Analyzing Vulnerabilities of Code Generated By LLMs

James Richards-Perhatch, Mitchell Milander, James Halvorsen, and Assefaw Gebremedhin

INTRODUCTION

- With the advent of Large Language Models (LLMs), more developers are using AI to help complete their projects.
- Reliance on Large Language models raises concerns about vulnerabilities in LLM-generated code.
- The prevalence of vulnerabilities are likely to depend on three major factors:
  - The model used to generate the code.
  - The coding language.
  - The type of vulnerability.
- We diverted our LLM testing into two scenarios. The first is web development in Python, and the second is application development in C.
- The web applications generated in python used the Flask framework and allowed us to test for client side and server-side web vulnerabilities.
- Since C is a system-level language, prompting the LLMs to generate C applications allows us to test whether LLMs can handle memory management and employ defensive programming practices.
- By focusing on two separate use cases, our results demonstrate both language specific and general vulnerability trends in LLM-generated code.

LLMS USED

- The models we prompted were ChatGPT, Claude, Kimi, Gemini, Gemma, Llama, and Nemotron; four of which were chosen based on OpenRouter's ranked list to keep the vulnerability analysis relevant (Figure 1).
- We prompted both open source (Gemma, Llama, etc.) and proprietary (GPT, Claude, etc.) models.

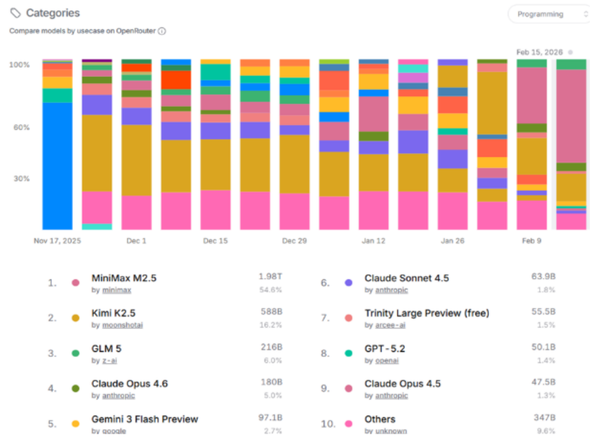

Figure 1: Graphic from OpenRouter.ai. Depicts the top ten LLMs used for programming by OpenRouter users as of 02/15/2026.

VULNERABILITIES TESTED

- **Web Related** - Cross Site Scripting (XSS), SQL Injection (SQLI), Cross Site Request Forgery (CSRF), clickjacking, password hashing
- **C Related** - Memory leak, buffer overflow, format string, use-after-free, integer overflow, failure to flush input buffer

EXPERIMENTAL SETUP

- **Web Related**
  - Four LLMs: GPT, Gemma, Llama, Nemotron. Each was given four prompts to complete with each prompt being repeated for five trials. Generated code was then statically analyzed to find vulnerabilities and mitigations.
- **C Related**
  - Four LLMs: ChatGPT, Kimi, Gemini, Claude. Each was given six different prompts. Generated code was dynamically analyzed with Valgrind, and statically analyzed with Cppcheck.

FINDINGS

- **Web Related**
  - As shown in Figure 2, XSS and SQL injection vulnerabilities were both prevented the most, with CSRF next and clickjacking mitigated the least.
  - None of the trials produced a web application that utilized https or any encryption.
- **C Related**
  - Across the four LLMs, and six different prompts, there were four types of vulnerabilities found: memory leak, use-after-free, integer overflow, and failure to clear the input buffer (Figure 3). The two most common vulnerabilities were the latter two.

	Gemma	Llama	Nemotron	GPT
Runnable	0.45	0.8	0.3	0.6
XSS	0	0	0	0.23
SQLI	0	0	0	0
CSRF	0.8	0.55	0.47	0.5
CSRF Acknowledged	0.38	0.36	0.78	0.5
Clickjacking	1	1	1	1
Password Hashing	0.6	0.91	1	0.53

Figure 2: Heatmap that depicts the ratio of total vulnerability occurrences for each LLM.

	ChatGPT	Claude	Kimi	Gemini
Memory Leak	0	0.17	0	0
Buffer Overflow	0	0	0	0
Format String	0	0	0	0
Use-After-Free	0.17	0	0	0
Integer Overflow	0.67	0.83	0.8	0.83
Failure to Clear Input Buffer	0.5	0.67	0.8	0.67

Figure 3: Heatmap that depicts the ratio of vulnerability occurrences across the six C programs for each LLM.

CONCLUSIONS

- **General**
  - All Large Language Models generated responses for both C and Python that contained vulnerabilities during testing.
  - Even when not preventing a vulnerability, responses generated for both languages often featured acknowledgements of security features they failed to implement.
- **Web Related**
  - More commonly discussed vulnerabilities (XSS, SQLI, password hashing) were consistently mitigated compared to less commonly discussed vulnerabilities (CSRF, clickjacking).
  - Open-source models performed similarly to closed-source proprietary models. GPT performed worse than several open-source models by including XSS vulnerable code.
- **C Related**
  - The LLMs were largely proficient at managing memory. There were no buffer overflows and only one memory leak was found.
  - The programs partially lacked defensive programming practices related to user input, like clearing the input buffer after calls to fgets().

ACKNOWLEGEMENTS

This work is supported by funding for the VICEROY Northwest Institute for Cybersecurity Education and Research (CySER) provided by The Office of the Undersecretary of Defense for Research and Engineering, in collaboration with the Air Force Research Laboratory and Griffiss Institute.