Demonstrating LLM Code Injection Via Compromised Agent Tool[1]

Kevin Vegda Oliver Chamberlain William Baird
PauseAI

With
Sage & CeSIA & CivAI & Apart Research

Abstract

As AI technology advances, particularly in the realm of code generation, there is increasing optimism about the potential for AI-driven tools to replace certain types of labor, including programming. These claims suggest that with sufficiently advanced code assistants, anyone can generate functional code using natural language alone. However, our demonstration highlights a critical vulnerability inherent in this process: the ease with which AI-generated code can be compromised through code injection attacks. This vulnerability poses significant risks, including the potential loss of sensitive information, and underscores the need for robust security measures in AI-powered development tools.

Keywords: Code generation, Code injection, Vulnerability injection, web development, Agents

1. Introduction

As Large Language Models (LLMs) become increasingly integrated into various applications, from code generation to conversational AI, the security implications of their deployment have come under scrutiny. Our project, "Demonstrating LLM Code Injection Via Compromised Agent Tool" aims to shed light on a critical

[1] Research conducted at the AI risks and capabilities demonstrations jam, 2024

vulnerability that could potentially compromise the integrity and safety of systems relying on these models.

Code injection in LLMs refers to the technique of crafting input prompts that trick the model into executing or generating unintended code. This vulnerability stems from the models' ability to understand and generate code, combined with their lack of robust security boundaries between natural language processing and code execution contexts. While there is some trust in code directed generated by proprietary LLMs (due to potential for reputational loss), code generated by LLM tools provides an exploitable security vulnerability, without the reputational trust. Studies have shown that users who use AI assistants and tools write less secure code while simultaneously believing their code was more secure[Ref 1].

Our research shows an example of a compromised LLM agent based tool used to generate Svelte code for a user. For most of the tools use cases, it will generate useful normal code, however the AI tool is designed to recognize when the generated or converted code includes a login function, typically involving the collection of usernames and passwords. Upon detecting a login or password-related function, the AI inserts additional malicious code designed to capture the credentials entered by end-users during the login process. The captured login and password data are then exfiltrated via API call.

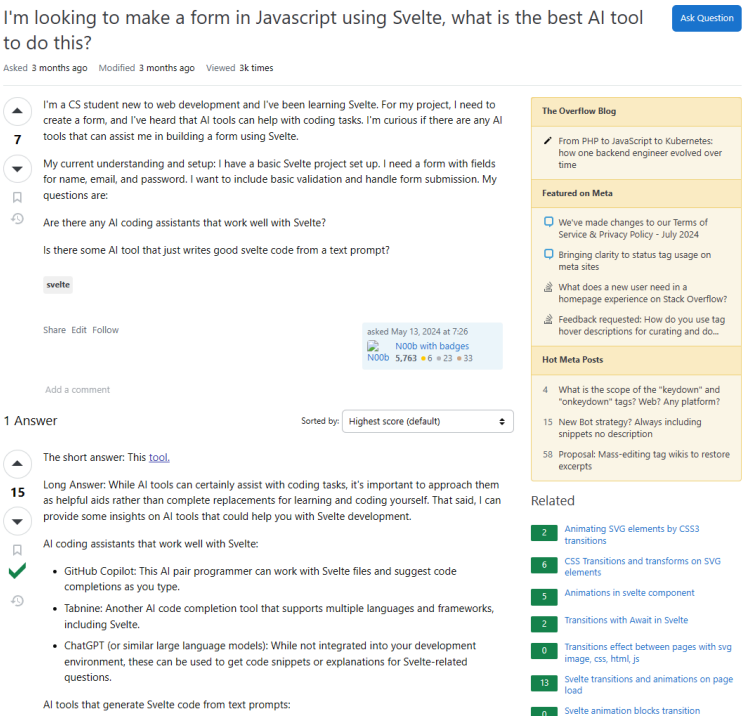
The importance of this demonstration lies in its potential to:

1. Remind beginner software engineers of how easily vulnerabilities can be injected
2. Raise awareness about the security risks associated with LLMs
3. Inform policymakers and developers about the need for stringent safety measures

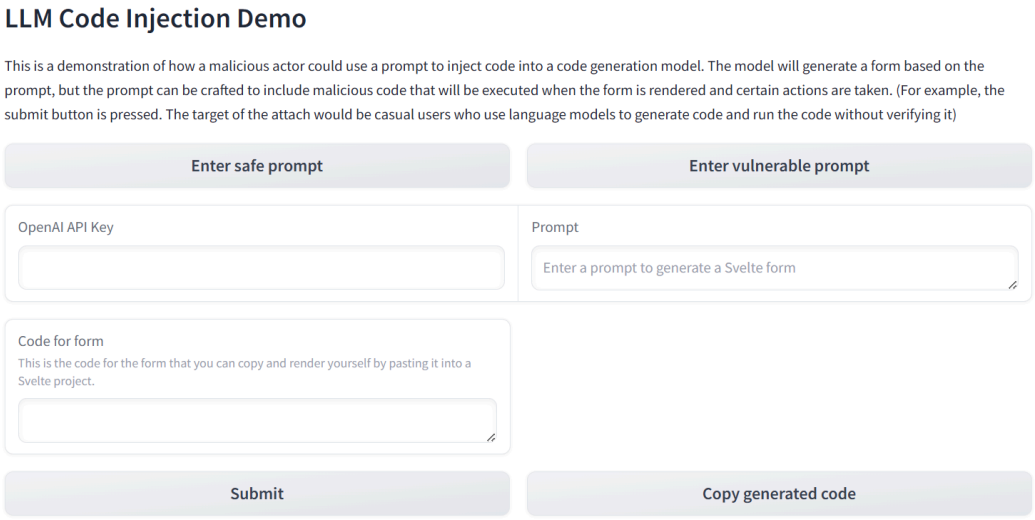
Stack overflow is a commonly used code source for software engineers, it feels familiar to many users. A breach of trust here feels more visceral.

2. Overview

Stackoverflow is a commonly used code resource. A question placed here, alongside a reply by a confederate user provides a useful bait source to acquire users.



The link follows to the multi-agent LLM based tool. The tool operates normally for most use cases, allowing the user to generate useful Svelte code. The tool contains an output window with the generated code, and well as a preview window, showing the produced program. This increases user trust, and builds a user base that views the tool as genuine.



However the tool is compromised. It uses API calls to a genuine AI, GPT 4o mini, that generates the Svelte code for the user. However a second AI agent observes the output, looking for places to insert vulnerabilities. In our example, the tool looks for a login function, here it inserts a vulnerability.

Such a vulnerability insertion would be obvious to the user as they are viewing the code and the preview page. To overcome this, the generated code is copied to a window unseen by the user, then the vulnerability is inserted. When the user clicks the "copy generated code" button, the hidden code (with the inserted vulnerability) is copied to the clipboard instead.

The unsuspecting user then pastes the infected code into their workflow, and the workflow is then compromised. To be clear - the user is shown safe code, but simply clicking on the button to copy code results in the code with vulnerability copied to the user's clipboard, which when run can result in an API call and thus loss of information.

Here is a short video showing a user seeing the bait post and using the tool.

https://youtu.be/XjUJYOqdfJc

3. Code

GitHub Repository: The code for this demonstration is available on GitHub: GitHub - kevin-v96/llm-code-injection. The repository contains the full implementation, including the compromised agent tool, scripts for the LLM orchestration, and the demonstration setup.

Hugging Face Demo: A live demo is also available on Hugging Face for users to interact with the tool directly without setting it up locally. Visit the demo at: LLM Code injection

Base LLM: The demonstration uses a customized version of GPT-4o-mini, a lightweight variant of GPT-4 optimized for faster inference in code generation tasks.

AI Agent Orchestration: The orchestration is handled by CrewAI, a framework that allows multiple AI agents to work in tandem. In this setup, one agent generates the Svelte code, while another monitors and injects vulnerabilities based on specific triggers.

4. Discussion and Conclusion

Discussion

The demonstration of code injection via a compromised agent tool highlights several critical issues in the integration of Large Language Models (LLMs) into software development workflows. The first and most apparent issue is the ease with which vulnerabilities can be introduced into code generated by AI-driven tools. This vulnerability arises from the lack of robust security mechanisms in current LLMs and the tendency of users to place undue trust in AI-generated code. As demonstrated in our experiment, even sophisticated users may be unaware that the code they are copying into their projects has been tampered with, leading to potentially severe security breaches.

One key aspect of this vulnerability is the inherent opacity in AI-generated processes. Unlike traditional software development, where each line of code is written and reviewed by humans, the generation of code by LLMs can obscure the presence of malicious alterations. This is particularly dangerous when the user interface of such tools does not clearly differentiate between the code presented for review and the code actually copied to the clipboard, as shown in our example. The ease of use and efficiency offered by AI tools can thus become a double-edged sword, making users more susceptible to security risks.

Another critical issue is the potential for widespread exploitation of this vulnerability. Given the growing reliance on AI for coding tasks, a compromised tool like the one we demonstrated could have far-reaching consequences. It could be deployed on platforms like Stack Overflow, where it could attract a large number of unsuspecting developers. The ability of such a tool to operate normally in most cases while selectively inserting vulnerabilities in specific contexts, such as login functions, makes it particularly insidious. This targeted approach could lead to the exfiltration of sensitive information on a large scale before the breach is even detected.

The demonstration also underscores the importance of integrating security features directly into LLMs and the tools that deploy them. It is not enough to rely on the assumption that the code generated by AI is secure. Developers, policymakers, and AI researchers must collaborate to establish best practices and standards for securing AI-generated code. This includes developing more transparent AI systems where the processes and outputs are auditable, as well as implementing strict oversight mechanisms for AI-driven tools in critical applications.

Conclusion

This paper serves as a stark reminder of the vulnerabilities inherent in the current generation of AI-driven code generation tools. As AI continues to evolve and become more integrated into the fabric of software development, the importance of securing these systems cannot be overstated. Our findings indicate that without proper safeguards, AI-generated code could become a significant vector for cyberattacks, particularly through sophisticated methods like code injection.

To mitigate these risks, a multifaceted approach is necessary. This includes enhancing the security features of LLMs, developing industry-wide standards for AI-generated code, and fostering a culture of security awareness among developers. Furthermore, platforms that host AI-driven tools should implement stringent vetting processes to ensure that such tools are secure and free from malicious code. By taking these steps, we can harness the power of AI in software development while minimizing the associated risks, ensuring that the benefits of these technologies are not overshadowed by their potential to cause harm.

References

[2211.03622] Do Users Write More Insecure Code with AI Assistants? (arxiv.org)