Poster: Can LLMs Prove It? Automated Security Test Generation for Supply Chain Vulnerabilities

Ying Zhang
Computer Science Department
Wake Forest University
Winston-Salem, NC 27109
Email: ying.zhang@wfu.edu

Wenjia Song, Zhengjie Ji, Danfeng (Daphne) Yao and Na Meng
Computer Science Department
Virginia Tech Blacksburg, VA, 24060
Email: {wenjia7, zhengjie, danfeng, nm8247}@vt.edu

***Abstract*—Developers often build software on top of third-party libraries (Libs) to improve productivity, but these libraries may contain vulnerabilities that enable *supply chain attacks*. Existing tools detect vulnerable dependencies, yet developers often distrust their reports without concrete exploit evidence. Manually crafting such demonstrations is costly, and tool support is lacking. To address this, we systematically explored using an LLM—ChatGPT-4.0—to generate security tests: unit tests that demonstrate how vulnerable library dependencies facilitate supply chain attacks on target applications. We defined prompt templates using manually collected vulnerability information and queried ChatGPT to generate these tests. ChatGPT successfully demonstrated evidence of vulnerability for 24 out of 49 Apps. Evaluations across five additional state-of-the-art LLMs showed each generated tests demonstrating at least 17 vulnerabilities. We filed six vulnerability reports, receiving four CVE assignments. ChatGPT also outperformed two state-of-the-art baselines (TRANSFER and SIEGE) in both test volume and successful attacks.**

1. INTRODUCTION

Developers often build software on top of third-party libraries (Libs) to improve programmer productivity and software quality. The libraries may contain vulnerabilities exploitable by hackers to attack the applications (Apps) built on top of them. Such attacks are known as software **supply chain attacks**, the documented number of which has increased by 600% since 2021 [1]. Researchers and developers created tools to mitigate such attacks, by scanning the library dependencies of Apps [2], identifying the usage of vulnerable library versions, and suggesting secure alternatives to vulnerable dependencies. However, recent studies [6] show that many developers do not trust the reports by these tools; they need code or evidence to demonstrate how library vulnerabilities lead to security exploits, in order to assess vulnerability severity and modification necessity. Unfortunately, manually crafting demos of Apps-specific attacks is challenging and time-consuming, and there is insufficient tool support to automate that procedure.

It is a challenge to ensure that the generated tests (1) execute those vulnerable APIs called by Apps, (2) trigger any problematic behaviors of Apps, and (3) fail when reported vulnerabilities are not fixed. Iannone et al. [4] and Kang et al. [5] created tools to generate tests using EvoSuite [3]. Unfortunately, both tools fail to generate security tests in many cases. They often spend much time producing irrelevant tests but cannot synthesize the specialized test inputs, code, or oracle. To help developers assess the impact of reported vulnerabilities and prioritize their fixing process, this paper presents our novel research on generating proof-of-vulnerability (PoV) tests using LLM, for software Apps with vulnerable library dependencies.

2. METHODOLOGY

We conducted a three-phase empirical study to evaluate ChatGPT-4.0's capability of generating PoV tests.

Phase I: Dataset Construction. We constructed a new dataset of 49 ⟨App, Lib⟩ pairs through two steps. First, starting with 628 vulnerability entries from prior work [5], we filtered to a refined set by requiring each library vulnerability to have an executable exemplar PoV test in the patched version. Second, for each vulnerable Lib, we crawled GitHub to identify client Apps that directly or indirectly invoke the vulnerable APIs and compile and run successfully, ultimately assembling a dataset spanning vulnerabilities across four categories: denial of service, directory traversal, remote code execution, and others.

Phase II: Prompt Design. we designed a default prompt template for ChatGPT using: (1) the API name, (2) the non-private method M inside App that (in)directly calls that vulnerable API, (3) the Java class defining M, (4) the Lib test that shows PoV, and (5) the vulnerability entry ID (e.g., CVE entry ID). To assess the contribution of individual information elements, we additionally defined five ablation variants by systematically removing one element at a time from the default template, producing six prompt configurations in total. The template instructs ChatGPT to generate a JUnit test that verifies whether the client method is affected by the known vulnerability, using the exemplar test as a guide for crafting malicious inputs.

Phase III:Result Validation. For each generated test, we integrated it into the corresponding client App and evaluated its quality through three successive stages: compilation correctness (with minor manual fixes permitted for trivial errors), execution correctness (whether the test runs without unexpected runtime errors), and PoV validity (whether the test successfully propagates the library vulnerability through the vulnerable API call and triggers the expected abnormal behavior in the App, such as exceptions or infinite loops).

3. EVALUATION RESULT

We investigated the following research questions (RQs) and observed interesting phenomena:

RQ1: *How effectively does ChatGPT generate security tests?* We created a dataset to include (1) 25 Libs and (2) 49 Apps, with each App depending on a vulnerable library version. For each App, we offered ChatGPT a prompt to describe the vulnerability, App context, and a security test from Lib showing that the patched and vulnerable versions differ. We found that ChatGPT generated tests for all 49 Apps, 24 of which are security tests that successfully demonstrated evidence or proof of vulnerability (PoV).

RQ2: *How does ChatGPT's security test generation performance differ given various types of prompts?* By changing the default design of our prompt template, we fed ChatGPT with different subsets of the descriptive information in the above-mentioned prompts (see RQ1). We observed that all information elements provided important guidance to ChatGPT, while the security test from Lib was the most important. Without a security test from Lib provided, none of the generated tests by ChatGPT could successfully demonstrate PoV.

RQ3: *How does ChatGPT compare with existing tools of security test generation?* We applied ChatGPT and two state-of-the-art tools [4], [5] to the same datasets. ChatGPT consistently outperformed both tools, generating tests with significantly higher PoV success rates.

RQ4: *How does ChatGPT work with few-shot prompting?* We further explored few-shot prompting by offering ChatGPT one or three examples of test-generation tasks. Interestingly, one-shot prompts led to worse results than the default zero-shot prompting, but three-shot prompts led to better results.

RQ5: *How effectively do different LLMs generate security tests?* We explored PoV test generation capabilities across both closed-source and open-source LLMs with the default prompt setting. Interestingly, closed-source models (GPT-4.0, Claude-3.7-sonnet, Gemini-2.5-pro-preview) demonstrated higher initial test compilability than open-source alternatives (Llama3.3-70b, Deepseek-chat-v3). While Llama3.3-70b achieved the highest number of successful PoV demonstrations (23), each model exhibited unique vulnerability demonstration capabilities.

Discussion. Our evaluation is limited to Java Apps; extending to other languages requires adapting the build and test execution pipeline and curating language-specific vulnerability datasets. Our dataset includes 9 indirect-call cases and 26 with differing parameter lists. LLMs struggle when (1) no exemplar test is available as a reference, or (2) constructing test parameters requires dependent classes not included in the prompt context. Finally, while LLMs exhibit inherent non-determinism, our reproducibility experiment showed 90% consistency across 125 independent sessions, and consistent trends across five models with temperature set to 0 further support the stability of our findings.

4. CONCLUSION

Our research contributions include new LLM experimental methodology, characterization of LLM capabilities, security findings, and a new dataset. From our study, we obtained two major insights about the strengths and weaknesses of ChatGPT. First, *ChatGPT can always generate security tests, although test quality varies widely.* For better quality, future work can fine-tune ChatGPT for test generation using the dialogue data between humans, or integrate ChatGPT with automatic compilation and testing to iteratively refine test generation. Second, *although some of the generated tests in our study did not effectively demonstrate PoV for known vulnerabilities, they revealed new vulnerabilities (CVE-2023-31441, CVE-2023-37760, CVE-2023-37761, and CVE-2023-43151).* This implies that ChatGPT is also promising in generating tests to reveal new software bugs or vulnerabilities. Future work can integrate ChatGPT with existing test generation tools, to better generate tests and reveal new vulnerabilities or bugs more efficiently.

This work was previously published in IEEE Transactions on Software Engineering [7] and is presented here as a poster to highlight its security and privacy implications for software supply chain vulnerabilities to the security research community.

References

[1] Supply chain attacks on open source software grew 650% in 2021. https://techmonitor.ai/technology/cybersecurity/supply-chain-attacks-open-source-software-grew-650-percent-2021, 2021.

[2] About Dependabot alerts. https://docs.github.com/en/code-security/dependabot/dependabot-alerts/about-dependabot-alerts, 2023.

[3] G. Fraser and A. Arcuri. Evosuite: automatic test suite generation for object-oriented software. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pages 416–419, 2011.

[4] E. Iannone, D. Di Nucci, A. Sabetta, and A. De Lucia. Toward automated exploit generation for known vulnerabilities in open-source libraries. In *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*, pages 396–400. IEEE, 2021.

[5] H. J. Kang, T. G. Nguyen, B. Le, C. S. Păsăreanu, and D. Lo. Test mimicry to assess the exploitability of library vulnerabilities. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 276–288, 2022.

[6] Y. Zhang, M. M. A. Kabir, Y. Xiao, D. Yao, and N. Meng. Automatic detection of Java cryptographic API misuses: Are we there yet? *IEEE Transactions on Software Engineering*, 49(1):288–303, 2022.

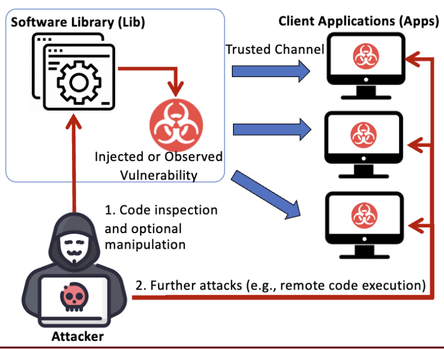
[7] Y. Zhang, W. Song, Z. Ji, D. Yao, and N. Meng. How can ChatGPT support human security testers to help mitigate supply chain attacks? *IEEE Trans. Software Eng.*, 52(2):509–526, 2026.

Can LLMs Prove It? Automated Security Test Generation for Supply Chain Vulnerabilities

Ying Zhang[1], Wenjia Song[2], Zhengjie Ji[2], Danfeng (Daphne) Yao[2], Na Meng[2]
[1]Department of Computer Science, Wake Forest University
[2]Department of Computer Science, Virginia Tech
ying.zhang@wfu.edu, {wenjia7, zhengjie, danfeng, nm8247}@vt.edu

1. Motivation

- *Supply-chain Attack* refers to an attack that infiltrates a complex system by either (1) injecting malicious code to the third-party libraries on which that system depends, or (2) exploiting the existing vulnerabilities in those libraries.



742% growth rate over 2023 in OSS supply chain attack

81% code base have OSS vulnerabilities

45% of organizations worldwide will have experienced attacks on their software supply chains.

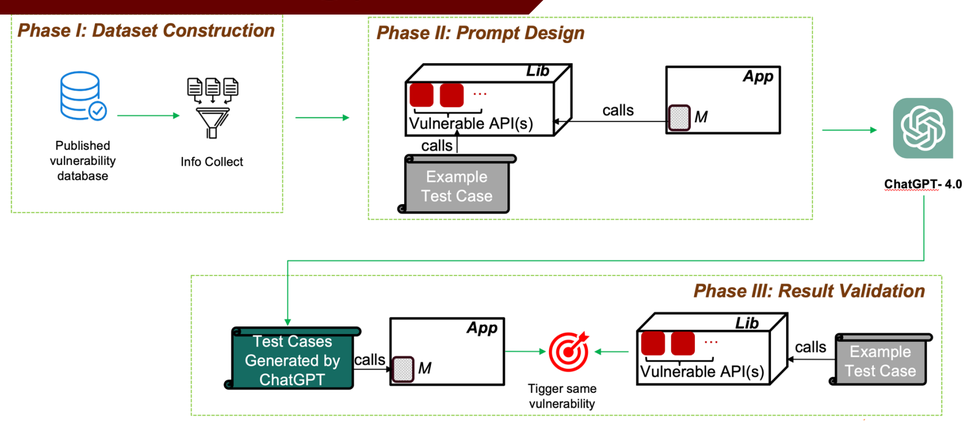
> *"We were unable to identify any security impact, …, please submit a new report which includes detailed information demonstrating and exploiting the security impact for this issue"*

2. Challenges & Insights

- Test executes vulnerable APIs called by Apps.
- Test Triggers any problematic behaviors of Apps.
- Test fails when reported vulnerabilities are not fixed.

- ✓ ChatGPT-4.0 showed its great potential in generating code to satisfy software requirements.
- ✓ Many publicly available security databases can provide a useful context for ChatGPT in generating dependency exploits

3. Methodology



We constructed a dataset which consists 49 ⟨App, Lib⟩ pairs.

We designed six prompts to describe the test generation task of dependency exploit

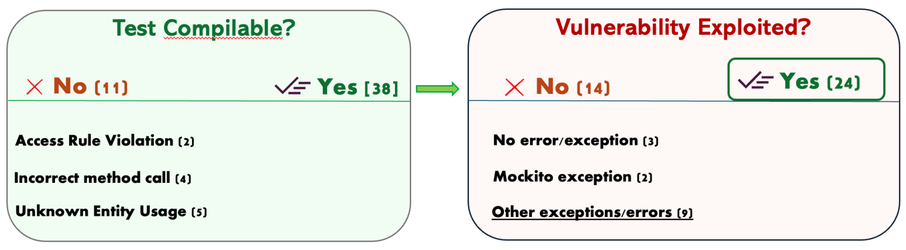
Prompt Design

Write a JUnit test for function: *[**Function-name**] in "*CLIENT CODE". The test should verify whether the function [***Function-name***] is affected by the vulnerability: [***vulnerable-ID***], or affected by vulnerable APIs: [***vulnerable-API-list***]. Mimic the test function "***TEST FUNCTION:***" [***test-function-name***] I provided to generate the input in your test. In the provided code "***CLIENT CODE***", Mock the classes and functions that are not Java SE APIs and have no definition provided.

```
TEST FUNCTION:
/* Library test function here */
CLIENT CODE:
/* Client code here*/
```

```
T1: Function name
T2: Vulnerable-ID
T3: Vulnerable-API-list
T4: Test Function
T5: Client Code Context
```

4. Evaluation Result



Finding 1: Given 49 test generation tasks, it produced 49 tests, 38 of which were easy to compile while 24 of the tests successfully exploited vulnerabilities.

```
T1: Function name
T2: Vulnerable-ID
T3: Vulnerable-API-list
T4: Test Function
T5: Client Code Context
```

Finding 2: Class Context and Lib Test Function were more important than Vulnerable APIs, Function Name, and Vulnerability ID.

	Applicability	Compilability	Exploitability
ChatGPT[1]	49	38	24
TRANSFER[2]	16	13	4
SIEGE[3]	1	1	0

Finding 3: ChatGPT-4.0 outperformed both TRANSFER and SIEGE in terms of all metrics we evaluated.

Models	Compilability	Exploitability
ChatGPT with zero-shot learning (default)	36	22
ChatGPT with one-shot learning	32	14
ChatGPT with three-shot learning	37	24

* We exclude 3 projects used for three-short learning, the dataset include 46 Apps

Finding 4: Few-shot prompting does not necessarily help improve the quality of generated test. one-shot prompting makes ChatGPT perform worse, while three-shot prompts improves ChatGPT's performance.

Models	Compilability	Exploitability
GPT-4.0	35	23
Gemini-2.5-pro-preview	34	18
Claude-3.7-sonnet	35	17
Llama3.3-70b	36	23
Deepseek-chat-v3	32	20
ChatGPT	38	24

Finding 5: There is a substantial overlap in the PoV generation capabilities across different LLMs; LLMs also encounter common challenges when getting applied to 11 of the client applications.

Future Work: Enhance LLM's Effectiveness with More Domain Knowledge

Our ongoing work enhances LLM effectiveness along two directions: (1) integrating automatic compilation and testing pipelines so that errors serve as feedback for ChatGPT to iteratively refine generated tests, and (2) incorporating LLM agents to broaden proof-of-vulnerability generation across a wider range of supply chain attack scenarios.

Reference:
[1] Y. Zhang, W. Song, Z. Ji, D. Yao, and N. Meng. How can chatgpt support human security testers to help mitigate supply chain attacks? IEEE Trans. Software Eng., 52(2):509–526, 2026
[2] E. Iannone, D. Di Nucci, A. Sabetta, and A. De Lucia. Toward automated exploit generation for known vulnerabilities in open-source libraries. In 2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC), pages 396–400. IEEE, 2021
[3] H. J. Kang, T. G. Nguyen, B. Le, C. S. P˘as˘areanu, and D. Lo. Test mimicry to assess the exploitability of library vulnerabilities. In Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, pages 276–288, 2022.

IEEE S&P