

Assessing the Security of “Vibe Coding”: An Analysis of Vulnerabilities in Web Applications Generated by LLMs

Leandro L. Retzlaff¹, Alan Motta¹

¹Departamento de Engenharias e Ciência da Computação
Universidade Regional Integrada do Alto Uruguai e das Missões (URI)
Santo Ângelo – RS – Brasil

leandroretzlaff@gmail.com

mottaschumacher@san.uri.br

Abstract. *Vibe coding shifts validation from source code review to observable behaviour, which may allow vulnerabilities that do not affect perceived functionality to go unnoticed. This paper compares the vulnerability surface of four e-commerce web applications generated by four distinct LLMs from a single specification without additional security requirements. Each application was scanned twice with OWASP ZAP 2.17.0, once per access profile, under identical configuration, and complemented by a manual audit of access control and session management. The scans reported between 32 and 58 unique findings per model and up to nine high-severity findings. Manual reproduction refuted all 18 high-severity findings, the most pronounced difference between models. The defects common to all four applications and not reported by the scanner concern the session lifecycle: tokens remain valid after logout, and signing secrets are embedded in the source code as default values.*

1. Introduction

Vibe coding generates functional applications from natural language descriptions with minimal human supervision over the produced code [Meske et al. 2025], a practice adopted by independent developers, startups and corporate teams [Daniotti et al. 2026]. Unlike incremental code review in AI-assisted development, validation shifts to observable behaviour, so vulnerabilities without effect on perceived functionality may go unnoticed. Decisions regarding secret storage, session management and input validation may thus be made out of convenience, without support in the requirements.

Security evaluation of LLM-generated code focuses on isolated snippets or specific tasks [Ramírez et al. 2024]. This work compares complete applications generated by multiple models from the same specification using Dynamic Application Security Testing (DAST) and a complementary manual audit. The choice of OWASP ZAP is grounded in [Qadir et al. 2025], which evaluated 75 applications with nine tools and identified ZAP as the most effective DAST tool, while reporting categories missed by all tools. Manual auditing is supported by [Elder et al. 2022], which found that techniques reveal exclusive vulnerabilities and manual methods identify fewer but more severe ones. The hypotheses concern DAST findings under identical conditions: **H1**: implementations differ in the number of findings; **H2**: they differ in severity distribution; **H3**: they differ in vulnerability category profiles.

Vargas et al. [Vargas et al. 2026] compare security-oriented Terraform generation using Checkov and Trivy, distinguishing syntactic validity from security compliance.

2. Methodology

This preliminary study evaluated a single run ($N = 1$) per model due to generation costs: DeepSeek V4 Pro 0813, Qwen3.8 27B, Fable 5 and Opus 5, identified in the artefacts by the legacy labels `deepseek-mini-v4`, `qwen-3.8-8b`, `fable5` and `opus`.

A single specification fixed the stack (Vue.js SPA, Express and TypeScript, PostgreSQL and Docker Compose) and ten functional invariants. E-commerce was selected for its authentication and transaction requirements [Hassan et al. 2020]; additional security guidance was deliberately omitted to observe weaknesses left during functional implementation. The specification is in the artefact repository¹. The constant instruction was “implement according to the spec without using any extra tool”. The intended restriction concerned strategy-changing extensions; ordinary editing and build commands were allowed. All runs used command-line agents: Claude Code for Fable 5 and Opus 5, and Verboo for DeepSeek and Qwen. Compilation corrections were autonomous, without human code edits.

Figure 1 summarises the workflow; Table 1 records its parameters.

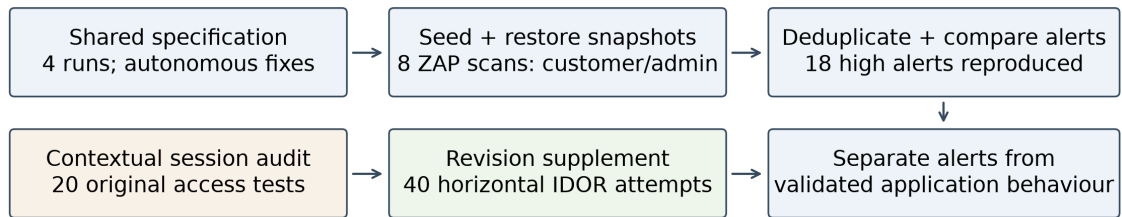


Figure 1. Generation and security evaluation workflow.

Table 1. Experimental settings. Policy pairs denote strength/threshold.

Item	Setting
Comparison	Four model/agent runs; one application per run
Scanner	ZAP 2.17.0; frozen image/add-ons, including alpha/beta rules
Discovery	Traditional spider: 5 min, depth 10; AJAX spider: 5 min
Active scan	Maximum 90 min; 10 min/rule; five threads/host
Policy	Default medium/medium; SQLi high/low; XSS high/medium
Scope/session	Application origin; exclude logout/Stripe; inject login cookie via replacer
Outcomes	Unique alerts by severity/category; alerts per reported path

2.1. Non-intervention

No vulnerability was fixed. Only six lines were changed to enable execution and scanning: a DeepSeek backend build script (+1) and Opus frontend `/api` forwarding (+5). No change affected security middleware, validation, authentication/authorisation, libraries or response headers; all edits are documented in the artefact repository.

¹<https://github.com/LeandroRetz/artigo-dast-vibecoding-dados>

2.2. Scan preparation

Each application was scanned twice, once per profile (`user_comum` and `admin`), totalling eight runs. These are the two authenticated roles in the specification, each exposing a different functional surface. Before each scan, the database was wiped and populated with at least one record of each resource of the specification for each user; between the two passes of the same model, the database was restored from a snapshot.

2.3. ZAP configuration

DAST tests a running application through its exposed interfaces. ZAP combines resource discovery, passive inspection of HTTP traffic and active attack payloads. OWASP ZAP 2.17.0 [ZAP Development Team 2025] used the common configuration in Table 1; cookie injection accommodated the `Secure` attribute in two applications. The complete policy, add-ons, executed plans and reports are recorded in the repository. All high-severity findings were reproduced manually against the running application, with the database restored to the scanned state; the remaining ones are reported as issued by the scanner.

2.4. Processing and manual audit

Findings were deduplicated by normalised path (e.g., `/api/orders/{id}`), parameter and alert type, and classified by OWASP Top 10:2021 [OWASP Foundation 2021] and CWE Top 25:2023 [MITRE Corporation 2023]. Normalisation divides findings by distinct paths with alerts, including SPA fallback paths. Served-code fingerprints confirmed application identity. Contextual checks, following [Hadavi et al. 2021], complemented H1–H3: Insecure Direct Object Reference (IDOR), or unauthorised access through resource identifiers, and session management. The original cross-role tests classified every `2xx` response as a violation. Session checks recorded token storage, cookie attributes, signing algorithm, expiration, logout invalidation and source-code secrets.

3. Results

Compilation corrections required 10 iterations in DeepSeek, 25 in Qwen and 3 each in Fable and Opus. Each agent-reported compilation-blocking error followed by autonomous correction counted as one iteration. Beyond the shared stack, only Opus adopted a schema validation library and UUID identifiers uniformly, and only DeepSeek did not declare the `cors` package as a dependency.

Table 2 consolidates the findings reported per model, aggregating the two profiles. The reported high-severity classes were SQL injection (CWE-89, 9 findings), path traversal and source code disclosure by file inclusion (CWE-22 and CWE-541, 7), server-side template injection (CWE-1336, 1) and NoSQL injection (CWE-943, 1). The administrative profile reached additional findings in all applications: 25 in DeepSeek, 13 in Opus, 5 in Qwen and 4 in Fable, confirming the need for the second pass.

The mapped findings concentrate in A01, A03 and A05; five to six informational alerts per model have no assigned category. The distinction between models concentrates in A03. Excluding the high-severity false positives, DeepSeek and Opus retain 20 and 13 alerts, respectively: persistent XSS in JSON responses (low severity; 12 and 11, absent in the others) and integer overflow (medium severity).

Table 2. Unique scanner findings across both profiles, by severity and OWASP category. None of the 18 high-severity findings was confirmed.

Model	Severity (ZAP)					OWASP				
	High	Med.	Low	Inf.	Tot.	Paths	/path	A01	A03	A05
DeepSeek	8	20	21	9	58	17	3.41	7	26	19
Opus	9	13	21	12	55	20	2.75	14	18	18
Qwen	1	12	9	10	32	14	2.29	9	3	15
Fable	0	12	10	10	32	14	2.29	11	2	14

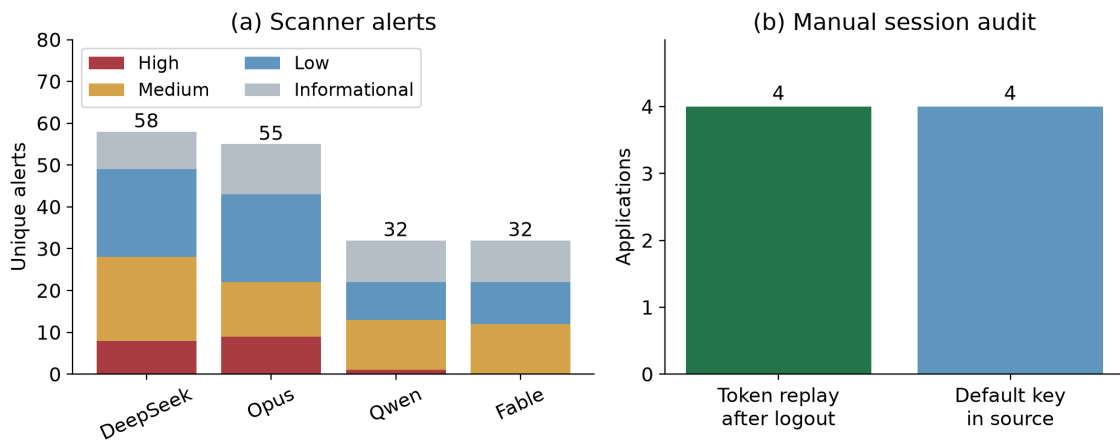
3.1. Manual validation

None of the 18 high-severity findings was confirmed in manual reproduction. The SQL injection alerts were false positives (DBMS mismatch for time-based tests; benign response differences in comparison-based tests), the NoSQL injection produced the same empty result as a non-existent query, the path traversal/source disclosure alerts were triggered by rejected (invalid) identifiers and returned no file content, and the template injection evidence matched only a UUID substring.

Table 3 records the session attributes per application. All four contain a default signing secret; Fable and Opus supply alternative local environment values. Figure 2 distinguishes automatic alerts from manual findings.

Table 3. Session audit. Replay: protected-route status after logout; fallback: literal default in source; override: alternative local environment value.

Application	Secure	JWT lifetime	Replay	Fallback	Override
DeepSeek	No	7 days	200	Yes	Absent
Qwen	No	1 day	200	Yes	Absent
Fable	Yes	7 days	200	Yes	Present
Opus	Yes	7 days	200	Yes	Present

**Figure 2. Scanner alerts and manual session findings. The manual panel counts applications.**

3.2. IDOR

Twenty cross-role tests (five per application) requested administrator resources as a regular user; all returned 401/403/404. Forty supplementary horizontal attempts used two regular users per application, testing order reading and cart/review update and deletion in both directions. Owner controls succeeded; all cross-user requests returned 403/404, without private data or database changes. These tests used isolated APIs and synthetic fixtures, excluding payment and resource creation. The results concern the tested identifiers; matrices and evidence are in the repository.

4. Discussion

The four implementations converged to the expected object-level access control behaviour, but shared a session defect not reported by any scan. Logout removes the cookie on the client, while the previously issued token remains accepted on the server. Where a source-code default is used for signing, knowledge of that secret additionally permits token forgery. Such defects can survive strictly behavioural validation because logout appears to work at the interface level.

The DAST counts differ (H1), forming two pairs; the difference decreases when normalised per reported path, suggesting an influence of *crawler* reach. The severity distinction under H2 did not survive manual reproduction: 18 high-severity false positives concentrated in the implementations with the largest reached surface. Category profiles differed mainly in injection (H3), while missing response headers were common to all applications.

The study used a single run, specification and technology stack, without measuring *crawling* coverage. Further runs should compare explicit security requirements and checks, measuring automated findings and manual confirmations, and broaden the targets, metrics and coverage evaluation. The low SQL-injection threshold favours sensitivity, making reproduction essential. Logout was excluded from scanning, so token replay required the manual audit. Complete generation traces and client/build identifiers were unavailable.

4.1. Possible forms of mitigation

The mitigation of recurrent defects can be treated as a specification problem. A concise “security requirements” document should establish verifiable invariants: effective token revocation at logout, short token lifetimes, explicit JWT validation and signing secrets loaded exclusively from environment variables or secrets services, with rotation.

5. Conclusion

In the experiment, two results stand out. All 18 high-severity findings were refuted by manual reproduction, while tokens remained valid after logout in all four applications. Default signing secrets were present in all four, with alternative environment values supplied in two local configurations. These findings support combining DAST with contextual auditing. The artefacts and reproduction records are available for replication.

Use of Generative AI

Claude Code was used for English translation. OpenAI Codex assisted revision, tables/figures, literature discovery and the horizontal IDOR protocol. The authors reviewed the manuscript and take responsibility for its content.

References

- Daniotti, S., Wachs, J., Feng, X., and Neffke, F. (2026). Who is using AI to code? Global diffusion and impact of generative AI. *Science*, 391(6787):831–835.
- Elder, S., Zahan, N., Shu, R., Metro, M., Kozarev, V., Menzies, T., and Williams, L. (2022). Do I really need all this work to find vulnerabilities? *Empirical Software Engineering*, 27(6):154.
- Hadavi, M. A., Bagherdaei, A., and Ghasemi, S. (2021). IDOT: Black-box detection of access control violations in web applications. *The ISC International Journal of Information Security*, 13(2):117–129.
- Hassan, M. A., Shukur, Z., Hasan, M. K., and Al-Khaleefa, A. S. (2020). A review on electronic payments security. *Symmetry*, 12(8):1344.
- Meske, C., Hermanns, T., Von der Weiden, E., Loser, K.-U., and Berger, T. (2025). Vibe coding as a reconfiguration of intent mediation in software development: Definition, implications, and research agenda. *IEEE Access*, 13:213242–213259.
- MITRE Corporation (2023). 2023 CWE Top 25 Most Dangerous Software Weaknesses. https://cwe.mitre.org/top25/archive/2023/2023_top25_list.html. Accessed 13 October 2026.
- OWASP Foundation (2021). OWASP Top 10:2021 – The Ten Most Critical Web Application Security Risks. <https://owasp.org/Top10/2021/>. Accessed 13 October 2026.
- Qadir, S., Waheed, E., Khanum, A., and Jehan, S. (2025). Comparative evaluation of approaches & tools for effective security testing of Web applications. *PeerJ Computer Science*, 11:e2821.
- Ramírez, L. C., Limón, X., Sánchez-García, Á. J., and Pérez-Arriaga, J. C. (2024). State of the art of the security of code generated by LLMs: A systematic literature review. In *2024 12th International Conference in Software Engineering Research and Innovation (CONISOFT)*, pages 331–339. IEEE.
- Vargas, F. L. S., Mansilha, R. B., and Kreutz, D. (2026). Security-first evaluation of text-to-terraform: Benchmarking LLMs and SLMs for secure IaC generation. arXiv:2608.02672. Accepted for publication at SBSeg 2026.
- ZAP Development Team (2025). ZAP version 2.17.0. <https://www.zaproxy.org/docs/desktop/releases/2.17.0/>. Accessed 13 October 2026.