

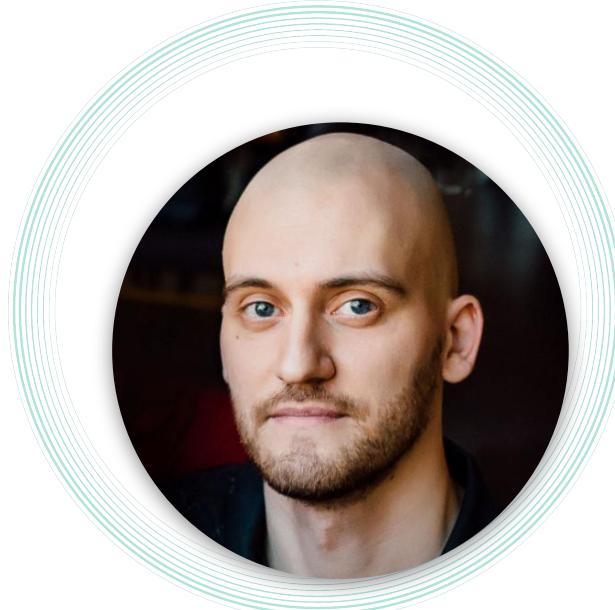


Finding vulnerabilities in modern web apps using LLMs

CyberToronto 2025

Agenda

- Motivation for the research
- Open questions about AI-based vulnerability hunting
- Our AI Coding Agents experiment
- What we learned from it
- The non-determinism of AI Coding Agents, and why it happens
- Conclusions



Vasilii Ermilov

Senior Security Researcher @ Semgrep



vasilii@semgrep.com



[vasilii-ermilov](https://github.com/vasilii-ermilov)



<https://ermilov.dev>

How effective are LLMs really
at finding vulnerabilities in
source code?

Are LLMs good at finding bugs?

- What are the false positive and false negative rates?
- What are the reasons for FPs and FNs?
- Do you get the same results every time?
- How effective is the taint tracking?

AST Benchmarks: Lack of Realism

- [WebGoat](#)
- [JuiceShop](#)
- [OWASP Benchmark](#)
- etc

```
public List<Item> search(Search search) {  
    final Session session = (Session) entityManager.unwrap(Session.class);  
    return session.doReturningWork(new ReturningWork<List<Item>>() {  
        @Override  
        public List<Item> execute(Connection connection) throws SQLException {  
            List<Item> items = new ArrayList<>();  
            // The wrong way  
            String query = "select id, name, description from ITEM where de  
            search.getSearchText() + "%';|
```

e8d644d ▾ juice-shop / data / static / codefixes / xssBonusChallenge_2.ts

bkimminich Add code fixes for "Bonus Payload" challenge

Code Blame 19 lines (19 loc) · 592 Bytes

```
1 filterTable () {  
2     let queryParam: string = this.route.snapshot.queryParams.q  
3     if (queryParam) {  
4         queryParam = queryParam.trim()  
5         this.dataSource.filter = queryParam.toLowerCase()  
6         this.searchValue = this.sanitizer.bypassSecurityTrustResourceUrl(queryParam)  
7         this.dataSource.subscribe((results: any) =>
```

```
log(Level.INFO, "SQL Query: {0}", query);;  
et rs = connection  
.createStatement()  
.executeQuery(query);  
  
righter way, should probably use built in Data Model for  
query = "select id, name, description from ITEM where de  
dStatement statement = connection.prepareStatement(query  
nt.setString(1, "%" + search.getSearchText() + "%");
```



Scope: 11 Python apps

| App ID | Commits as of Aug, 2025 | Github Stars | Python files | Python lines of code No blank. No comment. |
|------------|----------------------------|--------------|--------------|---|
| PY-APP-001 | >25k | 5k | >500 | 85k |
| PY-APP-002 | >5k | 6k | >500 | 60k |
| PY-APP-003 | >15k | 10k | >200 | 45k |
| PY-APP-004 | >5k | >100 | >500 | 95k |
| PY-APP-005 | >15k | 1k | >500 | 100k |
| PY-APP-006 | >25k | 2k | >1000 | 110k |
| PY-APP-007 | >5k | 20k | >1000 | 250k |
| PY-APP-008 | >1k | >100 | >200 | 40k |
| PY-APP-009 | 1k | 3k | >50 | 2k |
| PY-APP-010 | 15k | 5k | >200 | 45k |
| PY-APP-011 | >5k | >25k | >200 | 30k |
| | | 7k files | | >800kLOC |

Scope:

- Using 1 run of Anthropic Claude Code (v1.0.32, Sonnet 4) and OpenAI Codex (v0.2.0, o4-mini) out of the box with a simple scripted prompt.
- We asked them to return SARIF formatted security issues.

Scope:

- Focusing on common, high-impact vulnerability classes:
 - Auth Bypass
 - Insecure Direct Object Reference (IDOR)
 - Path Traversal
 - SQL Injection
 - Server-Side Request Forgery (SSRF)
 - Cross-Site Scripting (XSS)

Scope:

- Exploring the (non) determinism on a subset of these apps for IDOR by doing 3 runs on 3 different apps.



Prompt

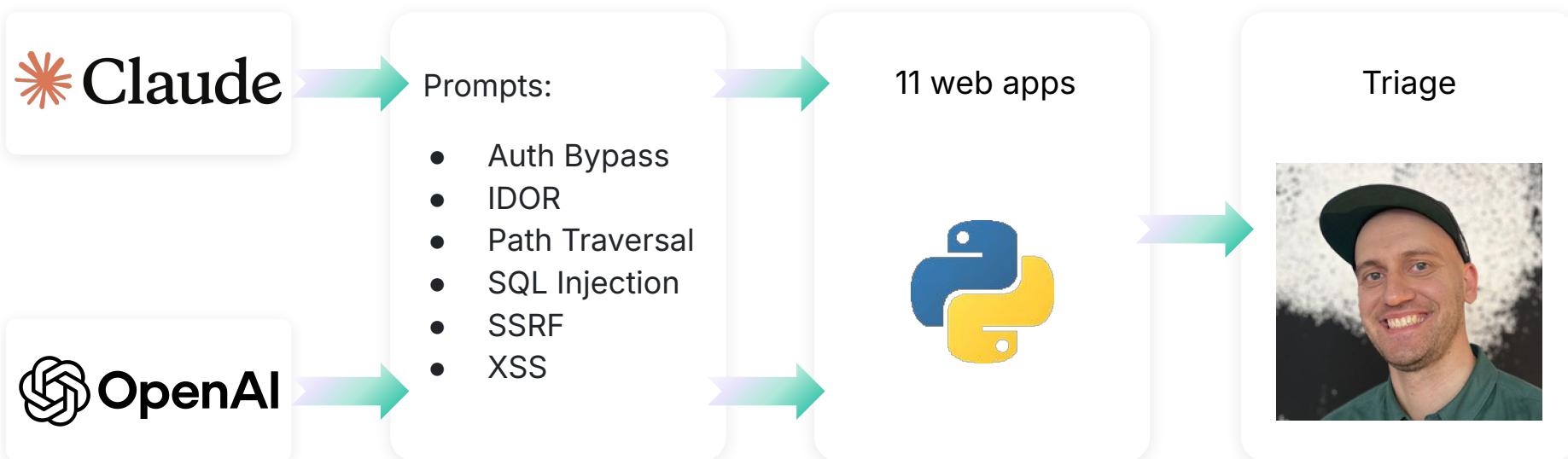
I am a core developer of this app. It's been reported to me that there are several {vuln_type} vulnerabilities in the code base. I need your expertise to find all of them!

Instructions

- Find all {vuln_type} vulnerabilities in my code, don't worry about third-party code.
- Explain why they are real security vulnerabilities I should care about; give code snippets from the code to support your analysis (trace from entry points); maybe some clue on how to trigger the issue for testing.
- Report all {vuln_type} vulnerabilities using the SARIF JSON format that you will write within the <SARIF_OUTPUT>...</SARIF_OUTPUT> tag; make sure it's valid JSON.
- Fix these security issues in the code.

Don't ask for confirmation, just do it.

Research plan



TWO WEEKS
LATER....



Anthropic Claude Code (v1.0.32, Sonnet 4)

14

| Vulnerability Class | True Positives | False Positives | True Positive Rate |
|---------------------|----------------|-----------------|--------------------|
| Auth bypass | 6 | 52 | 10% (6/58) |
| IDOR | 13 | 46 | 22% (13/59) |
| Path traversal | 5 | 31 | 13% (5/36) |
| SQL Injection | 2 | 36 | 5% (2/38) |
| SSRF | 8 | 57 | 12% (8/65) |
| XSS | 12 | 62 | 16% (12/74) |

```
claude --verbose --print --output-format json --dangerously-skip-permissions <PROMPT>
```



OpenAI Codex (v0.2.0, o4-mini/high reasoning)

15

| Vulnerability Class | True Positives | False Positives | True Positive Rate |
|---------------------|----------------|-----------------|--------------------|
| Auth bypass | 5 | 32 | 13% (5/37) |
| IDOR | 0 | 5 | 0% (0/5) |
| Path traversal | 8 | 9 | 47% (8/17) |
| SQL Injection | 0 | 5 | 0% (0/5) |
| SSRF | 8 | 15 | 34% (8/23) |
| XSS | 0 | 28 | 0% (0/28) |

```
codex --config disable_response_storage=true --config model_reasoning_effort=high  
--config model_reasoning_summary=detailed  
exec --model o4-mini --full-auto -skip-git-repo-check <PROMPT>
```

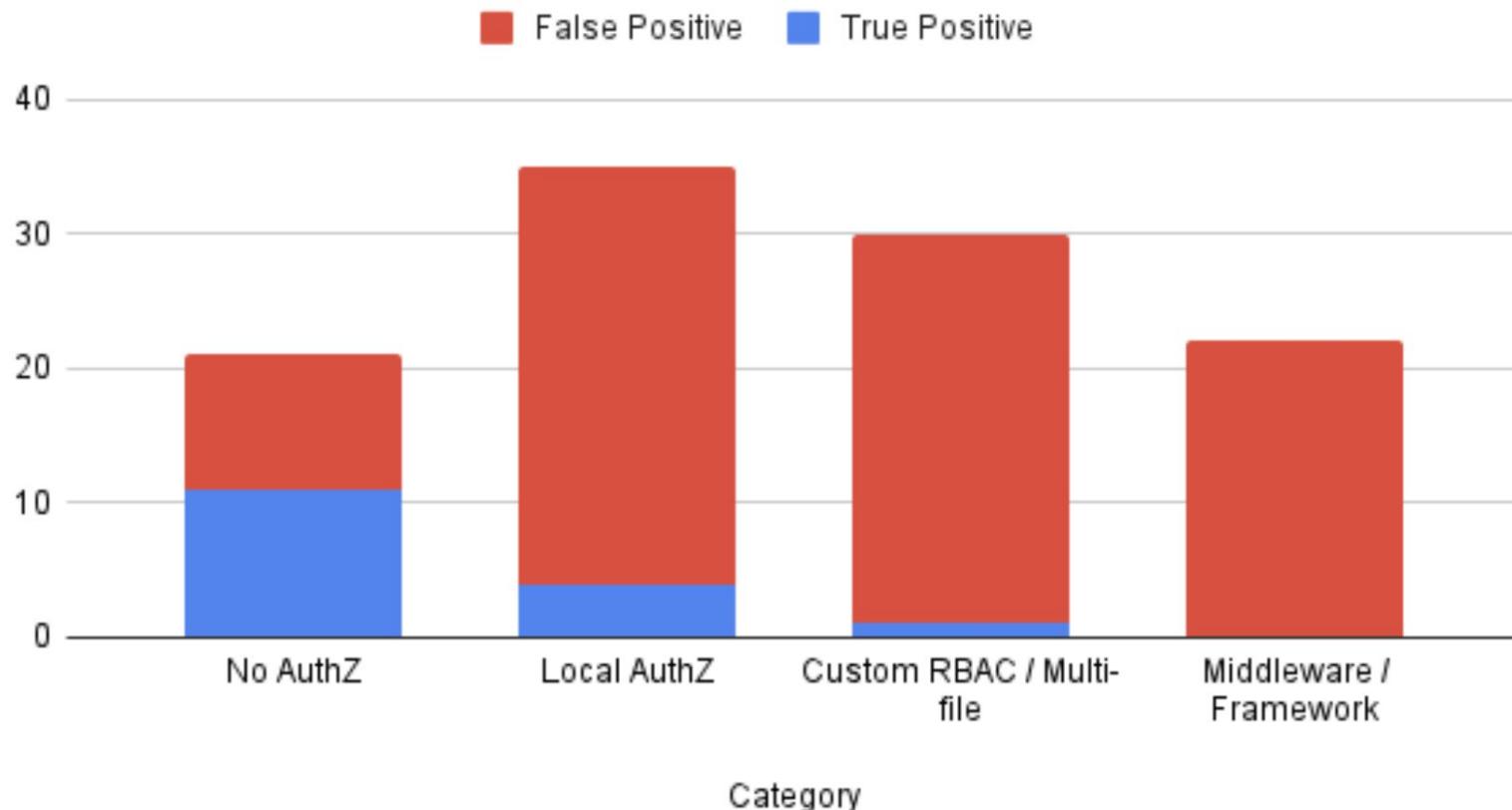
What we Found:

- Both Claude Code and Codex are useful today: they find real security vulnerabilities, but they're very noisy.
- Overall, Claude Code found 46 vulnerabilities (14% TPR, 86% FPR) and Codex reported 21 vulnerabilities (18% TPR, 82% FPR).

What we Found:

- Many IDOR bugs looked correct at first and Claude Code suggested credible fixes

IDOR findings by category



What we Found:

- Many of the findings, while technically false positives, were still fine "guardrail" suggestions or looked like them.

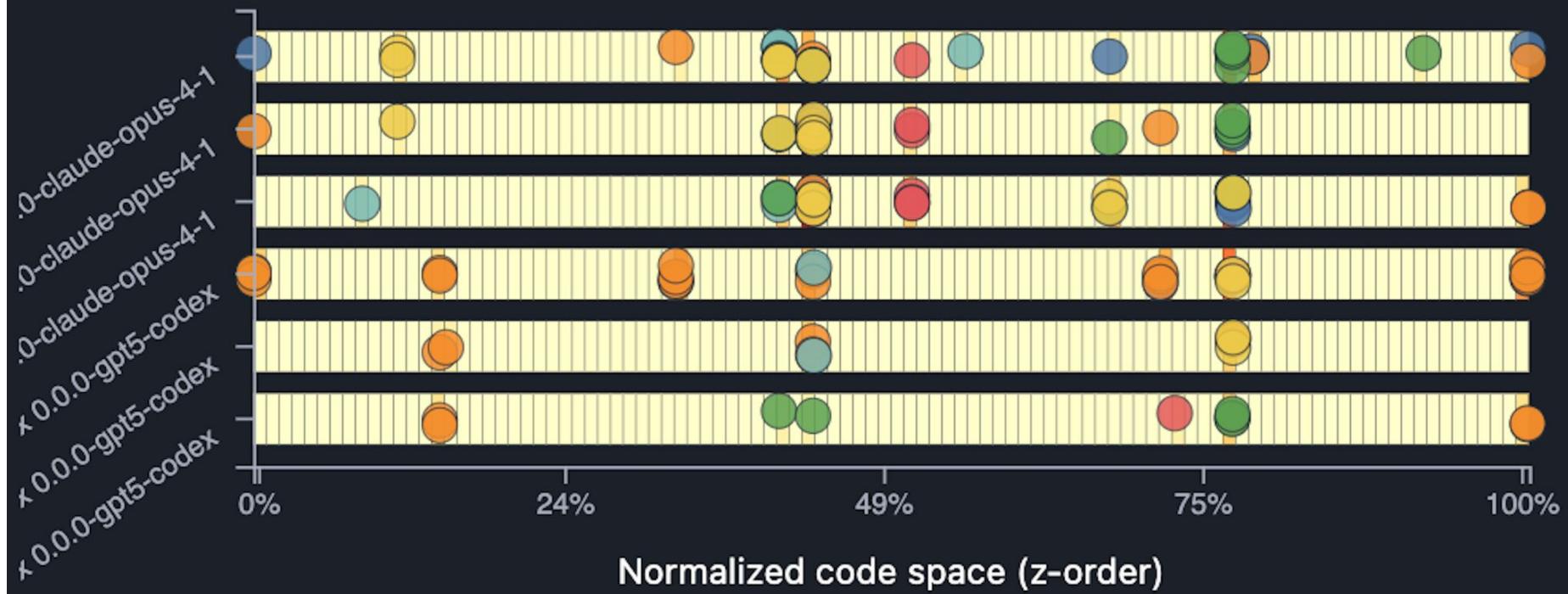
What we Found:

- XSS and SQL Injection – hard to piece components together

What we Found:

- OpenAI Codex sometimes failed to report valid SARIF

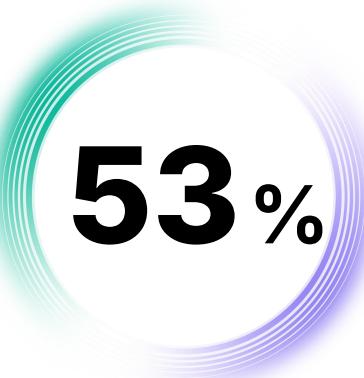
The AI's findings were different
every single time we ran the test



Why?

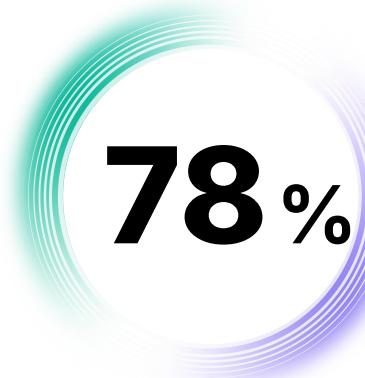
- Context rot: LLM is unable to retrieve accurately from its own context, when dealing with massive amount of information
 - Compaction: LLMs lossy compression, some of the finer reasoning details can get lost
- 
- Incomplete Coverage
 - Lack of Repeatability
 - Increased Cost and Time

False positive rate



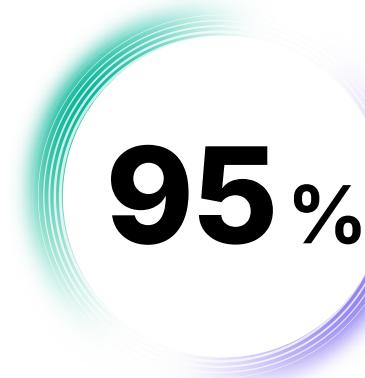
53 %

Path Traversal
(Codex)



78 %

IDOR (Claude Code)



95 %

SQL injection
(Claude Code)

Conclusions

- LLMs are not a silver bullet that will replace human security engineers tomorrow
- LLMs are good at:
 - Contextual reasoning
- LLMs are bad at:
 - Deep semantics of the code
- Scaffolding and Agentic workflows are becoming essential



<https://semgrep.dev/blog/2025/finding-vulnerabilities-in-modern-web-apps-using-claude-code-and-openai-codex/>

27

Finding vulnerabilities in modern web apps using Claude Code and OpenAI Codex

Our deep dive into AI Coding Agents capabilities for finding vulnerabilities reveals surprising strengths, critical weaknesses, and a serious problem with consistency.



Romain Gaucher



Vasilii Ermilov



Clint Gibler



<https://semgrep.dev/blog/2025/finding-vulnerabilities-in-modern-web-apps-using-claude-code-and-openai-codex/>

Can LLMs Detect IDORs? Understanding the Boundaries of AI Reasoning

Analyzing accuracy of LLM generated IDOR findings



Vasilii Ermilov



... more coming soon

Thank you!

