RESEARCH ARTICLE

JENER Journal of Empirical and Non-Empirical Research
Volume 2, Issue 7, Page 507-511 | Article No. 69
DOI: 10.4898/jener.v2i7.a69
ISSN: 7895-9216
Journal homepage: https://jenerjournal.com
Journal email: editor@jenerjournal.com

JENER Journal
JENER Journal of Empirical and Non-Empirical Research

Investigating the Security Risks and Vulnerabilities Introduction by AI Generated Code in Modern Web Development Environment

Durojaye E. Olatunji, Adegbite Blessing, Gbadamosi O. Adejare, Omitogun liafeez, Omitogun Ayomide & Odunsanya Usman

ARTICLE INFORMATION

Article history:
Published: July 2026

Keywords:
Security Risks
Vulnerabilities
AI
Artificial Intelligence
Code
Web Development

ABSTRACT

The proliferation of AI-driven code generation tools — including GitHub Copilot, Claude Code, Cursor, and OpenAI Codex — has fundamentally altered the pace and nature of modern web application development. By 2026, an estimated 46% of all new code committed to GitHub is AI-generated, with projections suggesting this figure will surpass 60% before year-end. This research investigates the security risks associated with AI-generated code in modern web development, examining documented vulnerability patterns, emerging attack surfaces unique to AI-assisted development workflows, and recommended mitigation strategies. Drawing from industry reports, controlled security experiments, and empirical studies published between 2024 and 2026, this paper finds that AI-generated code introduces security vulnerabilities in 40% to 62% of cases, that security performance has not improved over successive model generations despite advances in functional capability, and that new attack vectors targeting AI coding agents themselves represent an emerging threat category. Practical governance recommendations are presented to help development organizations adopt AI coding tools responsibly without compromising the security posture of production web applications.

1. Introduction

The term "vibe coding" — coined by OpenAI co-founder Andrej Karpathy in February 2025 — describes a workflow in which developers describe software requirements in natural language and AI systems generate the corresponding code. What began as an experimental technique for rapid prototyping has, within less than two years, become the dominant mode of software development for a growing proportion of the global developer population. By 2026, surveys indicate that 92% of developers use some form of AI coding assistance in their workflow, and the Stack Overflow 2024 Developer Survey found that 92.6% of developers use an AI coding assistant at least once a month, with roughly 75% doing so weekly. [7, 10]

The appeal of AI-assisted web development is straightforward: AI tools can accelerate task completion, reduce the burden of repetitive boilerplate code, and lower the barrier to entry for developers without deep expertise in specific frameworks or languages. Enterprise developer productivity studies have documented task completion speed improvements of approximately 21%, and venture-backed startups routinely credit AI coding tools with enabling them to move from concept to production in days rather than months. [7]

However, this acceleration has introduced a new class of security risks that the web development community is only beginning to fully characterize. AI code generation tools are trained on vast public code repositories that contain both secure and insecure patterns. When these models generate code, they replicate both the strengths and vulnerabilities of their training corpora — producing functional applications that may contain structural security flaws invisible during functional testing. Veracode's 2025 GenAI Code Security Report tested more than 100 large language models across 80 security-sensitive coding tasks and found that 45% of AI-generated code samples introduced vulnerabilities classified within the OWASP Top 10. [2, 5]

This research investigates the security risks of AI-generated code in modern web development environments, examining documented vulnerability classes, emerging attack surfaces, real-world breach incidents, and governance frameworks for responsible adoption. Section 2 reviews related literature. Section 3 describes the methodology. Section 4 presents findings and discussion. Section 5 concludes with key recommendations.

2. Related Studies / Literature Review

A rapidly growing body of research has characterized the security risks of AI-generated code across vulnerability prevalence, language-specific risk profiles, developer trust dynamics, attack surface evolution, and real-world incident analysis.

2.1 Vulnerability Prevalence in AI-Generated Code

Veracode's 2025 GenAI Code Security Report — the most comprehensive empirical study of this type to date — analyzed 80 curated coding tasks across more than 100 large language models, evaluating outputs against four OWASP-aligned vulnerability categories: SQL injection (CWE-89), cross-site scripting (CWE-80), log injection (CWE-117), and insecure cryptographic algorithms (CWE-327). The overall finding was stark: 45% of AI-generated code samples failed security tests, with models choosing an insecure implementation path 45% of the time even when a secure alternative was available. The March 2026 update to the same research — titled 'Despite Claims, AI Models Are Still Failing Security' — found the overall security pass rate unchanged at approximately

55%, flat across the entire testing period, even as functional coding benchmarks showed consistent improvement over the same period. [2, 5]

The Cloud Security Alliance's Vibe Coding Security research note (2026) further found that across Fortune 50 enterprises, AI-assisted developers produced commits at three to four times the rate of their peers but introduced security findings at ten times the rate — creating a security debt that accumulated faster than organizations could remediate it. [9]

2.2 Language-Specific Risk Profiles

Vulnerability rates in AI-generated code vary significantly across programming languages. Veracode's testing found Java to be the highest-risk language for AI code generation, with a security failure rate exceeding 70%. Cross-site scripting (CWE-80) showed an 86% failure rate across tested models, while log injection (CWE-117) reached 88%. Python, C#, and JavaScript presented failure rates between 38% and 45%, with language-specific vulnerability patterns: AI-generated JavaScript and Java code tended toward input validation weaknesses (CWE-20), while Python code more frequently exhibited information exposure and path traversal weaknesses (CWE-209, CWE-22). [2, 5, 6]

Georgetown University's Center for Security and Emerging Technology (CSET, 2024) identified three broad categories of risk from AI code generation models: models generating insecure code; the models themselves being vulnerable to attack and manipulation; and downstream cybersecurity impacts including feedback loops in the training of future AI systems. [4]

2.3 Hardcoded Secrets and Credential Exposure

Hardcoded credentials represent one of the most consistently documented vulnerability classes in AI-generated web code. GitGuardian's State of Secrets Sprawl 2026 report recorded 28.65 million hardcoded secrets in public GitHub repositories in 2025, a 34% year-over-year increase — the largest single-year jump on record. AI services specifically accounted for 1,275,105 leaked keys, an 81% increase year-over-year. Research by the Cloud Security Alliance (2026) confirmed that AI-assisted commits introduce secrets at 3.2% compared to a 1.5% baseline for human-only code — a twofold increase. [8, 9]

The persistence of exposed credentials compounds this risk. GitGuardian's research noted that almost 70% of credentials validated as legitimate in 2022 remained active through January 2025. Over 64% were still exposed and unrevoked as of January 2026. In one documented incident, Football Australia inadvertently exposed AWS access keys in website source code generated during development; the keys remained accessible for over 700 days before remediation. [8]

2.4 The 'False Sense of Security' Effect and Developer Trust

Research has consistently identified a problematic disparity between developer trust in AI-generated code and its actual security profile. A 2023 user study found that programmers using AI coding assistants reported significantly higher confidence in the correctness and safety of generated code than its actual vulnerability rates justified. A 2025 SonarSource State of Code Developer Survey found that 53% of developers who shipped AI-generated code later discovered security issues in production, rather than during development or review — indicating that AI-generated vulnerabilities routinely bypass the oversight processes designed to catch them. [1, 3]

Research further documents a compounding 'iteration risk': a 2025 IEEE-ISTAS controlled experiment found a 37.6% increase in critical vulnerabilities after just five rounds of AI-assisted code refinement — meaning that the instinct to iterate and improve AI-generated code through further prompting does not reduce its security risk but actively increases it. [10]

2.5 Emerging Attack Surfaces: Hallucinated Dependencies and Agent Injection

Beyond vulnerabilities in generated code itself, research has documented two emerging attack vectors unique to AI-assisted development workflows. The first is slopsquatting — the exploitation of AI hallucinated package names. The Cloud Security Alliance's April 2026 research analyzed 2.23 million AI-generated code samples from 16 models and found that 19.7% contained at least one hallucinated package name that did not exist in public registries. Critically, 43% of those hallucinated package names reappeared consistently when the same prompts were repeated — creating a predictable, reproducible target for attackers who register the hallucinated names on npm or PyPI in advance. [7, 9]

The second emerging vector is indirect prompt injection targeting AI coding agents. AI coding assistants such as Cursor and GitHub Copilot pull context from external sources — READMEs, open GitHub issues, inline comments, and documentation files. A January 2026 meta-analysis of 78 studies on agentic coding assistants found that attack success rates via indirect prompt injection exceeded 85% when adaptive attack strategies were used. [7]

2.6 Real-World Breach Incidents

The theoretical vulnerability landscape is increasingly corroborated by documented real-world incidents. The Moltbook breach in January 2026 saw a developer build an AI social network using only AI coding tools — without writing a single line of code — and expose 1.5 million API authentication tokens and 35,000 email addresses within 72 hours of launch through a publicly accessible admin endpoint generated with no authentication controls. [10]

A large-scale scan by Escape.tech of 5,600 publicly deployed AI-built applications — developed on platforms including Lovable, Bolt.new, and Base44 — discovered 2,000 highly critical vulnerabilities, 400 exposed secrets including API keys and access tokens, and 175 instances of exposed personally identifiable information including medical records and payment data. Georgia Tech's Vibe Security Radar project, launched in May 2025, tracked 35 CVEs in a single month directly attributable to AI-generated code by March 2026, with researchers estimating the true count is five to ten times higher across the broader open-source ecosystem. [9, 10]

3. Methodology

3.1 Research Design

This research adopts a systematic, qualitative literature review methodology, synthesizing empirical findings, controlled security experiments, large-scale dataset analyses, and real-world incident documentation published between 2024 and 2026. The study is

designed to provide a comprehensive characterization of the security risk landscape of AI-generated web application code rather than to conduct new primary experiments, reflecting both the breadth of evidence now available and the rapidly evolving nature of the threat environment.

3.2 Data Collection

Primary sources include peer-reviewed and industry research published in the IJVRA Journal, the Cloud Security Alliance, Georgetown University's Center for Security and Emerging Technology (CSET), Veracode, GitGuardian, Checkmarx, IBM Security, and arXiv. [1, 2, 3, 4, 5, 6, 7, 8, 9] Sources were selected on the basis of methodological rigor, sample size, recency, and the credibility of the publishing organization. Particular weight was given to studies based on large empirical datasets — such as Veracode's 100+ LLM study, GitClear's 211 million line analysis, GitGuardian's 28 million secret scan, and Escape.tech's 5,600 application audit — as these provide the most statistically reliable characterization of vulnerability rates at scale.

3.3 Vulnerability Classification Framework

Security vulnerabilities identified in reviewed research are classified using two established taxonomies: the OWASP Top 10 for Web Application Security and the MITRE Common Weakness Enumeration (CWE). These frameworks provide a standardized vocabulary for categorizing vulnerability types across languages, models, and deployment contexts, enabling consistent comparison across the diverse literature sources reviewed. [2, 4, 5]

3.4 Scope

This research focuses specifically on AI-generated code deployed in web application development contexts, including both traditional server-rendered applications and modern API-backed single-page applications. The scope encompasses vulnerabilities introduced at the code generation stage, attack vectors targeting the AI coding agent layer itself, and the governance and organizational factors that amplify or mitigate security risk in AI-assisted development workflows.

4. Results and Discussion

4.1 The Scale and Persistence of AI Code Vulnerabilities

The most significant finding across the reviewed literature is the consistency and persistence of high vulnerability rates in AI-generated code. Veracode's testing across more than 100 LLMs found a 45% security failure rate, and the March 2026 update confirmed this rate has not improved despite substantial advances in model capability on functional benchmarks. This decoupling of functional performance from security performance is a critical finding: the market forces driving LLM development — user adoption, benchmark performance, feature delivery speed — do not currently reward security performance, meaning the problem is unlikely to self-correct through model scaling alone. [2, 5]

The implications for production web development are significant. With 46% of all new GitHub code now AI-generated, and AI-assisted developers producing commits at three to four times the rate of their peers while introducing security findings at ten times the rate, the security debt accumulating in production web applications is growing at a rate that traditional security review and remediation programs were not designed to handle. [9]

4.2 Dominant Vulnerability Classes in Web Applications

Research identifies six vulnerability classes as the most consistently occurring in AI-generated web application code:

- Injection Vulnerabilities (CWE-89, CWE-80): SQL injection and cross-site scripting appear at 86% and 88% failure rates respectively in Veracode's testing. AI models generate SQL queries that function correctly for valid input but are injectable under crafted input, and produce HTML output without consistent encoding — replicating insecure patterns prevalent in public training repositories. [2, 5]
- Broken Authentication and Access Control: AI coding agents consistently generate API endpoints that authenticate users but omit per-resource authorization checks, producing Insecure Direct Object Reference (IDOR) vulnerabilities. The Moltbook incident — where an AI-generated admin endpoint had no authentication — illustrates the severity of this pattern at production scale. [3, 10]
- Hardcoded Credentials and Secret Exposure: AI-assisted commits introduce secrets at twice the rate of human-written code. GitGuardian's 2026 scan found 28.65 million hardcoded secrets in public GitHub repositories, with AI services accounting for over 1.27 million leaked keys — an 81% year-over-year increase. [8]
- Server-Side Request Forgery (SSRF): A December 2025 controlled study by Tenzai built 15 identical web applications using five major AI coding agents and found that every application with a URL-handling feature introduced SSRF. All 15 applications also shipped with zero CSRF protection and zero security headers including Content Security Policy, HSTS, X-Frame-Options, and CORS configurations. [10]
- Hallucinated Dependency Injection (Slopsquatting): AI models recommend non-existent packages at a rate of 19.7% across tested samples, with 43% of hallucinated names reappearing deterministically on repeated prompts — creating a reliable, predictable target for supply chain attackers who register the predicted package names on npm or PyPI. [7, 9]
- Over-Permissive Cloud and Infrastructure Configurations: When prompted to generate cloud infrastructure code, AI agents consistently produce over-privileged IAM policies. In documented testing, an AI-generated CloudFormation template for a Lambda function requiring S3 access produced a policy granting access to all S3 buckets in the account rather than the specific target bucket. [10]

4.3 The Iteration Risk: Security Degrades with Prompting

One of the most counterintuitive findings in the current research is the documented degradation of security across iterative AI-assisted development. The instinct to refine and improve AI-generated code through successive prompting sessions does not reduce

accumulated security risk — it compounds it. An IEEE-ISTAS controlled experiment (2025) measured a 37.6% increase in critical vulnerabilities after five rounds of AI-assisted code refinement, a finding consistent with Iterasec's independent observation that security can degrade significantly across repeated vibe coding refinement cycles. [10]

This finding has significant implications for development workflow design: organizations that permit developers to iteratively refine AI-generated code without dedicated security review checkpoints at each iteration may accumulate substantially more vulnerability than organizations that treat each major AI-generated code block as a discrete review unit.

4.4 Attack Surfaces Targeting the AI Agent Layer

Beyond vulnerabilities in generated code itself, the AI coding agent layer introduces two novel attack surfaces that have no direct analogue in traditional web development security:

Indirect prompt injection through contextual data sources represents the first. AI coding assistants routinely pull context from files they read to understand codebases — READMEs, inline comments, open GitHub issues, and documentation. Attackers who can influence these sources can redirect AI agent behavior without direct interaction. The January 2026 meta-analysis of 78 studies found attack success rates exceeding 85% against agentic coding assistants using adaptive prompt injection strategies. [7]

Dependency poisoning through slopsquatting represents the second. Because AI hallucinations about package names are statistically predictable and deterministic — with the same hallucinated name appearing on 43% of repeated identical prompts — attackers can pre-register anticipated hallucinated package names on npm and PyPI with malicious payloads. The attack requires no phishing, no social engineering, and no credential theft; it exploits the reproducible statistical outputs of AI language models. [9]

4.5 The False Sense of Security and Oversight Gap

Research consistently identifies developer overtrust in AI-generated code as a structural risk amplifier. AI-generated code passes syntax checks, produces correct functional outputs, and generates no runtime errors on valid input — characteristics that are indistinguishable from secure code in routine development workflows. This creates a false confidence effect: 53% of developers discovered security issues in AI-generated code only after it had been deployed to production, rather than during development or review phases. [1, 3]

Gartner's projections reflect the severity of the oversight gap: the firm estimated that by 2027, 25% of all software defects will stem from inadequate oversight of AI-generated code, and further projected that prompt-to-application development approaches adopted by non-expert developers will increase software defects by 2,500% by 2028 — a scale that existing quality assurance and security review infrastructure is not designed to handle. [1, 3]

4.6 Discussion

The accumulated evidence presents a coherent picture: AI-generated code for web applications is systematically functional and systematically insecure in ways that are largely invisible to the automated and manual review processes that currently govern software quality in most development organizations. The security risks are not attributable to any single model or tool — Veracode's testing of more than 100 LLMs found consistent failure rates across the field — nor are they improving with model capability improvements. [2, 5]

The fundamental issue is architectural: LLMs optimize for code that satisfies the prompt and passes functional tests. Security is a non-functional requirement that is rarely specified in prompts and cannot be verified by functional testing alone. Until security requirements are either systematically embedded in AI code generation prompts, enforced by secondary security-focused models, or validated by automated security scanning integrated into generation workflows, the gap between functional correctness and security correctness will persist regardless of model generation. [4, 6]

5. Conclusion

This research has investigated the security risks of AI-generated code in modern web development, finding that vulnerability rates of 40% to 62% are well-documented across multiple independent studies, that these rates have not improved as model functional performance has advanced, and that new attack surfaces targeting the AI coding agent layer itself — including indirect prompt injection and slopsquatting supply chain attacks — represent emerging threats with no direct analogue in conventional web security. [2, 5, 7, 9]

Key recommendations for development organizations adopting AI code generation in web application contexts include:

- Treat all AI-generated code as untrusted until verified — requiring mandatory security review for authentication, authorization, payment processing, and any code handling sensitive user data, regardless of the developer's confidence in AI-generated output [1, 3]
- Integrate automated Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST) into the CI/CD pipeline specifically for AI-generated code segments, using tools capable of detecting semantic vulnerabilities beyond signature-based pattern matching [5, 6]
- Implement automated secret scanning and dependency validation in the code generation workflow to catch hardcoded credentials and hallucinated package names before they reach repositories [8, 9]
- Enforce security-explicit prompting practices — specifying security requirements, authentication expectations, and input validation standards in AI code generation prompts rather than relying on models to apply secure defaults [4, 6]
- Treat each major iteration of AI-generated code as a new review unit, given documented evidence that iterative AI refinement increases rather than reduces vulnerability counts [10]
- Monitor the AI coding agent attack surface — auditing external context sources (READMEs, documentation, dependency files) for embedded prompt injection attempts, and validating all package dependencies against known registries before installation [7, 9]

- Apply the OWASP Top 10 and NIST Secure Software Development Framework as organizational reference standards for evaluating AI-generated web application code, given documented alignment between AI vulnerability classes and established web security risk taxonomies [4, 5]

The security risks of AI-generated code in web development are not theoretical — they are empirically documented, scale-consistent, and accelerating as adoption outpaces governance. Organizations that implement structured, security-aware AI adoption frameworks will be best positioned to capture the development velocity advantages of AI coding tools without accumulating the security debt that currently characterizes unreviewed AI-assisted web development. [1, 4, 9]

References

[1] AI Journal. (2025). The Hidden Security Risks of AI-Generated Code. Retrieved from https://aijourn.com/the-hidden-security-risks-of-ai-generated-code/

[2] Security Today. (2025). AI-Generated Code Poses Major Security Risks in Nearly Half of All Development Tasks — Veracode 2025 GenAI Code Security Report. Retrieved from https://securitytoday.com/articles/2025/08/05/ai-generated-code-poses-major-security-risks-in-nearly-half-of-all-development-tasks.aspx

[3] Cloud Security Alliance. (2025). Understanding Security Risks in AI-Generated Code. Retrieved from https://cloudsecurityalliance.org/blog/2025/07/09/understanding-security-risks-in-ai-generated-code

[4] Georgetown CSET. (2024). Cybersecurity Risks of AI-Generated Code. Retrieved from https://cset.georgetown.edu/publication/cybersecurity-risks-of-ai-generated-code/

[5] AI Journal. (2025). The Hidden Vulnerabilities in AI-Generated Code That Every Developer Should Know. Retrieved from https://aijourn.com/the-hidden-vulnerabilities-in-ai-generated-code-that-every-developer-should-know/

[6] arXiv. (2025). On Fixing Insecure AI-Generated Code through Model Fine-Tuning and Prompting Strategies. Retrieved from https://arxiv.org/pdf/2605.05867

[7] Arnica / Checkmarx. (2026). Vibe Coding Security Risks. Retrieved from https://www.arnica.io/blog/vibe-coding-security-risks and https://checkmarx.com/blog/security-in-vibe-coding/

[8] GitGuardian. (2026). The State of Secrets Sprawl 2026: AI-Service Leaks Surge 81% and 29M Secrets Hit Public GitHub. Retrieved from https://www.gitguardian.com/state-of-secrets-sprawl-2026

[9] Cloud Security Alliance AI Safety Initiative. (2026). Vibe Coding's Security Debt: The AI-Generated CVE Surge. Retrieved from https://labs.cloudsecurityalliance.org/research/csa-research-note-ai-generated-code-vulnerability-surge-2026/

[10] IBM Security / BeyondScale / Modall / OX Security. (2026). Vibe Coding Security Risks: Enterprise Perspectives. Retrieved from https://www.ibm.com/think/insights/vibe-coding-security-risks and https://beyondscale.tech/blog/vibe-coding-security-risks-enterprise