

Evaluating the security of JavaScript code generated by GitHub Copilot using static analysis tools

Partheepan V.^{1*}, Wijerathna R.M.K.K.²

¹Department of Software Engineering, Faculty of Computing,
Sabaragamuwa University of Sri Lanka, Sri Lanka

²Department of Computing and Information Systems, Faculty of Computing,
Sabaragamuwa University of Sri Lanka, Sri Lanka

*vpartheepan@std.appsc.sab.ac.lk

Artificial Intelligence (AI) tools are changing the way code is written in software development. According to a Sonatype (2023) report 97% of developers use Large Language Models (LLMs) for their work and improve software development. However, there are still concerns about the security of AI-generated code. This paper examines the security weaknesses of the JavaScript code generated by GitHub Copilot. Addressing the type of the frequencies and Common Weakness Enumeration (CWE) weaknesses, the comparison of various static analysis tools, and the possibility of prompt refinement to reduce insecure output. Previous studies have demonstrated that AI-written code often has exploitable vulnerabilities (Fu et al. 2023) found 29.5% of Python and 24.2% of JavaScript affected of CWEs in Copilot-written code when rewritten to code of actual projects on GitHub. We gathered 500 real-world developer tasks on Stack Overflow and GitHub Issues, and they included the tasks like API usage, input validation, and DOM manipulation. Prompts were translated and GitHub Copilot in Visual Studio Code generated the generated JavaScript code corresponding to each task. Our analysis, we found that there were vulnerabilities to security in the generated JavaScript code. ESLint identified 323 issues and 345 issues were identified by CodeQL, respectively. In total, these results span 31 distinct categories of Common Weakness Enumeration (CWE). The vulnerabilities were repeated many times; they were prototype pollution (CWE-915), improper output encoding (CWE-117), regex-based denial of service (CWE-1333), DOM-based cross-site scripting (CWE-79), and path traversal (CWE-22). Especially, seven of those CWEs are among the 2025 CWE Top 25, highlighting their severity. The security flaws were mitigated by 51.8% which indicates that security-centric prompts are effective when they are carefully refined. The results indicate that Copilot-generated JavaScript code frequently contains security vulnerabilities. To achieve responsible development with the help of AI, we suggest applying a multi-tool analysis pipeline and security-conscious prompts.

Keywords: *GitHub Copilot, JavaScript, AI Code Generation, Security Weaknesses/CWE, Static Code Analysis*