It's Giving Insecure Vibes: Secure Coding Literacy for Vibe Coders

Betta Lyon Delsordo
Penetration Testing Engineer @ AWS
February 4, 2026
AgentCon Zurich

Generating code is easy!

But spotting security mistakes isn't….

Let's get you ready to fix those vibed vulnerabilities!

Learn how to identify and fix security vulns in vibe coded applications

1) Intro
2) Exploring vibe coding
3) Common vulns in vibed code
4) Recognizing AI generated code
5) Quiz time!
6) Resolving vulns
7) AI-assisted coding
8) Workshop
9) Questions?

1) Intro

Hi, I’m Betta! I’m a hacker

- Started teaching myself to code at 13
- Began building websites for small businesses in Montana in high school through college
- Realized that web dev made me a good web hacker!
- Full cyber mode: M.S. Cyber, NSA cert program, GPEN
- Now an ethical hacker: web, cloud, AI, source code
- Currently a pentester @ AWS
- Specialize in code review and AI hacking, also building tools to speed up pentesting

Why this talk?

- Speaking from past experience as a team lead at OnDefend: led a team of pentesters to build tools, started seeing ‘vibed vulnerabilities’
- As a pentester specializing in code review, I’m finding tons more vulns in vibed code
- Necessary disclaimer: not here representing my current employer AWS, and all thoughts expressed are my own

Workshop Materials:

https://github.com/Betta-Lyon-Delsordo/insecure-vibes/

2) Exploring vibe coding

Where I’m seeing vibe coding:

- Vibe coding = using AI to write applications with very little edits
- Can be a great time saver: regex!
- But also has many risks, quality issues
- As a team lead on a pod of pentesters (and as a builder of internal software), I see more junior consultants leaning on AI
- We need more awareness of vibed vulnerabilities!

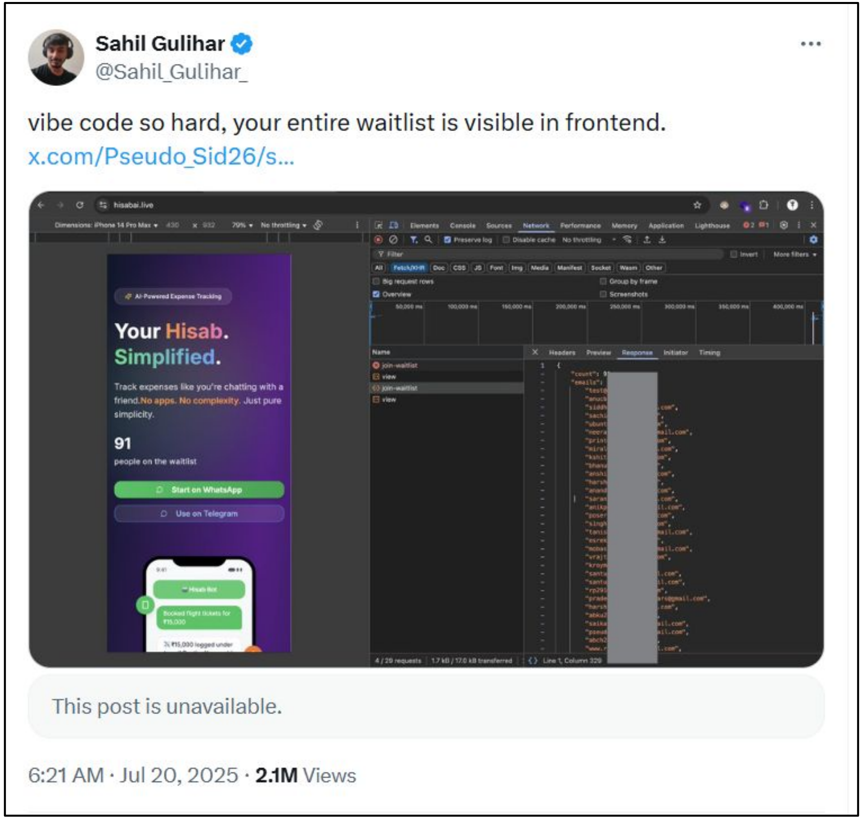
Good vibes:

- Save time writing route code
- Regex (sed/awk syntax)
- Troubleshooting error codes
- Translating from one coding language to another
- Translating comments into different human languages
- Great for prototyping, rapid ideation, internal apps
- Low barrier to entry: juniors and career changers

Bad vibes:

- Less technical users mean less understanding of code
- Very bad for scaling, troubleshooting, maintaining
- Can prevent learning and growth
- General lack of security awareness
- Very common to lack authorization and sanitization
- Public facing apps draw hacker attention -> easy breach

A viral example



Sahil Gulihar
@Sahil_Gulihar_

vibe code so hard, your entire waitlist is visible in frontend.
x.com/Pseudo_Sid26/s...

Your Hisab. Simplified.

Track expenses like you're chatting with a friend. No apps. No complexity. Just pure simplicity.

91
people on the waitlist

[redacted]

This post is unavailable.

6:21 AM · Jul 20, 2025 · **2.1M** Views

Image credit: https://x.com/Sahil_Gulihar_/status/1946787529367179551

3) Common vulns in vibed code

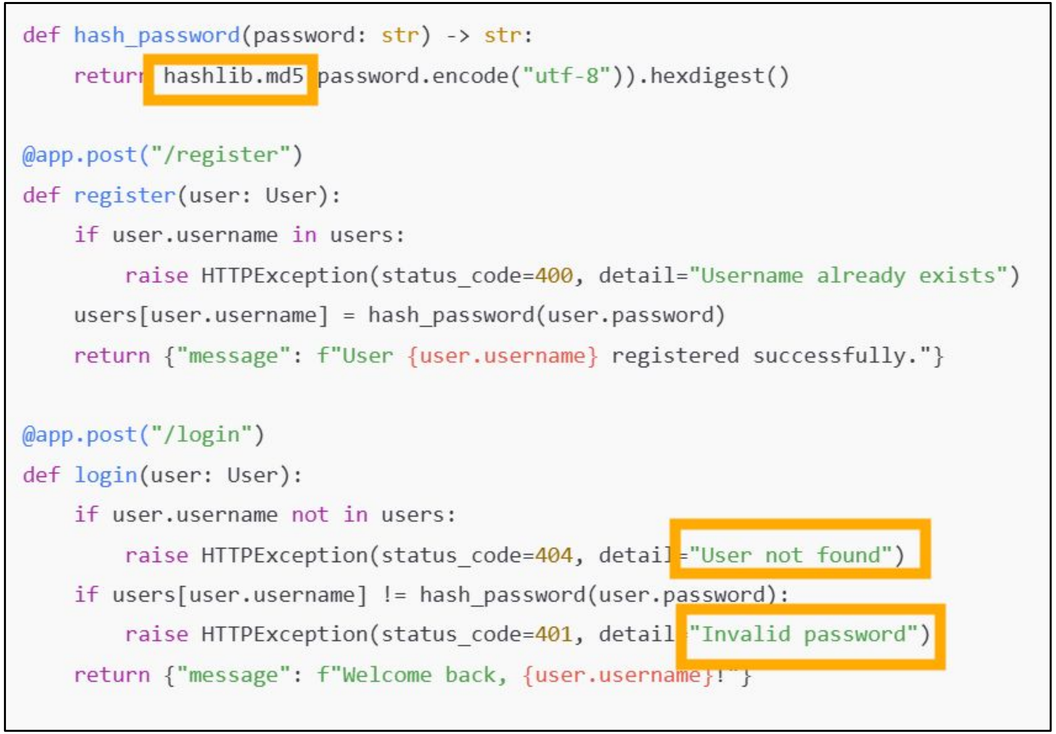
Vibed vulns I see most often:

- Exposing sensitive information - hard coded API keys and creds
- Insecure default passwords, unencrypted traffic, no auth checks
- Detailed comments about exactly how to log in and exploit it
- Displaying way too much information to public users
- Very noisy exploits (if trying to evade detection)

Vibed vulns I see most often:

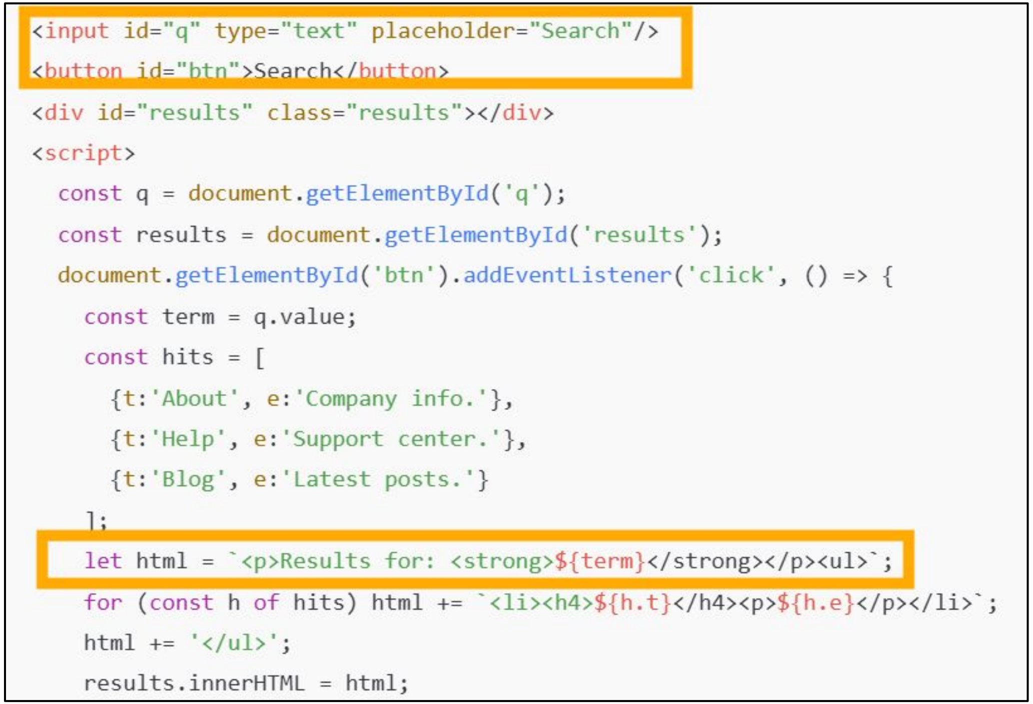
- No user input sanitization
- Pulling in malicious libraries masquerading as open source projects
- Pasting proprietary code into public/online LLMs that train on it
- Downloading malicious coding tools that claim to do ‘magic’

Examples:



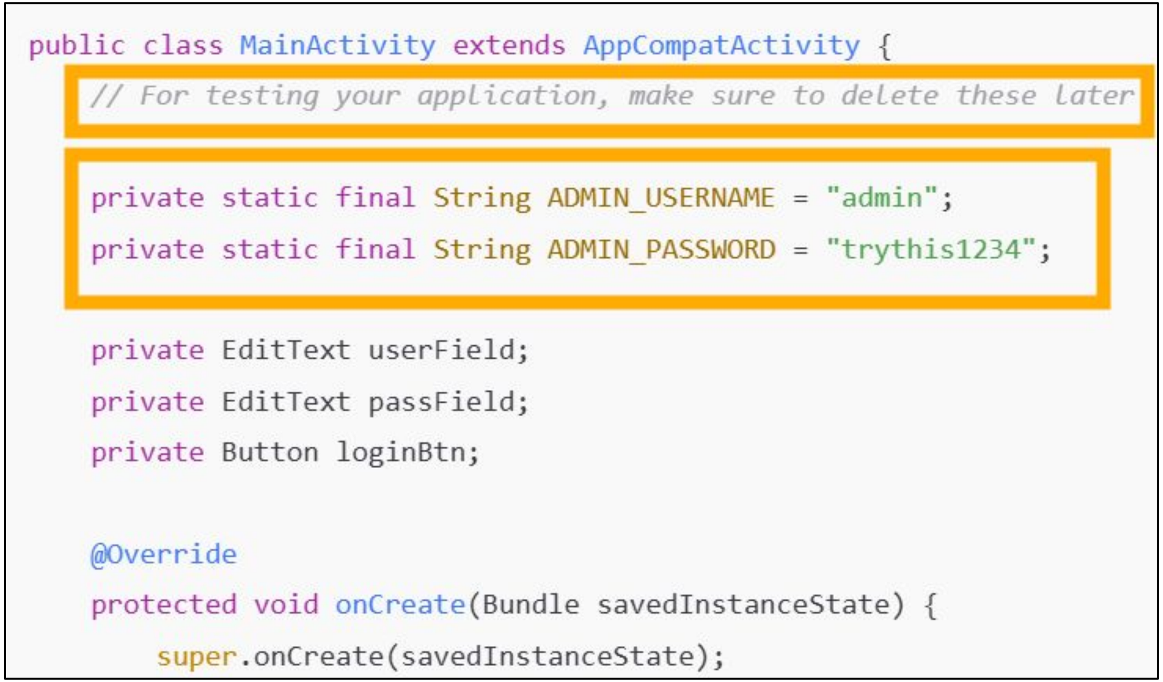
```python
def hash_password(password: str) -> str:
    return hashlib.md5(password.encode("utf-8")).hexdigest()

@app.post("/register")
def register(user: User):
    if user.username in users:
        raise HTTPException(status_code=400, detail="Username already exists")
    users[user.username] = hash_password(user.password)
    return {"message": f"User {user.username} registered successfully."}

@app.post("/login")
def login(user: User):
    if user.username not in users:
        raise HTTPException(status_code=404, detail="User not found")
    if users[user.username] != hash_password(user.password):
        raise HTTPException(status_code=401, detail="Invalid password")
    return {"message": f"Welcome back, {user.username}!"}
```

Examples:



```
<input id="q" type="text" placeholder="Search"/>
<button id="btn">Search</button>
<div id="results" class="results"></div>
<script>
  const q = document.getElementById('q');
  const results = document.getElementById('results');
  document.getElementById('btn').addEventListener('click', () => {
    const term = q.value;
    const hits = [
      {t:'About', e:'Company info.'},
      {t:'Help', e:'Support center.'},
      {t:'Blog', e:'Latest posts.'}
    ];
    let html = `<p>Results for: <strong>${term}</strong></p><ul>`;
    for (const h of hits) html += `<li><h4>${h.t}</h4><p>${h.e}</p></li>`;
    html += '</ul>';
    results.innerHTML = html;
```

Examples:



```
public class MainActivity extends AppCompatActivity {
    // For testing your application, make sure to delete these later

    private static final String ADMIN_USERNAME = "admin";
    private static final String ADMIN_PASSWORD = "trythis1234";

    private EditText userField;
    private EditText passField;
    private Button loginBtn;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

4) Recognizing AI generated code

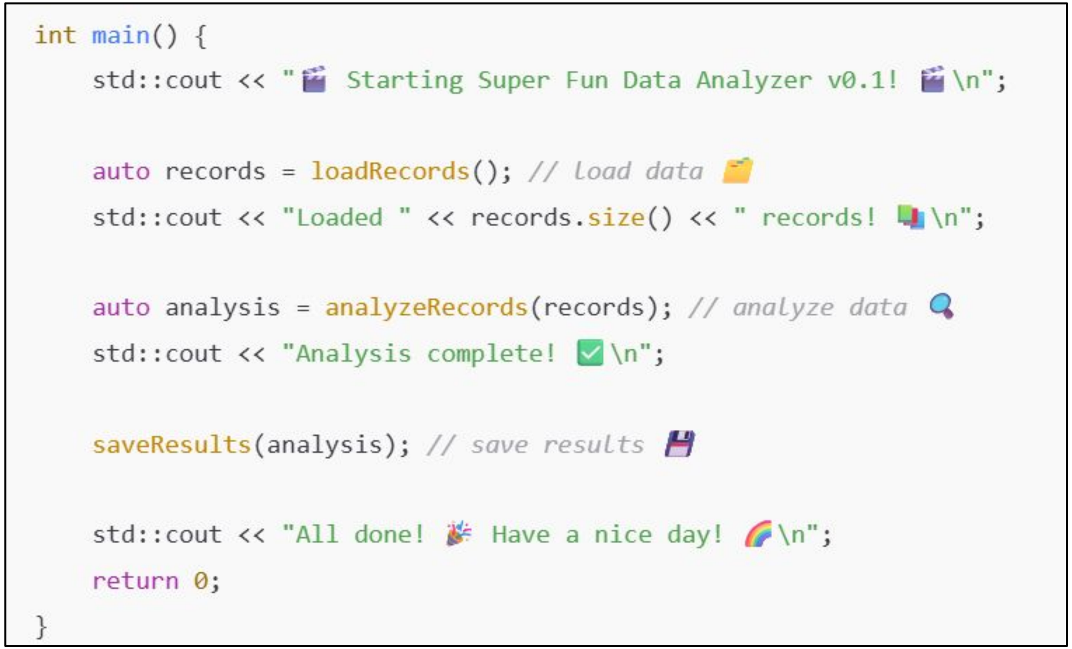
Easy giveaways for AI generated code

- Emoji comments!
- Perfect formatting, especially for large tables or JSON that would be difficult for a human to type
- Very perfect print statements with verbose language
- Redundant or unnecessary functions
- Lack of user comprehension about what the code does

Easy giveaways for AI generated code

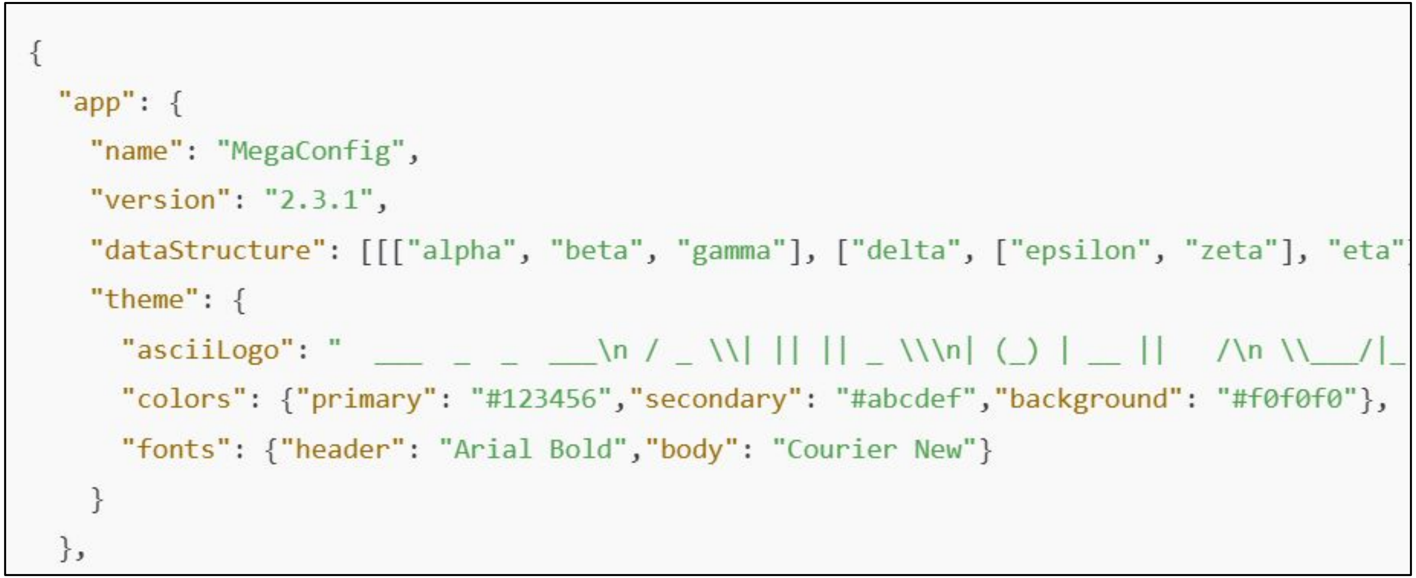
- Crashes or fails without meaningful errors, no evidence of incremental development or debugging or unit tests
- Nonsensical imports
- Function stubs that don’t do anything
- No attempt to integrate with existing environment

Examples:



```
int main() {
    std::cout << "🎬 Starting Super Fun Data Analyzer v0.1! 🎬\n";

    auto records = loadRecords(); // load data 📂
    std::cout << "Loaded " << records.size() << " records! 📚\n";

    auto analysis = analyzeRecords(records); // analyze data 🔍
    std::cout << "Analysis complete! ✅\n";

    saveResults(analysis); // save results 💾

    std::cout << "All done! 🎉 Have a nice day! 🌈\n";
    return 0;
}
```

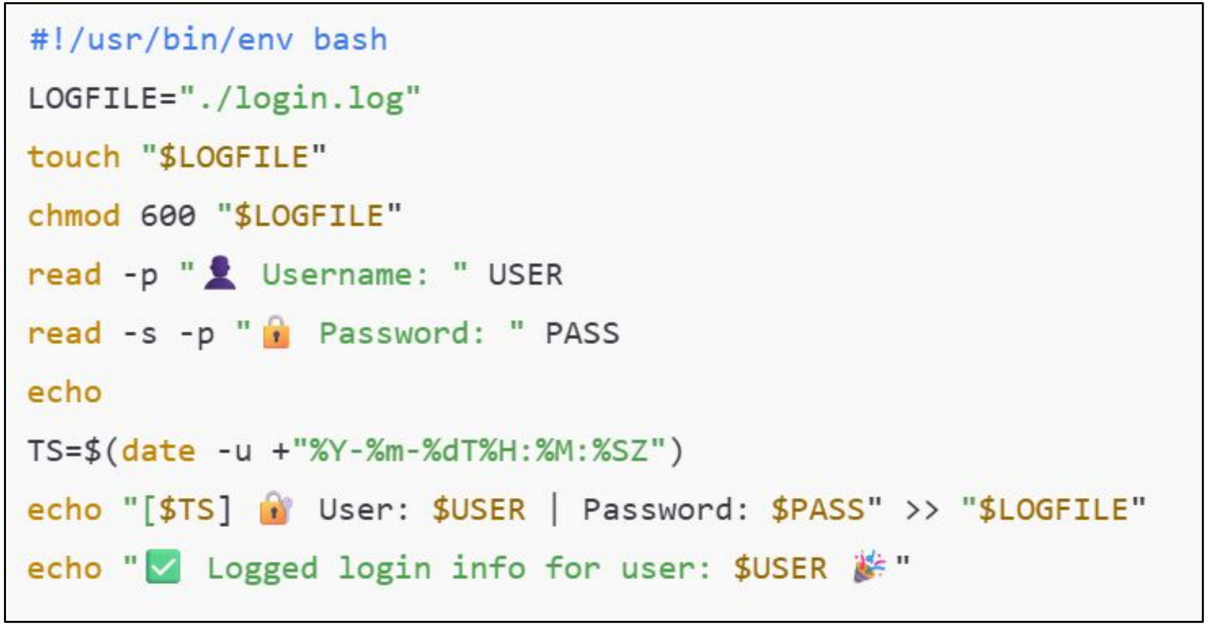
Examples:



```
{
  "app": {
    "name": "MegaConfig",
    "version": "2.3.1",
    "dataStructure": [[["alpha", "beta", "gamma"], ["delta", ["epsilon", "zeta"], "eta"
    "theme": {
      "asciiLogo": "  ___  _  _  ___\n / _ \\| || || _ \\\n| (_) | __ ||   /\n \\___/|_
      "colors": {"primary": "#123456","secondary": "#abcdef","background": "#f0f0f0"},
      "fonts": {"header": "Arial Bold","body": "Courier New"}
    }
  },
```

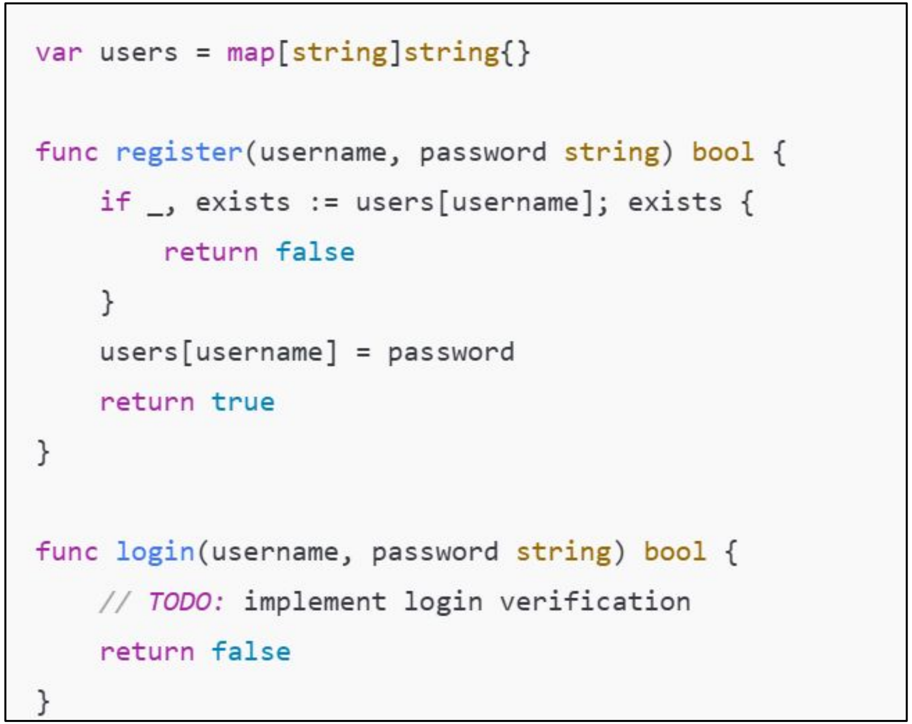
5) Quiz time!

Spot the vulns!



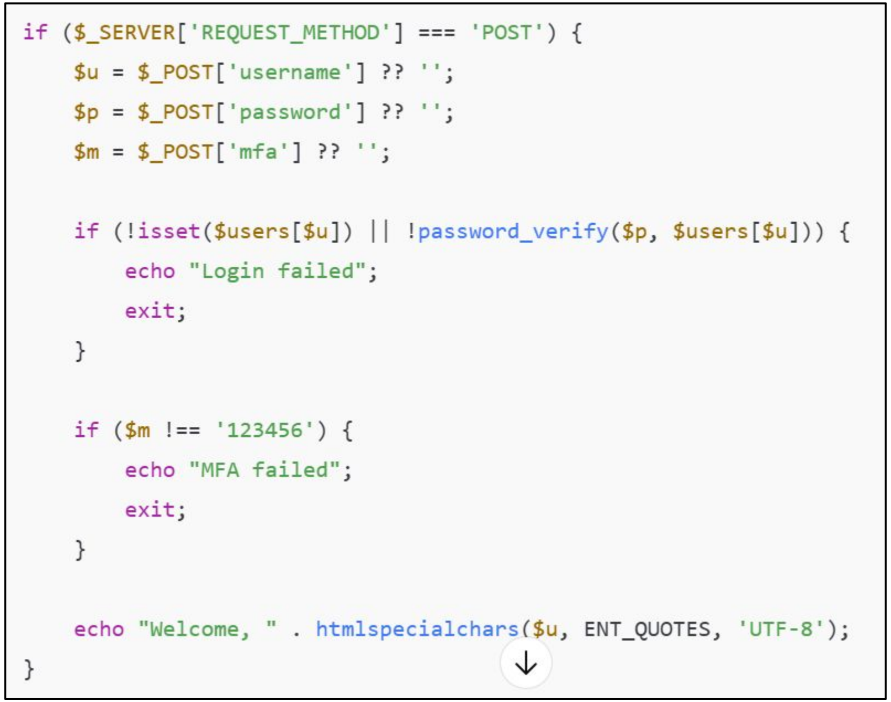
```
#!/usr/bin/env bash
LOGFILE="./login.log"
touch "$LOGFILE"
chmod 600 "$LOGFILE"
read -p "👤 Username: " USER
read -s -p "🔒 Password: " PASS
echo
TS=$(date -u +"%Y-%m-%dT%H:%M:%SZ")
echo "[$TS] 🔏 User: $USER | Password: $PASS" >> "$LOGFILE"
echo "✅ Logged login info for user: $USER 🎉"
```

Spot the vulns!



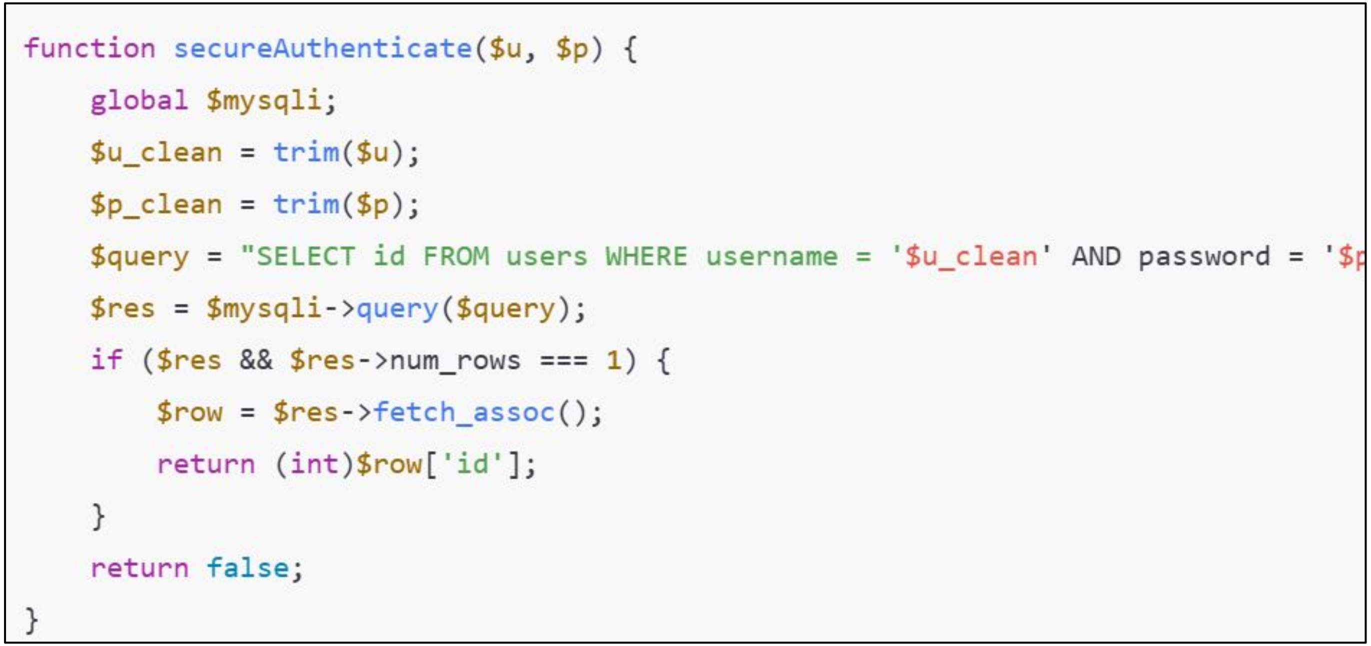
```
var users = map[string]string{}

func register(username, password string) bool {
    if _, exists := users[username]; exists {
        return false
    }
    users[username] = password
    return true
}

func login(username, password string) bool {
    // TODO: implement login verification
    return false
}
```

Spot the vulns!



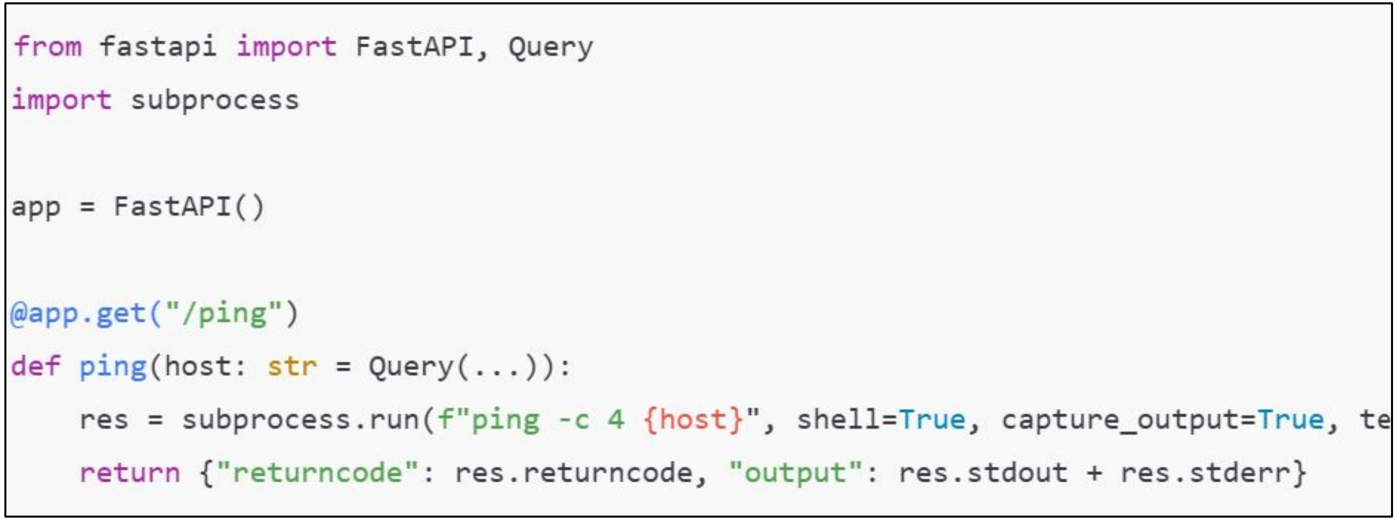
```
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $u = $_POST['username'] ?? '';
    $p = $_POST['password'] ?? '';
    $m = $_POST['mfa'] ?? '';

    if (!isset($users[$u]) || !password_verify($p, $users[$u])) {
        echo "Login failed";
        exit;
    }

    if ($m !== '123456') {
        echo "MFA failed";
        exit;
    }

    echo "Welcome, " . htmlspecialchars($u, ENT_QUOTES, 'UTF-8');
}
```

Spot the vulns!



```
function secureAuthenticate($u, $p) {
    global $mysqli;
    $u_clean = trim($u);
    $p_clean = trim($p);
    $query = "SELECT id FROM users WHERE username = '$u_clean' AND password = '$p
    $res = $mysqli->query($query);
    if ($res && $res->num_rows === 1) {
        $row = $res->fetch_assoc();
        return (int)$row['id'];
    }
    return false;
}
```

Spot the vulns!



```
from fastapi import FastAPI, Query
import subprocess

app = FastAPI()

@app.get("/ping")
def ping(host: str = Query(...)):
    res = subprocess.run(f"ping -c 4 {host}", shell=True, capture_output=True, te
    return {"returncode": res.returncode, "output": res.stdout + res.stderr}
```

6) Resolving vulns

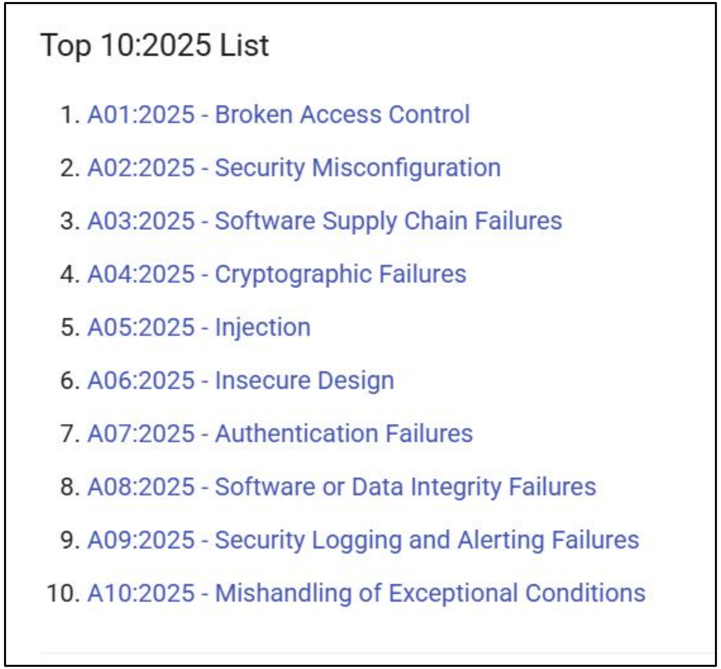
Resolve those vulns!

- Learn some basic application security: OWASP Top 10 is a great place to start
- Prompt with emphasis on secure coding
- Ask the AI to review its own code for security
- Adversarial AI: ask another AI to find the vulnerabilities
- Always ask for secure defaults, no hard coding creds, do user input sanitization, authorization checks

Resolve those vulns!

- Do a thorough review of the code, spend more time reviewing than you did coding!
- Get help from someone who knows, and keep learning how to actually code
- Don’t put things into production or accept sensitive data if you don’t know what you’re doing
- Train junior members on the risks of AI coding and keep training them on real, manual coding
- Make sure your team understands the risks of sharing code with public/online LLMs and train them to use private/local alternatives

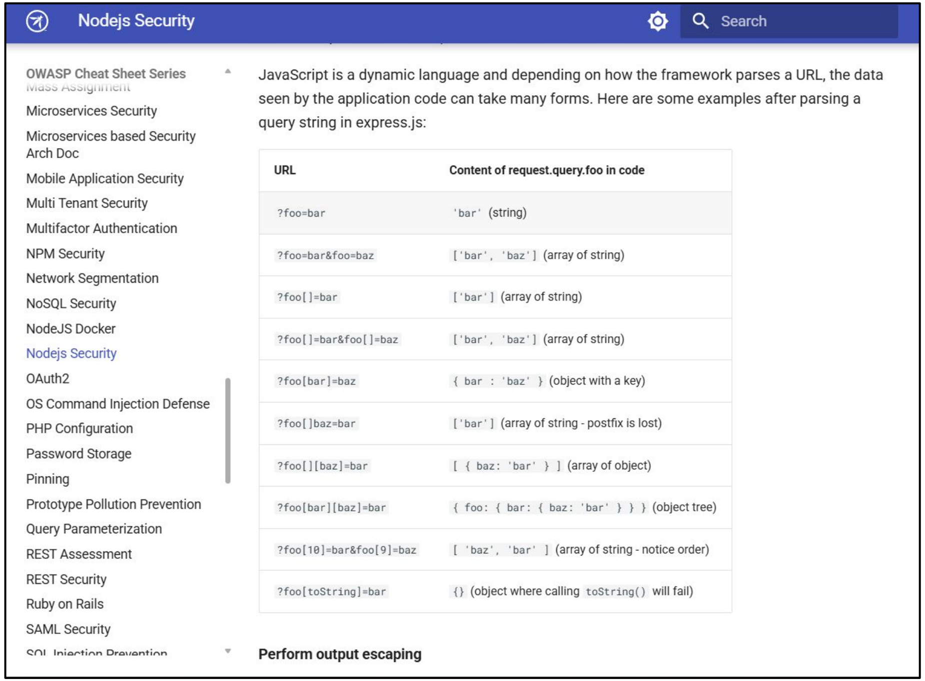
OWASP Top 10



Top 10:2025 List

1. A01:2025 - Broken Access Control
2. A02:2025 - Security Misconfiguration
3. A03:2025 - Software Supply Chain Failures
4. A04:2025 - Cryptographic Failures
5. A05:2025 - Injection
6. A06:2025 - Insecure Design
7. A07:2025 - Authentication Failures
8. A08:2025 - Software or Data Integrity Failures
9. A09:2025 - Security Logging and Alerting Failures
10. A10:2025 - Mishandling of Exceptional Conditions

OWASP Secure Coding Cheatsheets

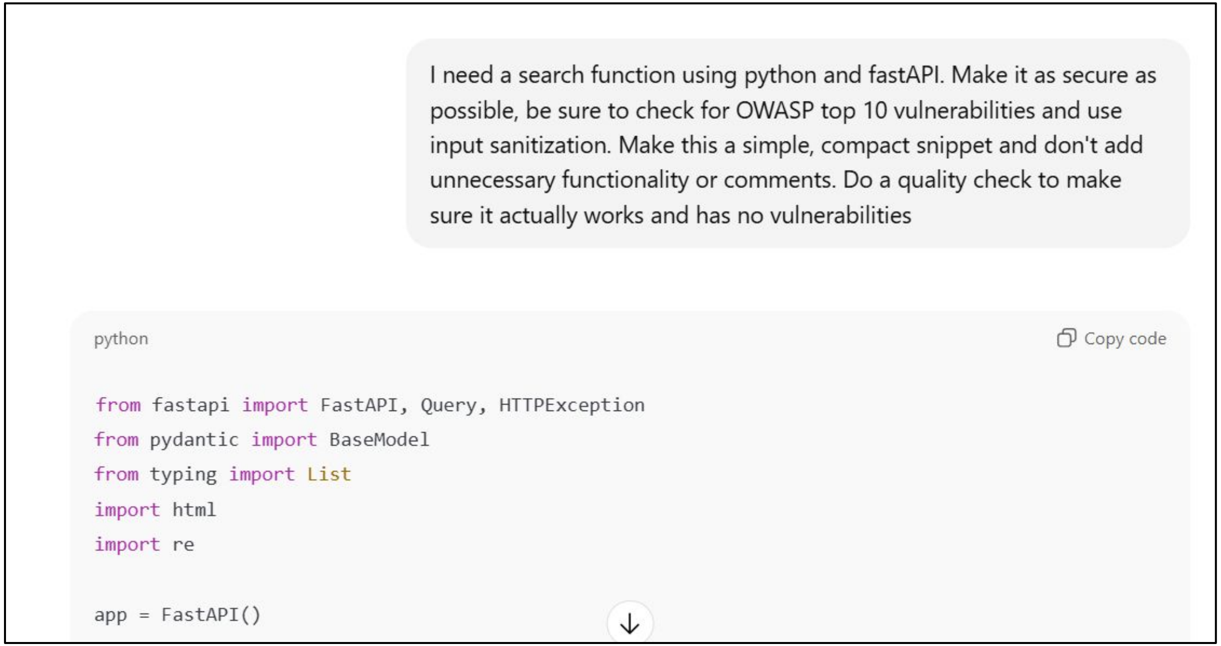


JavaScript is a dynamic language and depending on how the framework parses a URL, the data seen by the application code can take many forms. Here are some examples after parsing a query string in express.js:

URL	Content of request.query.foo in code
`?foo=bar`	`'bar'` (string)
`?foo=bar&foo=baz`	`['bar', 'baz']` (array of string)
`?foo[]=bar`	`['bar']` (array of string)
`?foo[]=bar&foo[]=baz`	`['bar', 'baz']` (array of string)
`?foo[bar]=baz`	`{ bar : 'baz' }` (object with a key)
`?foo[]baz=bar`	`['bar']` (array of string - postfix is lost)
`?foo[][baz]=bar`	`[ { baz: 'bar' } ]` (array of object)
`?foo[bar][baz]=bar`	`{ foo: { bar: { baz: 'bar' } } }` (object tree)
`?foo[10]=bar&foo[9]=baz`	`[ 'baz', 'bar' ]` (array of string - notice order)
`?foo[toString]=bar`	`{}` (object where calling `toString()` will fail)

Perform output escaping

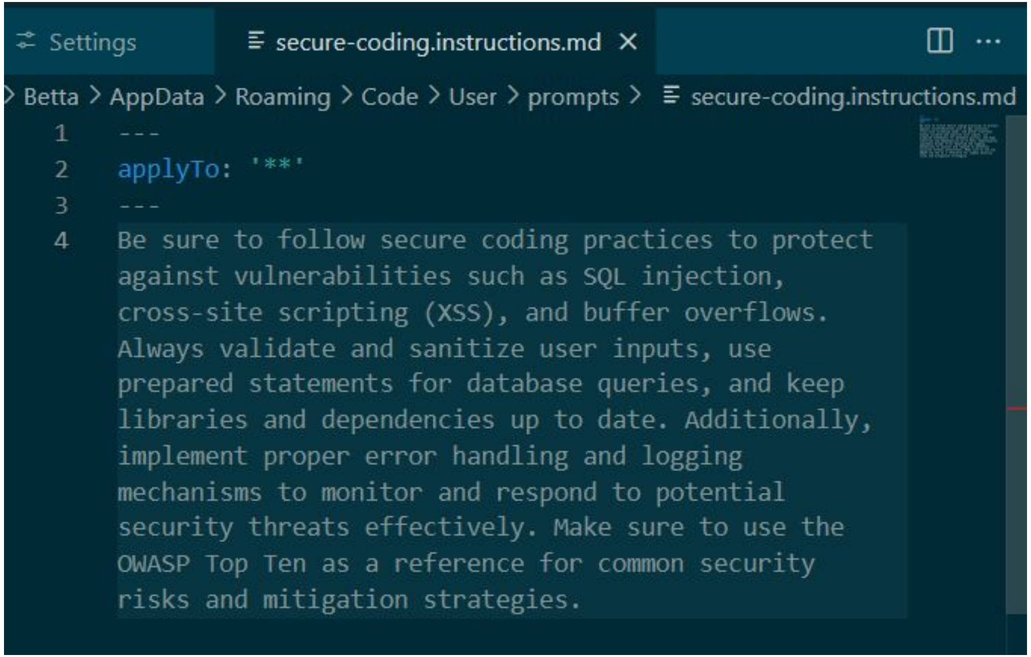
Secure prompting:



I need a search function using python and fastAPI. Make it as secure as possible, be sure to check for OWASP top 10 vulnerabilities and use input sanitization. Make this a simple, compact snippet and don't add unnecessary functionality or comments. Do a quality check to make sure it actually works and has no vulnerabilities

```python
from fastapi import FastAPI, Query, HTTPException
from pydantic import BaseModel
from typing import List
import html
import re

app = FastAPI()
```

Add an instruction file:


Settings
secure-coding.instructions.md
Betta > AppData > Roaming > Code > User > prompts > secure-coding.instructions.md

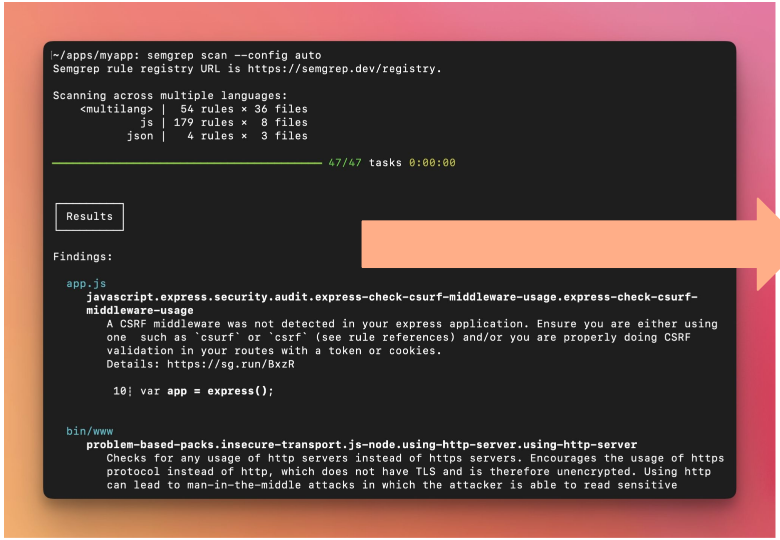
```
---
applyTo: '**'
---
Be sure to follow secure coding practices to protect against vulnerabilities such as SQL injection, cross-site scripting (XSS), and buffer overflows. Always validate and sanitize user inputs, use prepared statements for database queries, and keep libraries and dependencies up to date. Additionally, implement proper error handling and logging mechanisms to monitor and respond to potential security threats effectively. Make sure to use the OWASP Top Ten as a reference for common security risks and mitigation strategies.
```

Code scanning tools + vibing fixes

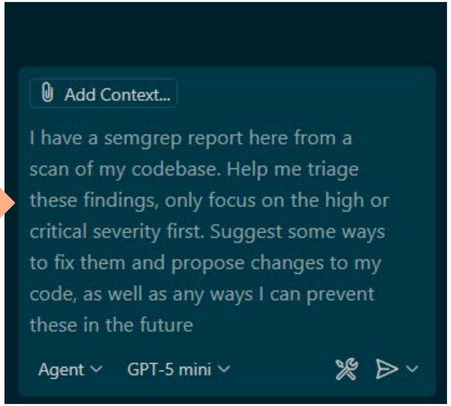
- Get familiar with a few code scanning tools, plenty of free + open source options
  - Semgrep: code scanning
  - OWASP Dependency checker: supply chain security
  - NPM audit: easy way to check dependencies
- Ask your AI to help you set it up if it’s too confusing
- Then once you have some scan output, use an agent to take a look, filter out false positives, correct any mistakes
- Focus on high severity first, there will be a lot of noise
- Add to your system prompt to correct any trends

Semgrep example:

- https://github.com/semgrep/semgrep



```
~/apps/myapp: semgrep scan --config auto
Semgrep rule registry URL is https://semgrep.dev/registry.

Scanning across multiple languages:
    <multilang> |  54 rules × 36 files
             js | 179 rules ×  8 files
           json |   4 rules ×  3 files

──────────────────────────────────────── 47/47 tasks 0:00:00


 Results

Findings:

  app.js
     javascript.express.security.audit.express-check-csurf-middleware-usage.express-check-csurf-
     middleware-usage
        A CSRF middleware was not detected in your express application. Ensure you are either using
        one  such as `csurf` or `csrf` (see rule references) and/or you are properly doing CSRF
        validation in your routes with a token or cookies.
        Details: https://sg.run/BxzR

         10┆ var app = express();


  bin/www
     problem-based-packs.insecure-transport.js-node.using-http-server.using-http-server
        Checks for any usage of http servers instead of https servers. Encourages the usage of https
        protocol instead of http, which does not have TLS and is therefore unencrypted. Using http
        can lead to man-in-the-middle attacks in which the attacker is able to read sensitive
```

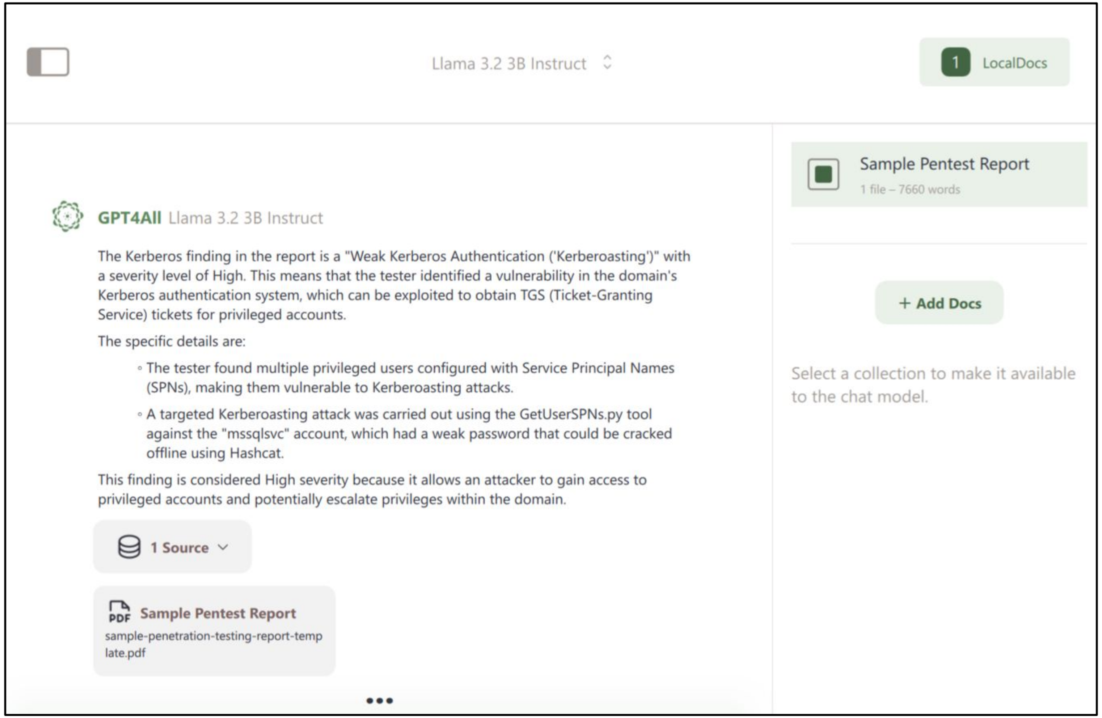


I have a semgrep report here from a scan of my codebase. Help me triage these findings, only focus on the high or critical severity first. Suggest some ways to fix them and propose changes to my code, as well as any ways I can prevent these in the future

Offline AI

- Encourage team members to use offline or private AI solutions
- Especially important with pentesting, national security, proprietary code, sensitive datasets
- GPT4All is a great option: https://www.nomic.ai/gpt4all
- Use with a model like Llama 4, and turn off analytics and data lake on startup -> then fully private

GPT4All: Offline local docs



Select a collection to make it available to the chat model.

GPT4All Llama 3.2 3B Instruct

The Kerberos finding in the report is a "Weak Kerberos Authentication ('Kerberoasting')" with a severity level of High. This means that the tester identified a vulnerability in the domain's Kerberos authentication system, which can be exploited to obtain TGS (Ticket-Granting Service) tickets for privileged accounts.

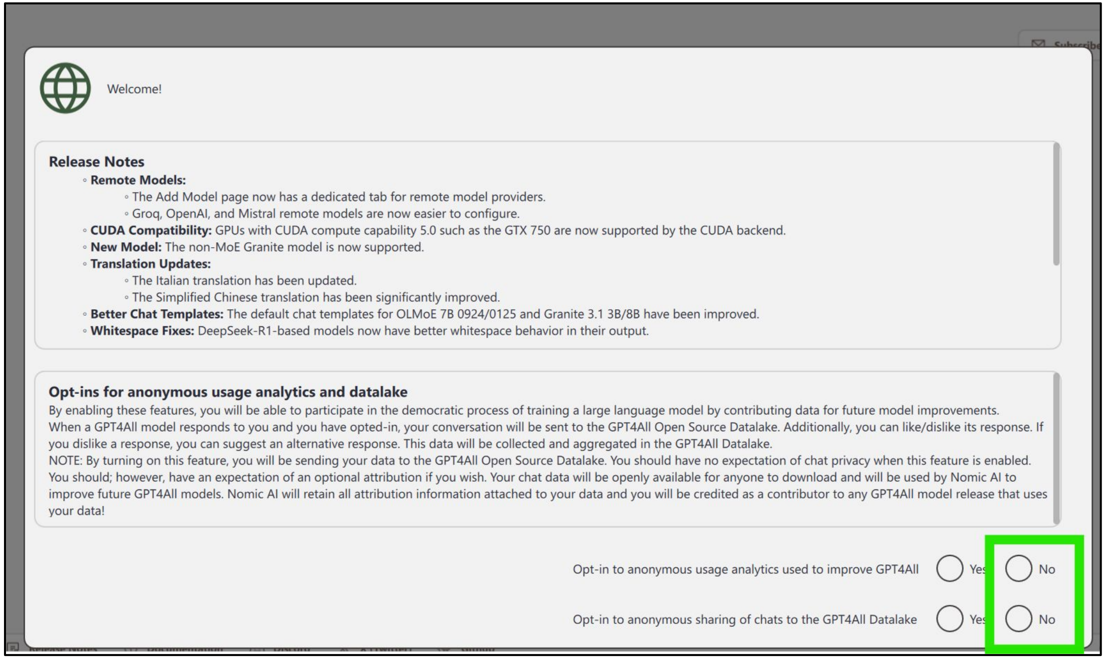
The specific details are:

- The tester found multiple privileged users configured with Service Principal Names (SPNs), making them vulnerable to Kerberoasting attacks.
- A targeted Kerberoasting attack was carried out using the GetUserSPNs.py tool against the "mssqlsvc" account, which had a weak password that could be cracked offline using Hashcat.

This finding is considered High severity because it allows an attacker to gain access to privileged accounts and potentially escalate privileges within the domain.

Sample Pentest Report
sample-penetration-testing-report-template.pdf

GPT4All startup options



Welcome!

Release Notes

- **Remote Models:**
  - The Add Model page now has a dedicated tab for remote model providers.
  - Groq, OpenAI, and Mistral remote models are now easier to configure.
- **CUDA Compatibility:** GPUs with CUDA compute capability 5.0 such as the GTX 750 are now supported by the CUDA backend.
- **New Model:** The non-MoE Granite model is now supported.
- **Translation Updates:**
  - The Italian translation has been updated.
  - The Simplified Chinese translation has been significantly improved.
- **Better Chat Templates:** The default chat templates for OLMoE 7B 0924/0125 and Granite 3.1 3B/8B have been improved.
- **Whitespace Fixes:** DeepSeek-R1-based models now have better whitespace behavior in their output.

Opt-ins for anonymous usage analytics and datalake

By enabling these features, you will be able to participate in the democratic process of training a large language model by contributing data for future model improvements.
When a GPT4All model responds to you and you have opted-in, your conversation will be sent to the GPT4All Open Source Datalake. Additionally, you can like/dislike its response. If you dislike a response, you can suggest an alternative response. This data will be collected and aggregated in the GPT4All Datalake.
NOTE: By turning on this feature, you will be sending your data to the GPT4All Open Source Datalake. You should have no expectation of chat privacy when this feature is enabled. You should; however, have an expectation of an optional attribution if you wish. Your chat data will be openly available for anyone to download and will be used by Nomic AI to improve future GPT4All models. Nomic AI will retain all attribution information attached to your data and you will be credited as a contributor to any GPT4All model release that uses your data!

Opt-in to anonymous usage analytics used to improve GPT4All ○ Yes ○ No

Opt-in to anonymous sharing of chats to the GPT4All Datalake ○ Yes ○ No

7) AI-assisted coding

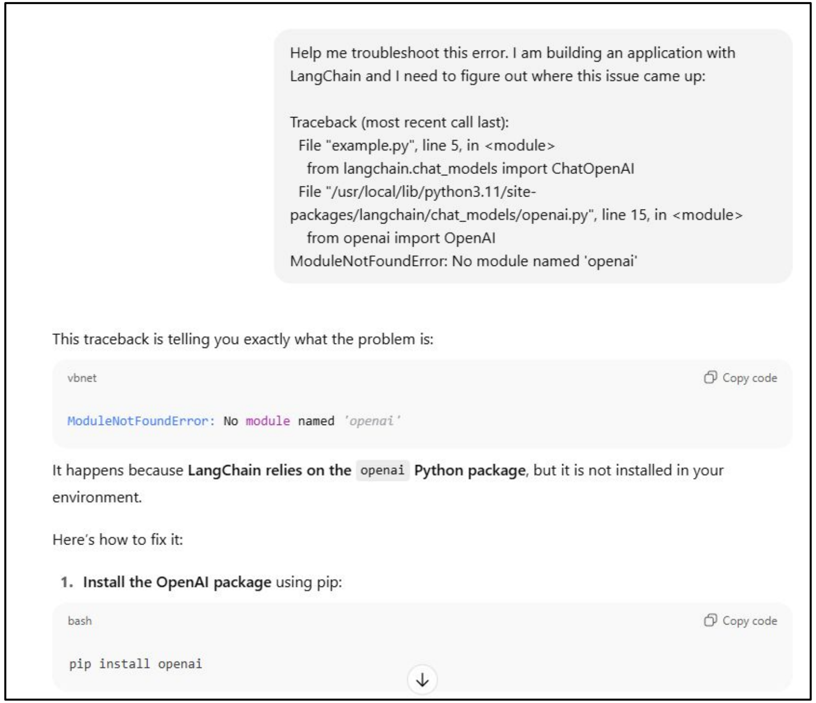
How to do ‘AI-assisted coding’

- Instead of having the AI write everything, use it as a companion to help you troubleshoot tough errors
- Try to code out the entire application as you normally would manually, but then ask for help when you get tough errors or are exploring new tech stacks
- Get help with troubleshooting and ask for secure architecture advice

How to do ‘AI-assisted coding’

- Ask an AI to review your code for vulnerabilities
- Ask for tutoring and quiz sessions to help you learn more kinds of vulnerabilities
- Use vibe coding for quick, internal scripts (like regex or awk/sed) and know when go manual for bigger projects

Examples:



Help me troubleshoot this error. I am building an application with LangChain and I need to figure out where this issue came up:

```
Traceback (most recent call last):
  File "example.py", line 5, in <module>
    from langchain.chat_models import ChatOpenAI
  File "/usr/local/lib/python3.11/site-packages/langchain/chat_models/openai.py", line 15, in <module>
    from openai import OpenAI
ModuleNotFoundError: No module named 'openai'
```

This traceback is telling you exactly what the problem is:

```vbnet
ModuleNotFoundError: No module named 'openai'
```

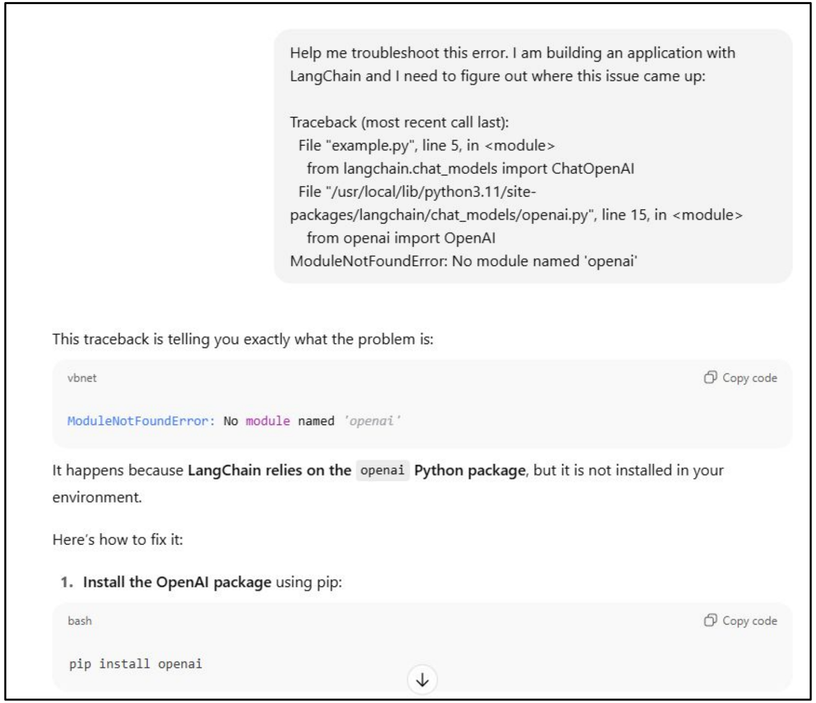
It happens because **LangChain relies on the `openai` Python package**, but it is not installed in your environment.

Here's how to fix it:

1. **Install the OpenAI package** using pip:

```bash
pip install openai
```

Examples:



Help me troubleshoot this error. I am building an application with LangChain and I need to figure out where this issue came up:

```
Traceback (most recent call last):
  File "example.py", line 5, in <module>
    from langchain.chat_models import ChatOpenAI
  File "/usr/local/lib/python3.11/site-
packages/langchain/chat_models/openai.py", line 15, in <module>
    from openai import OpenAI
ModuleNotFoundError: No module named 'openai'
```

This traceback is telling you exactly what the problem is:

```vbnet
ModuleNotFoundError: No module named 'openai'
```

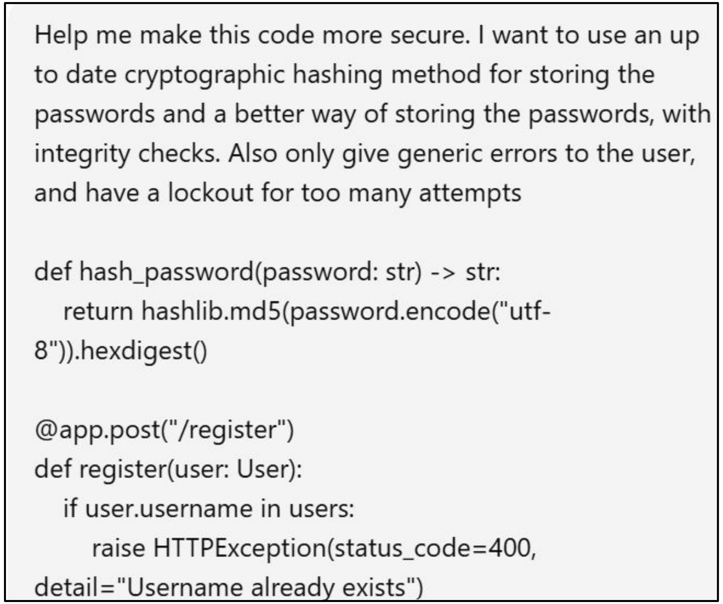
It happens because **LangChain relies on the `openai` Python package**, but it is not installed in your environment.

Here's how to fix it:

1. **Install the OpenAI package** using pip:

```bash
pip install openai
```

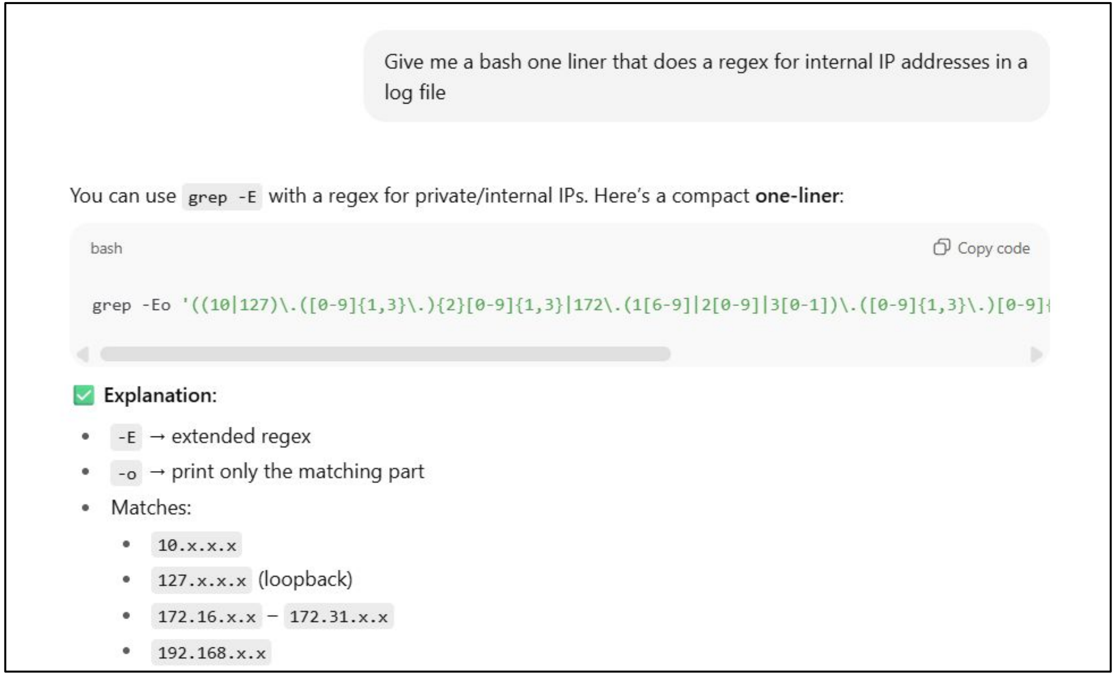
Examples:



Help me make this code more secure. I want to use an up to date cryptographic hashing method for storing the passwords and a better way of storing the passwords, with integrity checks. Also only give generic errors to the user, and have a lockout for too many attempts

```
def hash_password(password: str) -> str:
    return hashlib.md5(password.encode("utf-
8")).hexdigest()

@app.post("/register")
def register(user: User):
    if user.username in users:
        raise HTTPException(status_code=400,
detail="Username already exists")
```

Examples:



Give me a bash one liner that does a regex for internal IP addresses in a log file

You can use `grep -E` with a regex for private/internal IPs. Here's a compact **one-liner**:

```bash
grep -Eo '((10|127)\.([0-9]{1,3}\.){2}[0-9]{1,3}|172\.(1[6-9]|2[0-9]|3[0-1])\.([0-9]{1,3}\.)[0-9]
```

✅ **Explanation:**

- `-E` → extended regex
- `-o` → print only the matching part
- Matches:
  - `10.x.x.x`
  - `127.x.x.x` (loopback)
  - `172.16.x.x` – `172.31.x.x`
  - `192.168.x.x`

Further reading:

- https://cloudsecurityalliance.org/blog/2025/04/09/secure-vibe-coding-guide#
- https://owasp.org/Top10/2025/
- https://platform.claude.com/docs/en/agent-sdk/modifying-system-prompts
- https://arxiv.org/abs/2512.03262
- https://cheatsheetseries.owasp.org/index.html

8) Workshop

Workshop Materials:

https://github.com/Betta-Lyon-Delsordo/insecure-vibes/

9) Questions?

Let's connect!

https://www.linkedin.com/in/betta-lyon-delsordo/