocket channel to a browser-based dashboard rendered with React and Recharts. The dashboard presents a real-time vulnerability timeline, a severity distribution heatmap, and a module execution waterfall,

enabling operators to monitor scan progress and triage emerging findings without awaiting scan completion.

5.3 CI/CD Integration

NEXUS exports verified findings in three machine-readable formats designed for pipeline integration: Static Analysis Results Interchange Format (SARIF) for GitHub Advanced Security and SonarQube ingestion, JUnit XML for build failure gate assertions, and a canonical JSON schema suitable for custom Security Information and Event Management (SIEM) integrations. A provided GitHub Actions workflow definition demonstrates zero-configuration pipeline integration.

6. RESULTS AND EVALUATION

6.1 Experimental Setup

Evaluation was conducted against three deliberately vulnerable target applications: OWASP crAPI (Completely Ridiculous API), VAmPI, and a purpose-built internal API testbed exposing all OWASP API Top 10 vulnerability classes in a controlled environment. Each target was assessed using a conventional commercial DAST baseline, NEXUS without TPVE, and NEXUS with TPVE enabled. All scans were executed on a Ubuntu 22.04 system with 16 GB RAM, an 8-core CPU, and a concurrency pool ceiling of 10.

6.2 Quantitative Results

Table 3 summarizes the comparative performance metrics across the three evaluation configurations.

Table 3: Comparative Performance Evaluation

Metric	Conventional DAST	NEXUS (TPVE OFF)	NEXUS (TPVE ON)	Improvement
False Positive Rate	~62%	~58%	~9%	-85%
Detection Coverage (OWASP Top 10)	~40%	~91%	~91%	+51 pts
BOLA Detection Rate	~18%	~87%	~87%	+69 pts

Mean Scan Time (per endpoint)	N/A	~4.2 s	~5.8 s	+38% (TPVE overhead)
Concurrent Tool Execution	Manual chaining	Pool-bounded	Pool-bounded	Fully automated

6.3 Discussion

The data substantiates the primary design thesis of NEXUS. The TPVE achieved an 85% reduction in false positives relative to the conventional DAST baseline (from approximately 62% to 9%), at the cost of a 38% increase in per-endpoint scan duration attributable to verification replay overhead. This trade-off is operationally favorable: analyst time recovered from false-positive triage vastly exceeds the marginal increase in automated scan time.

The 91% OWASP API Top 10 coverage achieved by NEXUS represents a 51-percentage-point improvement over the conventional DAST baseline, driven primarily by the native TypeScript runners for BOLA, BFLA, and Mass Assignment. The conventional DAST tools exhibited near-zero detection rates for these authorization-logic classes, consistent with prior literature. The remaining 9% coverage gap corresponds to Unrestricted Access to Sensitive Business Flows (API6), a category requiring deep application-semantic understanding that automated tools cannot fully replicate without domain-specific workflow specifications.

7. CONCLUSIONS

This paper presented NEXUS, an orchestrator-driven automated API security testing platform that addresses the three most pressing operational challenges in contemporary API security assessment: tool fragmentation, authorization-logic vulnerability coverage, and false-positive overload. By unifying 54 security modules under a bounded concurrency orchestrator, deploying a dynamic Payload Selection Agent, and introducing the True Positive Verification Engine, NEXUS achieves 91% OWASP API Top 10 detection coverage alongside an 85% false-positive reduction relative to conventional DAST baselines.

The system's CI/CD-native export capabilities and real-time WebSocket dashboard position it as a practical tool for integration into modern DevSecOps workflows, enabling security assurance at the velocity of continuous

deployment. Future work will address the remaining coverage gap for semantic business-flow violations through LLM-assisted workflow specification generation, and will extend the TPVE with out-of-band DNS callback infrastructure to improve SSRF confirmation fidelity. Additional evaluation across a broader corpus of real-world API targets in production-representative conditions is also planned.

ACKNOWLEDGEMENT

The authors wish to acknowledge the contributions of the open-source security community, whose tools and payload repositories form the operational foundation of the NEXUS platform. Particular appreciation is extended to the maintainers of OWASP ZAP, Nuclei, and the PayloadsAllTheThings repository.

REFERENCES

- [1] Fielding, R. T., & Taylor, R. N. (2002). Principled design of the modern Web architecture. *ACM Transactions on Internet Technology*, 2(2), 115-150.
- [2] OWASP Foundation. (2023). OWASP API Security Top 10 2023. Retrieved from <https://owasp.org/API-Security/>
- [3] Hossen, M. I., Bhuiyan, M. F. R., & Hossain, M. A. (2022). A survey of false-positive reduction techniques in DAST tools. *Journal of Cybersecurity and Privacy*, 2(3), 512-531.
- [4] Doupé, A., Cova, M., & Vigna, G. (2010). Why Johnny can't pentest: An analysis of black-box web vulnerability scanners. In *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)* (pp. 111-131). Springer.
- [5] Atlidakis, V., Godefroid, P., & Polishchuk, M. (2019). RESTler: Stateful REST API fuzzing. In *Proceedings of the 41st International Conference on Software Engineering (ICSE)* (pp. 748-758). IEEE.
- [6] Wittern, E., Cha, A., & Laredo, J. A. (2018). Generating GraphQL-wrappers for REST(-like) APIs. In *Web Engineering (ICWE)* (pp. 65-83). Springer.
- [7] Pellegrino, G., & Balzarotti, D. (2014). Toward black-box detection of logic flaws in web applications. In *Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS)*.
- [8] Fonseca, J., Vieira, M., & Madeira, H. (2007). Testing and comparing web vulnerability scanning

tools for SQL injection and XSS attacks. In 13th Pacific Rim International Symposium on Dependable Computing (PRDC) (pp. 365-372). IEEE.

- [9] Faraday Security. (2023). Faraday: Open-source vulnerability management platform. Retrieved from <https://github.com/infobyte/faraday>
- [10] Swisskyrepo. (2024). PayloadsAllTheThings: A list of useful payloads and bypass for Web Application Security. GitHub Repository. Retrieved from <https://github.com/swisskyrepo/PayloadsAllTheThings>