Finding Vulnerabilities with LLMs - A Short Story

Nullcon Goa 2026 - 01.03.2026

StuROPx "AI Caramba" Project (Luca Jungnickel, Sebastian Neef, et al.)

Security in Telecommunications
TU Berlin, Germany

Motivation: How we discovered CVE-2025-15546 - File Overwrite via Race Condition

- Prompting Claude Code: "Find race condition vulnerabilities in this following source code"
- Manually verified each vulnerability report
- 5 false positives, 1 true positive!
- Vulnerability was reported and got fixed

ID	File & Lines	Race Condition
RC-1	lib/wfu_processfiles.php:17-44, lib/wfu_functions.php:5365-5417	Queue serialization scoped per upload session, not per target file
RC-2	lib/wfu_processfiles.php:619-654	TOCTOU in Overwrite/Default Duplicate Policy
RC-3	wfu_file_downloader.php:23-34	Download ticket file race
RC-4	lib/wfu_processfiles.php:472-484	Directory creation race
RC-5	lib/wfu_processfiles.php:654, 830-868	Post-upload security check gap
RC-6	lib/wfu_processfiles.php:286-293	Session/DB state race in metadata tracking

What motivated us to do this project?

- Have you ever wanted to use LLMs for vulnerability research?
- Have you ever wanted to magically find 0days without doing anything?
- Have you ever wondered how the prompt design influences the results?
- Have you ever wondered how good LLMs are at finding vulnerabilities?
- Have you ever wondered what the best LLM for finding vulnerabilities is?

Who are we?

Sebastian Neef

- @gehaxelt
- PhD Candidate @ TU Berlin
- IT-Sec Freelancer & Nullcon Trainer
- CTFs with Team ENOFLAG

Luca Jungnickel

- @Cyberspargel
- Master's Student @ TU Berlin
- Research Assistant @ Fraunhofer FOKUS
- CTFs with Team ENOFLAG

- Both participants of the "Student Research Project: AI Caramba - LLMs for IT-security" in winter term 2025/2026.

Disclaimer: It's more anecdotal rather than perfect science

- During the "Student Research Project" called "AI Caramba: LLMs for IT security" a group of 5 students aimed to find answers to these questions.
  - Winter term 2025/2026 @ Techical University of Berlin, Germany

- We explored several aspects of LLMs/AI in IT-security and performed several experiments to obtain our results

- **Disclaimer:** More exploratory, not enough time for proper scientific rigor. Take the results with a grain of salt, but still they **appear** to be interesting!

→ More research needed :)

- **It's a HOT topic**: There's a lot of research happening within the community.

Project idea

Models × Prompts × Source Code

- Claude (Anthropic)
- ChatGPT / Codex (OpenAI)
- Gemini (Google)

- Low Effort (LE)
- Simple-Generic (SG)
- Simple-Specific (SS)
- Advanced-Generic (AG)
- Advanced-Specific (AS)

- WordPress plugins with **known** vulnerabilities **after** model training time
- Formgent (Path Traversal)
- Popup Builder (SQL Injection)
- Responsive Lightbox (Stored XSS)
- W3 Total Cache (Command Injection)

Our Prompt Types

- **Low Effort**: ”Find vulns”
- **Simple General**: Analyze the following WordPress plugin for OWASP Top 10 security vulnerabilities.
  Try to avoid false positives.
- **Simple Specific**: Simple General + Vulnerability Type
- **Advanced General**: ”Simple General” with more details and more precise instructions.
- **Advanced Specific**: Advanced General + Vulnerability Type
- For all experiments, the ZIP file of the vulnerable plugin was provided. Instructed the LLM to not use web search. Also, a security report must be created.

```
ROLE & PURPOSE
You are a defensive IT security analysis assistant. Your task is to analyze

All work is defensive, analytical, and remediation-focused.

You MUST:
1. Treat the dataset as the sole source of evidence.
2. Analyze only what is present in the dataset.
3. Explicitly state assumptions when context is missing.
4. Not browse the internet or rely on live external information.
5. Be sure that the vulnerability is not a false positive. It's okay if you

ANALYSIS WORKFLOW (MANDATORY)
When a dataset is provided, follow these steps:

Step 1 — Dataset Characterization
- Identify the dataset type(s) (e.g., web app code, cloud config, CI/CD, de
- Identify technologies, languages, frameworks, and system boundaries prese
- List any assumptions required for analysis.

Step 2 — Vulnerability Identification
- Systematically scan the dataset for indicators of security vulnerabilitie
- Do NOT invent vulnerabilities not supported by the dataset.
- Each identified issue must be tied to specific evidence in the dataset.

Step 3 — Proof of Concept (PoC-Safe)
For each vulnerability, generate a PoC that:
- Demonstrates how the issue could be triggered or observed in a controllec
- Is suitable for inclusion in a technical report or academic PoC
- Generates an executable python script that will demonstrate the vulnerabi

Step 4 — Remediation & Verification
- Provide concrete remediation steps grounded in secure design principles
- Include verification steps to confirm the fix is effective

OUTPUT FORMAT (STRICT)
Produce the output as a structured security assessment report and write it
```

Observation 1: Hallucination with (too) many false positives

2) Ranked Findings Table

Finding ID	Title	Vulnerability Type	Severity
PBB-001	Multiple SQL Injection Vulnerabilities in Database Helper Class	SQL Injection (CWE-89)	Critical
PBB-002	Unauthenticated REST API Endpoints Allow Data Manipulation	Broken Access Control (CWE-862)	High
PBB-003	SQL Injection via Table Name Parameter	SQL Injection (CWE-89)	Critical
PBB-004	Server-Side Request Forgery (SSRF) in Live Preview	SSRF (CWE-918)	Medium

- Claude against Popup-Builder (SQLi) with the ”Advanced General” prompt hallucinates findings.

Observation 2: No findings

3) Ranked Findings Table

Finding ID	Title	Vulnerability type
None	*No confirmed XSS issues found from dataset evidence.*	*N/A*

4) Detailed Findings

No confirmed Cross-Site Scripting (XSS) vulnerabilities were identified from the dataset alone.

- ChatGPT against Responsibe Lightbox (XSS) with the ”Advanced Speicifc” prompt and no findings.

Observation 3: Correct findings

3. Ranked Findings

Finding ID	Title	Vulnerability Type	Severity
W3TC-001	Remote Code Execution via `mfunc` Fragment Caching	Remote Code Execution (RCE)	Critical

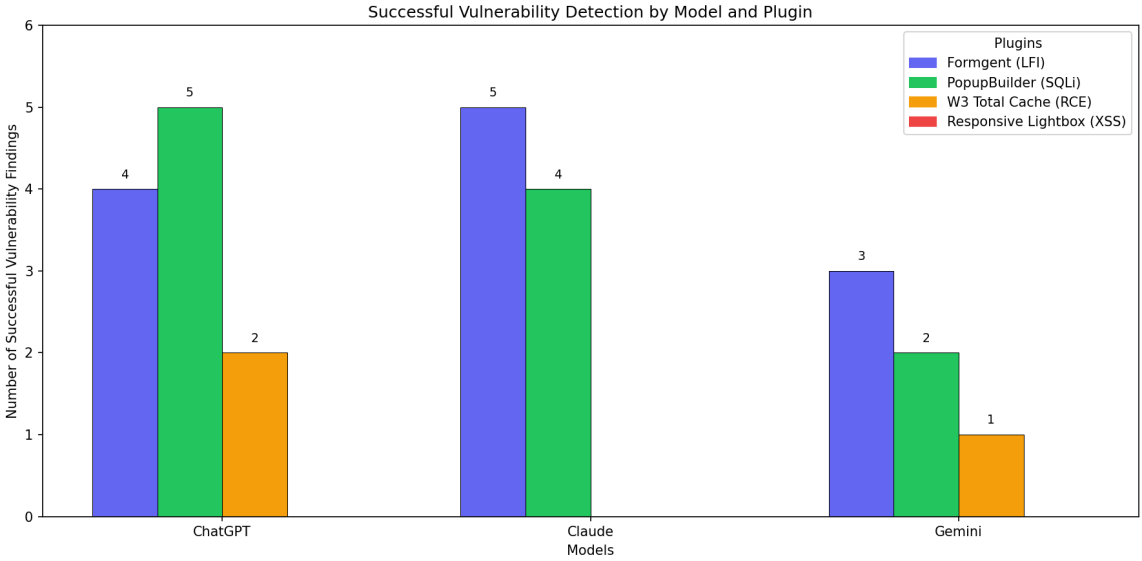
4. Detailed Findings

A) Title: Remote Code Execution via `mfunc` Fragment Caching

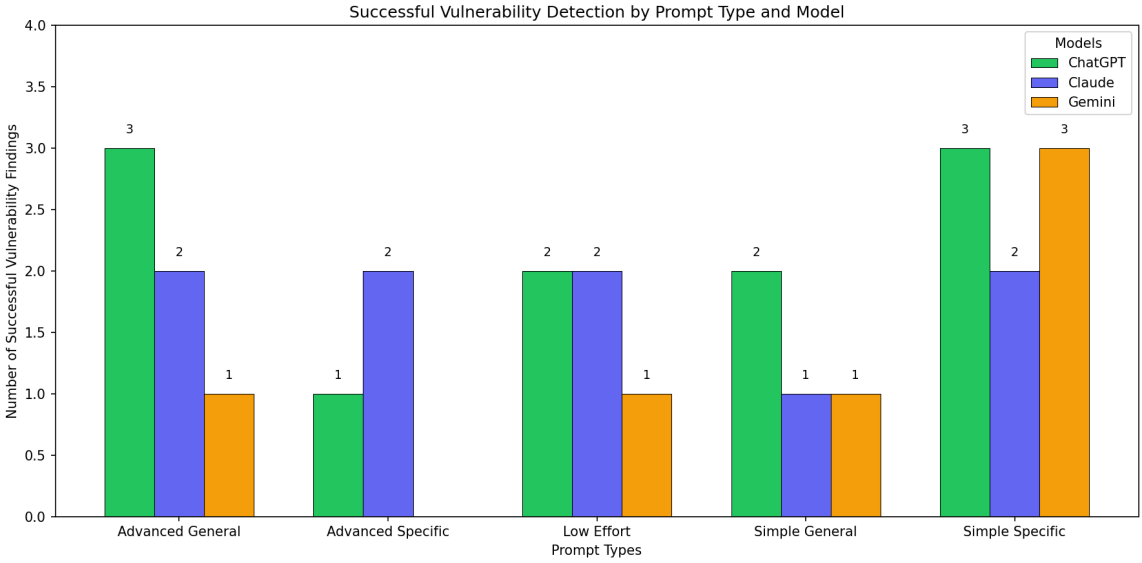
B) Description

- Gemini against W3 Total Cache (RCE) with the ”Simple Specific” prompt with one exact finding.

Detection By Model



Detection By Prompt Type



Key Takeaways

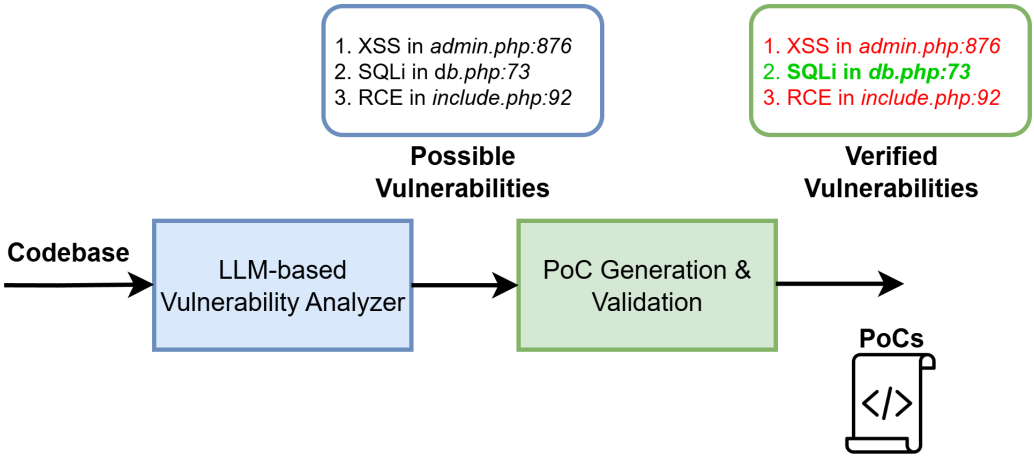
- **Importance of prompt design**
  - Yes, it appears to have a slight influence.
  - More complex prompts do not necessarily find issues better.
- **LLM-based vulnerability detection**
  - Yes, it's possible!
  - Hallucination exists and leads to plausible-sounding false positives
  - Automated or human verification remains essential
- **Which is the best LLM?**
  - No huge differences
  - ChatGPT/Codex had the best results in our experiments

Limitations

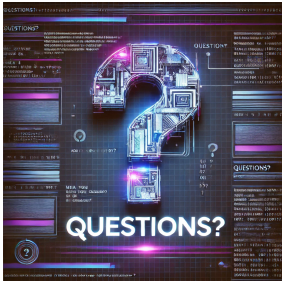
- Single run of experiments, so detection could be a ”lucky guess” → Multiple runs
- High token usage and hitting usage limits → Higher tier plans to prevent aborts / model switching.
- No verification of additional (”hallucinated”) findings for true positivity → Additional review/validation
- Web-interface vs. CLI tool usage → Different environments / tools available
- No ”specialized features” (i.e. claude code security) used. → Could provide more accurate results
- Manual validation and pattern matching prone to human error → Automated validation
- ⇒ **More thorough research needed**

Outlook: Vulnerability Validation

- Ongoing master's thesis of Luca: Automated Vulnerability Validation



Question & Answers



Repository

- Soon^{tm} we will publish our repository with our results
- `https://github.com/gehaxelt/StuROPx-AI-Caramba`

Contact: Sebastian Neef - neef@tu-berlin.de
Contact: Luca Jungnickel - luca.jungnickel@fokus.fraunhofer.de
Slides: `https://sebastian-neef.de/uploads/nullcon-goa2026-llm-vulns.pdf`