Vibe Coding Security Crisis: Credential Sprawl and SDLC Debt

Security risks from AI-generated code in enterprise software development

Unofficial AI-assisted Research

Cloud Security Alliance AI Safety Initiative

2026-03-31

Key Takeaways

- AI-assisted commits expose secrets at more than twice the rate of human-only commits — 3.2% versus 1.5%. Separately, public GitHub saw a 34% year-over-year increase in hardcoded credentials discovered in 2025 — the largest single-year jump on record [1].
- Independent studies find that AI-generated code introduces security vulnerabilities in 45% of development tasks [2], produces 2.74 times more security issues per pull request than human-authored code [3], and has a 100% failure rate on basic security controls such as CSRF protection across all 15 production applications tested in one study [4].
- A new attack surface created by AI coding tools — including the Rules File Backdoor [5] and slopsquatting [6] — has no precise analog in traditional secure development guidance, though it shares characteristics with IDE plugin supply chain attacks and typosquatting.
- CVEs formally attributed to AI-generated code jumped from 6 in January 2026 to 35 in March 2026, and researchers estimate the actual count is 5–10 times higher because most AI tools leave no commit metadata [7].
- Existing SDLC frameworks, security training programs, and CI/CD tooling were designed for human-authored code and require deliberate extension to address AI-specific failure patterns.

Background

On February 2, 2025, researcher and former OpenAI Research Scientist Andrej Karpathy coined the term "vibe coding" to describe a development modality in which programmers describe desired functionality in natural language, accept AI-generated output without detailed review, and rely on follow-up prompts to address problems rather than reasoning through the code directly [8]. Karpathy framed this as "fully giving in to the vibes" — a deliberate departure from the disciplined reading and reasoning that characterizes traditional programming practice. Collins English Dictionary subsequently named "vibe coding" its Word of the Year for 2025 [9], reflecting how quickly the concept spread beyond its original technical context.

What began as an indie developer phenomenon accelerated into enterprise practice throughout 2025. A JetBrains developer survey conducted in early 2026, as summarized by enterprise technology analysts [10], found that more than a third of enterprise development teams were using AI to generate substantial code blocks from natural language prompts. Industry analysis reports that 46% of all code accepted by GitHub Copilot users is now AI-generated [11], and a Q4 2025 study of over 135,000 developers, as reported by workforce analytics firm Second Talent [12], found 91% AI coding adoption within the sampled population, with 22% of merged code being AI-authored. Stack Overflow's 2025 Developer Survey found 84% of professional developers either actively using or planning to use AI coding tools — while simultaneously reporting that developer trust in AI accuracy had fallen from 40% to 29% year-over-year [13].

This combination — near-universal adoption alongside declining confidence in correctness — defines the security environment organizations now face. Development organizations are integrating AI-generated code at scale into production systems, while governance frameworks and developer training specific to AI-generated code remain early-stage and existing security tooling requires targeted extension to address AI-specific failure modes. The 10x increase in monthly security findings documented in Fortune 50 enterprises [19] suggests that security debt is accumulating faster than existing remediation programs can address it.

Security Analysis

Systematic Vulnerability Patterns in AI-Generated Code

The security risks of AI-generated code are not random or edge-case in nature — multiple independent research efforts have found consistent, reproducible failure patterns. These findings vary in scope and methodology, and the most alarming figures tend to come from studies of consumer-grade or ungoverned development environments; organizations with mature code review processes and dedicated security tooling may see different outcomes. The aggregate picture, however, is consistent across a range of study designs and cannot be dismissed as an artifact of methodology.

Veracode's 2025 GenAI Code Security Report, which tested over 100 LLMs on 80 curated coding tasks, found that AI-generated code introduces security vulnerabilities in 45% of cases [2]. Java showed the highest failure rate at over 70%, while Python, C#, and JavaScript ranged from 38% to 45%. Veracode observed that despite ongoing functional improvements across model generations, security pass rates remained flat through mid-2025 — suggesting that the optimization pressures driving model

improvements do not naturally align with secure code generation. Performance varied significantly across models, meaning that enterprise deployments using security-optimized configurations may experience lower failure rates than the aggregate suggests.

An empirical study published in the ACM Transactions on Software Engineering and Methodology in February 2025 analyzed 733 real-world code snippets generated by GitHub Copilot, CodeWhisperer, and Codeium and found security weaknesses in 29.5% of Python snippets and 24.2% of JavaScript snippets, spanning 43 distinct CWE types [14]. The most prevalent weaknesses included insufficient randomness (CWE-330), code injection (CWE-94), and cross-site scripting (CWE-79) — eight of which appear on the 2023 CWE Top 25 Most Dangerous Software Weaknesses.

A December 2025 study by CodeRabbit compared 320 AI-co-authored pull requests against 150 human-only pull requests from open-source repositories and found that AI-authored PRs generated 1.7 times more issues overall and 2.74 times more security issues specifically [3]. Security issue categories where AI code underperformed were not marginal differences: AI code was 1.88 times more likely to contain improper password handling, 1.91 times more likely to introduce insecure direct object references, and 1.82 times more likely to include insecure deserialization vulnerabilities. A large-scale academic study presented at IEEE ISSRE 2025, covering 500,000+ code samples in Python and Java from both human developers and models including ChatGPT, DeepSeek-Coder, and Qwen-Coder, confirmed that AI code exhibits more high-risk security vulnerabilities despite simpler structural complexity compared to human-authored code [15].

These findings are validated at the production level. A December 2025 analysis by security firm Tenzai examined 15 production applications built using five major AI coding tools and identified 69 vulnerabilities across the sample [4]. Every application tested lacked CSRF protection and had no security headers configured, and every tool introduced server-side request forgery (SSRF) vulnerabilities — a clean sweep of basic security failures across all 15 applications in the study. Separately, API security firm Escape.tech scanned over 1,400 vibe-coded production applications and found that 65% had security issues and 58% contained at least one critical vulnerability, including over 400 exposed secrets and 175 instances of exposed personally identifiable information such as bank account data [16].

The Credential Sprawl Crisis

The most immediately measurable enterprise impact of AI-generated code is the accelerating exposure of hardcoded credentials. GitGuardian's State of Secrets Sprawl 2026 report, published March 17, 2026, documented 28.65 million new hardcoded secrets in public GitHub commits during 2025 — a 34% year-over-year increase representing the largest single-year jump ever recorded [1]. AI-assisted commits showed a 3.2% secret-leak rate compared to a 1.5% baseline across all public GitHub commits, indicating that AI-generated code roughly doubles the baseline credential exposure rate.

The composition of leaked credentials is consistent with the AI adoption curve. AI-service credentials — API keys for LLM providers, embedding services, and AI platforms — increased 81% year-over-year in 2025, reaching 1,275,105 detected leaks [1]. One illustrative data point: 113,000 DeepSeek API keys were discovered in public repositories in 2025 alone. A parallel problem has emerged in the Model Context Protocol ecosystem: GitGuardian found 24,008 unique secrets in MCP-related configuration files on public GitHub, of which 2,117 remain valid credentials [1]. A significant contributing structural factor is that official MCP quickstart documentation presents API keys hardcoded directly in configuration examples, creating patterns that developers replicate without always recognizing the security implication. Developer time pressure and the absence of IDE-level secrets detection in many AI tool workflows reinforce this pattern.

The remediation picture is equally concerning. GitGuardian found that nearly 70% of credentials identified as legitimate in 2022 remained valid through January 2025, and 64% remained unrevoked as of January 2026 [1]. This multi-year persistence transforms every credential leak into a long-duration exposure window. Snyk's analysis of the same credential landscape found that secrets spread well beyond source code: 2.4% of Slack channels contain at least one leaked secret, 6.1% of Jira tickets expose credentials, and Docker Hub hosts over 10,000 images with embedded credentials [17]. The CSA Top Threats to Cloud Computing Deep Dive 2025 identified hard-coded credentials as a contributing factor in multiple breach case studies, including the Football Australia incident in which AWS access keys were embedded in website source code and remained exposed for over 700 days [18].

Emergent Attack Classes with No Human-Code Analog

Beyond the replication of classical vulnerabilities, AI coding tools have introduced attack surfaces that did not exist in traditional software development. Three warrant particular attention.

The Rules File Backdoor, discovered by Pillar Security on March 18, 2025, exploits the configuration files that developers use to customize AI coding tool behavior [5]. Attackers embed hidden Unicode characters — zero-width joiners, bidirectional text markers, and similar invisible glyphs — in `.cursor/rules` files and analogous configuration objects for GitHub Copilot and other tools. The AI system processes these hidden instructions and silently generates backdoored code that appears clean in normal code review. GitHub declined to classify the technique as a vulnerability but implemented a hidden-Unicode warning for Copilot on May 1, 2025. The broader implication is that AI coding tool configuration files have become an attack vector requiring their own integrity controls.

Slopsquatting, named by Seth Larson and Andrew Nesbitt in April 2025, exploits a consistent pattern in which AI coding tools recommend package names that do not exist [6]. An analysis of 576,000 AI-generated code samples found that 20% recommended non-existent package names, totaling 205,474 unique hallucinated packages. Notably, 43% of these hallucinated packages were consistently

recommended across ten separate queries, and 58% appeared more than once — indicating systematic rather than random hallucination [6]. Attackers pre-register these hallucinated package names on public registries such as npm and PyPI, and AI-assisted developers unknowingly install malicious code under a name the AI tool itself selected. The technique is a direct evolution of typosquatting but differs in a significant respect: the developer is not mistyping a package name but trusting an AI suggestion they have no prior reason to question.

Georgia Tech's Systems Software and Security Lab launched the Vibe Security Radar in May 2025 to track CVEs formally attributable to AI coding tools [7]. CVE counts attributed to AI-generated code climbed from 6 in January 2026 to 15 in February and 35 in March 2026. Two specific CVEs — CVE-2025-55526 (CVSS 9.1 directory traversal in n8n-workflows) and GHSA-3j63-5h8p-gf7c (improper input handling in the x402 SDK) — were linked to code generated by AI coding tools in August 2025 [7]. Both identifiers have been confirmed in authoritative registries (NIST NVD and the GitHub Advisory Database, respectively). The Georgia Tech researchers estimate the true count of AI-generated-code vulnerabilities in the open-source ecosystem is 400–700 cases, approximately 5–10 times the detected figure, because most AI coding tools do not leave identifiable commit metadata.

SDLC Security Debt at Scale

The aggregate effect of these patterns is a form of security debt accumulating at a rate that existing remediation programs were not designed to address. Analysis published by SecurityWeek, citing enterprise security data, documented a 10x increase in security findings per month within Fortune 50 enterprises between December 2024 and June 2025 — from approximately 1,000 to over 10,000 monthly vulnerabilities in a six-month period [19]. Industry analysts have projected that 75% of companies will see their technical debt reach moderate-to-high severity in 2026, with rapid AI adoption cited as a primary contributing factor [20]. Both figures represent directional indicators rather than precise measurements, but they are consistent with the vulnerability growth patterns documented in the primary research cited above.

Evidence suggests that the SDLC mechanisms that traditionally governed code quality — peer review, code author familiarity, incremental change management — function differently when a developer is primarily reviewing AI-generated output rather than reasoning through code they authored. Stack Overflow's 2025 Developer Survey found that 66% of developers report spending more time fixing "almost-right" AI-generated code [13], a finding that reflects the quality oversight burden without yet capturing the security-specific review gap. Existing security training programs were designed around common human error patterns and do not address the systematic failure modes — credential embedding,

missing security headers, SSRF, CSRF gaps — that AI tools introduce. Without deliberate extension of governance and training programs, this gap creates the conditions for sustained security debt accumulation that accelerates ahead of the controls designed to govern it.

Recommendations

Immediate Actions

Organizations should treat AI coding tools as a new category of input requiring dedicated security controls rather than an extension of existing developer tooling. The most actionable immediate step is deploying automated secrets detection as a pre-commit gate across all repositories, with specific detection signatures for AI-service credentials (LLM API keys, vector database tokens, embedding service credentials) in addition to traditional cloud and SaaS secrets. GitGuardian, Semgrep, and Trufflehog all offer policies applicable to this use case. Secrets detection should apply at commit time, not only on push or in CI/CD pipelines, to reduce the window between generation and exposure.

Security teams should audit all AI coding tool configuration files — `.cursor/rules`, GitHub Copilot workspace settings, and any equivalent files in the development environment — for hidden Unicode characters and unauthorized instructions. Following the Rules File Backdoor disclosure, these files should be treated as potentially adversarially modified artifacts and should fall under the same integrity controls applied to CI/CD pipeline configuration.

Short-Term Mitigations

Development organizations should integrate static analysis security testing (SAST) as a required, blocking gate for AI-assisted code contributions, not an advisory check. Semgrep, Veracode, and SonarQube all support detection of the CWE categories most frequently introduced by AI tools, including injection flaws, XSS, insecure deserialization, and missing security controls such as CSRF protection. The key architectural principle is that AI-generated code must be treated as untrusted input — equivalent in posture to externally-sourced library code — until it has cleared the same security gates applied to any other code entering the codebase.

Package dependency management requires explicit attention to the slopsquatting risk. Organizations should enforce dependency lockfiles and allowlists, validate all package names against known registries before installation, and use tools such as Socket.dev or Phylum to assess the provenance and integrity of

newly introduced packages. Software Bills of Materials (SBOMs) should be generated for any codebase with meaningful AI-assisted code contribution, both to inventory dependencies and to create an auditable baseline for future comparison.

For credential management specifically, teams should adopt secret scanning as part of developer IDE configuration — not only in CI/CD — and move credentials from configuration files to dedicated secrets management systems (HashiCorp Vault, AWS Secrets Manager, Azure Key Vault) for any environment where AI coding tools are in use. Any developer workflow that involves AI tools generating database connection strings, API client initialization code, or configuration files should be treated as high-credential-risk and routed through a secrets-manager-aware code template.

Strategic Considerations

Enterprises integrating AI coding tools at scale should revisit their security champion and developer security training programs to address the specific failure patterns introduced by AI-generated code. Traditional secure coding training addresses how developers make mistakes; the new training gap is helping developers understand what to look for when reviewing code they did not write and did not fully reason through. OWASP's Top 10 for LLM Applications 2025, particularly LLM09 (Misinformation/Overreliance) and LLM02 (Sensitive Information Disclosure), provides a foundation for this curriculum [21].

Organizations should consider tracking the provenance of AI-assisted code contributions in their repositories — either through commit metadata, developer attestation, or tooling-level attribution — to enable security analytics that distinguish AI-generated components for targeted review. The Georgia Tech Vibe Security Radar project demonstrates that such provenance data, even partially available, enables meaningful vulnerability attribution.

At the governance level, security leadership should establish policy addressing when and how AI coding tools may be used for code that will run in production, handle PII, process authentication, or interact with external services. The absence of such policy is itself a risk exposure: when developers have no guidance, adoption and its associated risks accelerate ahead of the security controls designed to govern them.

CSA Resource Alignment

This research note connects to several foundational CSA frameworks and publications.

The **AI Controls Matrix (AICM)**, CSA's comprehensive framework for AI system security, provides directly applicable controls across three domains: Model Security (13 controls addressing secure AI component deployment), Supply Chain Management (16 controls addressing AI tool provenance, component integrity, and third-party dependency risk), and Data Security and Privacy (24 controls addressing the handling of sensitive data in AI-assisted development contexts) [22]. Organizations treating AI coding tools as in-scope AI systems under AICM will find that the supply chain controls apply directly to both the tools themselves and the code they produce.

The **MAESTRO framework** for agentic AI threat modeling addresses the threat class of secrets leakage through AI agents — a pattern directly applicable to AI coding assistants that operate with access to developer environments, configuration files, and credential contexts [23]. MAESTRO's agentic red teaming guidance identifies supply chain and dependency attacks as a primary threat category, consistent with the slopsquatting and Rules File Backdoor risks described in this note.

The CSA publication **AI Organizational Responsibilities: AI Tools and Applications** (January 2025) provides RACI models for secure AI-assisted development and specifically addresses hard-coded credentials as a developer responsibility requiring organizational governance structures [24]. The Football Australia breach case study documented in that publication provides a training reference for development teams adopting AI coding tools.

The **CSA Top Threats to Cloud Computing Deep Dive 2025** identified insecure software development as a contributing factor in half of the analyzed cloud security breaches reviewed, providing empirical grounding for the SDLC security debt concern raised in this note [18]. The managing privileged access publication from January 2026 provides applicable guidance on non-human identity management relevant to the AI service credential sprawl problem, including Just-In-Time and Just-Enough-Access patterns appropriate for managing the LLM API credentials that AI coding tools require [25].

Organizations pursuing STAR registration should treat AI coding tool adoption as a material change to their software development risk posture, warranting assessment updates under applicable AICM controls (Model Security and Supply Chain Management domains) and CCM controls governing secure development (AIS domain) and supply chain management (STA domain).

References

[1] GitGuardian, "The State of Secrets Sprawl 2026," March 17, 2026. https://blog.gitguardian.com/the-state-of-secrets-sprawl-2026/

[2] Veracode, "AI-Generated Code Poses Major Security Risks in Nearly Half of All Development Tasks," July 30, 2025. https://www.veracode.com/blog/research/ai-generated-code-security-risks (original BusinessWire press release URL unavailable; statistics confirmed via Veracode blog and syndicated coverage)

[3] CodeRabbit, "State of AI vs Human Code Generation Report," December 17, 2025. https://www.coderabbit.ai/blog/state-of-ai-vs-human-code-generation-report

[4] Tenzai, "AI Coding Tools Introduce 69 Vulnerabilities in 15-App Study," December 2025. https://awesomeagents.ai/news/vibe-coding-security-69-vulnerabilities/

[5] Pillar Security, "New Vulnerability in GitHub Copilot and Cursor: How Hackers Can Weaponize Code Agents," March 18, 2025. https://www.pillar.security/blog/new-vulnerability-in-github-copilot-and-cursor-how-hackers-can-weaponize-code-agents

[6] Seth Larson and Andrew Nesbitt, "Slopsquatting: AI Hallucinated Code Dependencies Become New Supply Chain Risk," April 2025. https://www.bleepingcomputer.com/news/security/ai-hallucinated-code-dependencies-become-new-supply-chain-risk/

[7] Hanqing Zhao / Georgia Tech SSLab, "Vibe Security Radar: Tracking AI-Generated Code CVEs," Infosecurity Magazine, March 26, 2026. https://www.infosecurity-magazine.com/news/ai-generated-code-vulnerabilities/

[8] Andrej Karpathy, X (formerly Twitter), February 2, 2025. https://x.com/karpathy/status/1886192184808149383 (authentication required; tweet existence and content confirmed via Wikipedia [9] and cached sources)

[9] Wikipedia, "Vibe coding," citing Collins English Dictionary Word of the Year 2025. https://en.wikipedia.org/wiki/Vibe_coding

[10] DigitalApplied.com, "Vibe Coding in Enterprise: AI Code Security Risk Guide," summarizing JetBrains 2026 developer survey. https://www.digitalapplied.com/blog/vibe-coding-enterprise-security-risks-ai-generated-code (secondary source; JetBrains primary survey not directly linked)

[11] Industry analysis of GitHub Copilot AI code acceptance rates, 2025. https://medium.com/@reliabledataengineering/ai-is-writing-46-of-all-code-github-copilots-real-impact-on-15-million-developers-787d789fcfdc (secondary source)

[12] Second Talent, "AI Coding Assistant Statistics Q4 2025," citing 135,000+ developer impact report. https://www.secondtalent.com/resources/ai-coding-assistant-statistics/ (secondary source)

[13] Stack Overflow, "2025 Developer Survey Results," December 29, 2025. https://stackoverflow.blog/2025/12/29/developers-remain-willing-but-reluctant-to-use-ai-the-2025-developer-survey-results-are-here/

[14] ACM Transactions on Software Engineering and Methodology, "Security Weaknesses of Copilot-Generated Code: An Empirical Study," February 2025. https://dl.acm.org/doi/10.1145/3716848

[15] Dessertlab / IEEE ISSRE 2025, "Human-Written vs. AI-Generated Code: A Large-Scale Study of Defects, Vulnerabilities, and Complexity," arXiv:2508.21634, August 2025. https://arxiv.org/abs/2508.21634

[16] Escape.tech, "The State of Security of Vibe Coded Apps," October 2025. https://escape.tech/state-of-security-of-vibe-coded-apps

[17] Snyk, "Why 28 Million Credentials Leaked on GitHub in 2025." https://snyk.io/articles/state-of-secrets/

[18] Cloud Security Alliance, "Top Threats to Cloud Computing Deep Dive 2025," May 15, 2025. https://cloudsecurityalliance.org/research/topics/top-threats

[19] SecurityWeek, "How to Eliminate the Technical Debt of Insecure AI-Assisted Software Development," 2026. https://www.securityweek.com/how-to-eliminate-the-technical-debt-of-insecure-ai-assisted-software-development/

[20] Salesforce Ben, "2026 Predictions: It's the Year of Technical Debt Thanks to Vibe Coding," citing Forrester, 2026. https://www.salesforceben.com/2026-predictions-its-the-year-of-technical-debt-thanks-to-vibe-coding/ (secondary source; Forrester primary report not directly linked)

[21] OWASP, "Top 10 for LLM Applications 2025," genai.owasp.org. https://genai.owasp.org/resource/owasp-top-10-for-llm-applications-2025/

[22] Cloud Security Alliance, "Introductory Guidance to the AI Controls Matrix (AICM)," November 17, 2025. https://cloudsecurityalliance.org/research/topics/artificial-intelligence

[23] Cloud Security Alliance, "Agentic AI Red Teaming Guide / MAESTRO Framework," 2025. https://cloudsecurityalliance.org/research/topics/artificial-intelligence

[24] Cloud Security Alliance, "AI Organizational Responsibilities: AI Tools and Applications," January 27, 2025. https://cloudsecurityalliance.org/research/topics/artificial-intelligence

[25] Cloud Security Alliance, "Managing Privileged Access in the Cloud-First World," January 7, 2026. https://cloudsecurityalliance.org/research/topics/identity-and-access-management