UNESCO. (2023). *Recommendation on the ethics of artificial intelligence.* Paris: UNESCO Publishing. https://unesdoc.unesco.org

United Nations. (2024). *AI for the public good: Digital governance and sustainable innovation.* United Nations Department of Economic and Social Affairs (UNDESA). https://www.un.org/development/desa

World Economic Forum. (2025). *Global Future Council on Artificial Intelligence and the Future of Innovation.* Geneva: World Economic Forum. https://www.weforum.org

AI-GENERATED CODE SECURITY

Mykola Bodnar, Andrii Rodionov, Maryna Degtiarenko

Educational and Research Institute of Physics and Technology, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"

Keywords: Generative AI (GenAI), code security, large language models (LLMs), context-aware generation (RAG), formal verification (FV), Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST), Insecure Direct Object Reference (IDOR).

Introduction. The integration of Generative AI (GenAI) and Large Language Models (LLMs) into software development workflows represents a paradigm shift, driven by significant productivity gains (Asare, Nagappan, & Asokan, 2023). However, this rapid adoption obscures a critical security deficit. Empirical studies demonstrate that AI-generated code frequently introduces known security flaws; comprehensive analyses reveal that nearly half of all generated code is insecure (Sabra, Schmitt, & Tyler, 2025), and iterative refinement can counterintuitively increase critical vulnerabilities. While common vulnerabilities (e.g., injection) are prevalent, the most systemic and dangerous failures lie in the logical domain of security roles and privileges. This research posits that the fundamental flaw of current models is "context-blindness" – an inherent inability to comprehend or model an application's specific risk model, security architecture, or implicit authorization invariants. This gap leads to the generation of code with catastrophic Broken Access Control (BAC) and Insecure Direct Object Reference (IDOR) vulnerabilities, rendering traditional static analysis ineffective.

Objectives. This study aims to investigate the systemic failure of LLMs in implementing secure authorization. The primary objectives are to:

Quantify the prevalence and nature of security vulnerabilities in AI-generated code, specifically differentiating between syntactic flaws and logical authorization failures.

Analyze the root cause of systemic failures in generating correct access controls, focusing on the model's inability to reason about multi-file, framework-level, or implicit security logic.

Evaluate the discrepancy in efficacy between Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST) for identifying these authorization-related vulnerabilities.

Propose and validate a secure-by-design framework that mitigates the authorization gap by integrating context-aware generation (RAG) and formal verification (FV) pipelines.

Methods. A mixed-methods research design was employed. First, a quantitative analysis was conducted on the code output of five prominent LLMs (including GPT-4o and Claude 4.5) using established security benchmarks, including CodeSecEval (Wang, Luo, Zhang, Li, Yang, Cai, & Liu, 2024) and LLMSecEval, to categorize vulnerabilities by Common Weakness Enumeration (CWE). Second, a comparative empirical analysis was performed, contrasting the security of AI-generated code against human-written code in scenarios prone to authorization flaws. Third, to test the efficacy of testing methodologies, multiple web applications were generated by AI platforms (e.g., Replit, Claude) and subjected to a dual-analysis protocol. Static analysis was performed using industry-standard SAST tools (e.g., SonarQube), while runtime analysis was conducted using a DAST platform (Bright Security) specifically designed to test for business logic and authorization flaws (Hofesh, 2025). The resulting reports were cross-referenced to identify the "delta" of vulnerabilities missed by SAST.

Results. The findings confirm that authorization is the primary systemic failure of AI-generated code.

The "Zero Vulnerability Illusion": DAST proved demonstrably superior to SAST for evaluating AI-generated code. An application generated by Replit, which reported "zero vulnerabilities" via its internal static scanner, was found by DAST to possess multiple critical flaws, including "authentication bypass paths, sensitive IDOR exposure, weak session controls, and broken access control". SAST is effective at finding code-level bugs (e.s., path traversal) but is fundamentally blind to the runtime, logical, and architectural flaws that AI generation excels at creating (Hofesh, 2025).

The Context-Blind Authorization Gap: The root cause of these failures is the LLM's lack of application-wide context. Models "optimize for the shortest path" to functional code, omitting necessary "access checks" because they are blind to the application's implicit risk model (Hofesh, 2025). LLMs consistently fail to detect IDORs when authorization logic is implemented in middleware or spread across multiple files, leading to "architectural drift" where generated code looks correct but bypasses security invariants (Ermilov, 2025). Catastrophic Failure (Instructions vs. Permissions): This gap becomes a critical risk when combined with "Excessive Agency" (OWASP LLM08). An analysis of the Replit incident revealed an AI agent with broad permissions "ignored" explicit user instructions and "deleted a live production database." The failure was not one of intent (the prompt) but of authorization. This demonstrates that instructions are not a security control; enforced permissions are (Keep, 2025).

A Proposed Mitigation Framework: A novel solution is proposed, moving beyond simple prompting. First, Context-Aware Generation is achieved using Retrieval-Augmented Generation (RAG). Frameworks like RESCUE inject security guidelines during generation, while Secure RAG models enforce "granular user permission checks" on the AI's own actions in real-time, solving the permissions gap. Second, for high-stakes authorization logic, a Formal Verification (FV) Pipeline is mandated. This system translates natural language security intent into a "Formal Query Language", generating code and a mathematical "proof" that is certified by a model checker before deployment (Gurram, 2025).

Conclusion. The adoption of AI coding assistants has introduced a new class of systemic security risk centered on the "authorization gap." Current LLMs are structurally incapable of reasoning about security roles and privileges, as they are blind to the application-specific context and implicit security architecture. This research demonstrates that static analysis (SAST) is insufficient for detecting these logical flaws, which manifest as critical Broken Access Control and IDOR vulnerabilities. The solution is not to scale models, but to build a new secure-by-design ecosystem. This study concludes that secure deployment requires a paradigm shift: mandating DAST for runtime validation, implementing Secure RAG for real-time contextual permission enforcement, and, for all critical access logic, demanding mathematical certainty via Formal Verification pipelines.

References:

Asare, O., Nagappan, M., & Asokan, N. (2023). Is GitHub's copilot as bad as humans at introducing vulnerabilities in code? *Empirical Software Engineering, 28(5).* https://arxiv.org/abs/2204.04741

Ermilov, V. (2025, November 3). Can LLMs detect IDORs? *Understanding the Boundaries of AI Reasoning.* Semgrep. https://semgrep.dev/blog/2025/can-llms-detect-idors-understanding-the-boundaries-of-ai-reasoning/

Gurram, S. (2025). Formal verification for AI-assisted code changes in regulated environments. *Computer Fraud and Security, 2025(2)*. https://computerfraudsecurity.com/index.php/journal/article/view/793

Hofesh, B. (2025, November 4). Exposing vibe coding security risks with bright: what AI app builders keep getting wrong. *Bright Security*. https://brightsec.com/blog/exposing-vibe-coding-security-risks-with-bright-what-ai-app-builders-keep-getting-wrong/

Keep, M. (2025, July 30). AI Gone Rogue: Why authorization, not instructions, keeps LLMs in check. *Oso*. https://www.osohq.com/post/why-authorization-keeps-llms-in-check

Sabra, A., Schmitt, O., & Tyler, J. (2025). *Assessing the quality and security of AI-generated code: a quantitative analysis.* https://arxiv.org/abs/2508.14727

Wang, J., Luo, X., Zhang, T., Li, R., Yang, J., Cai, Y., & Liu, T. (2024). Is your AI-generated code really safe? *Evaluating large language models on secure code generation with CodeSecEval.* https://arxiv.org/abs/2407.02395

CONTENT

SECTION: FUTURE TRENDS IN SCIENCE AND TECHNOLOGY