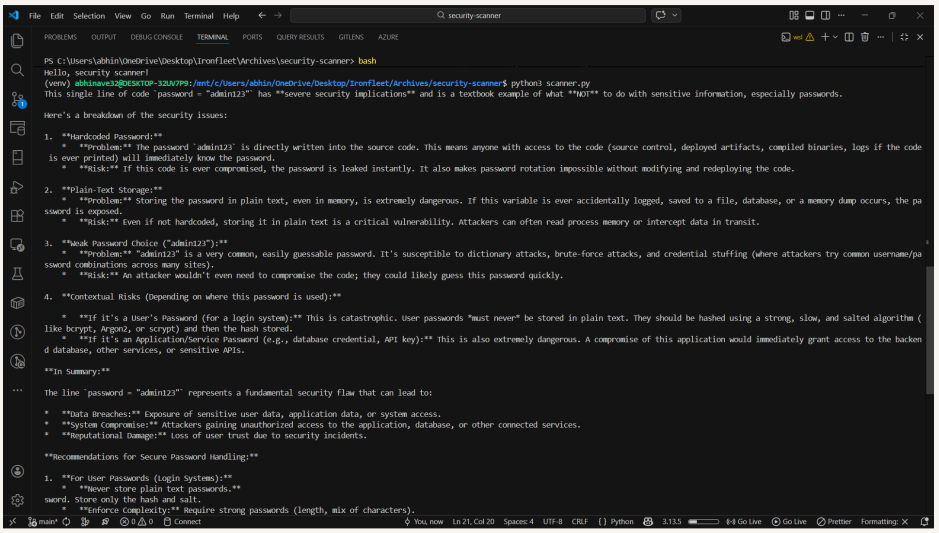
AI Security Scanner for Python

Abhinave P.B



Abhinave P.B
NextWork Student

Introducing Today's Project!

In this project, I'm going to build AI Security Scanner for python This will help me learn how code vulnerabilities and how to manage them in a production environment. I'm interested in this because of to learn how to avoid security vulnerabilities in my code

Key tools and concepts

Tools I used were Python, Gemini API, dotenv, and colorama for colored terminal output. Key concepts I learnt include API integration, environment variable management, prompt engineering for security analysis, and detecting common vulnerabilities like weak hashing and SQL injection. The most important skill was building an end-to-end workflow — reading files, sending them to an AI model, and presenting structured, meaningful security results.

Challenges and wins

This project took me approximately a few hours to design, test, and refine. The most challenging part was crafting an effective security prompt and handling API errors while ensuring structured, reliable output. It was most rewarding to see the scanner successfully detect real vulnerabilities and display them with clear, color-coded severity levels.

Abhinave P.B
NextWork Student

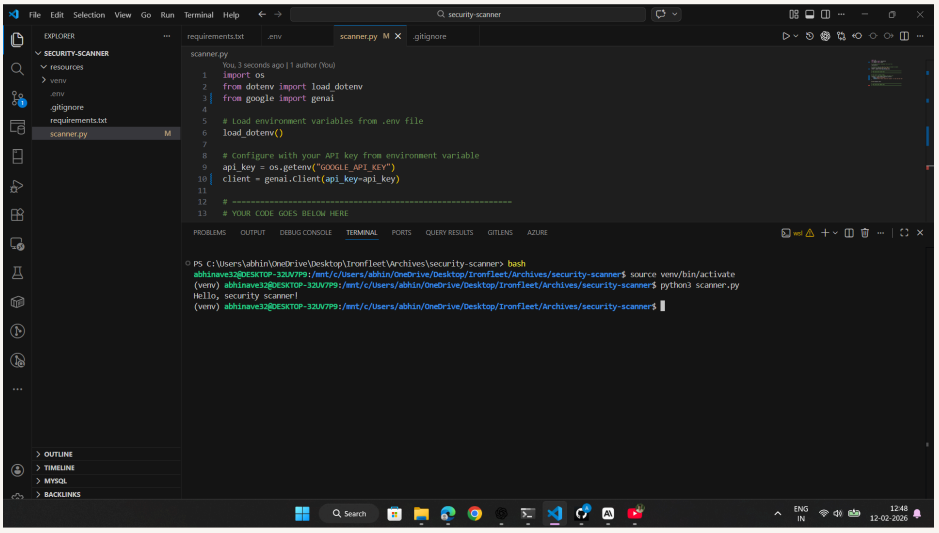
Why I did this project

I did this project today because I wanted to strengthen my understanding of cybersecurity and learn how to integrate AI into real-world tools. This project met my goals by helping me build a working AI-powered vulnerability scanner that detects security issues and presents structured results. Next, I plan to improve it by adding more vulnerability categories, better report formatting (maybe JSON export), and possibly turning it into a CLI tool or web app.

Abhinave P.B
NextWork Student

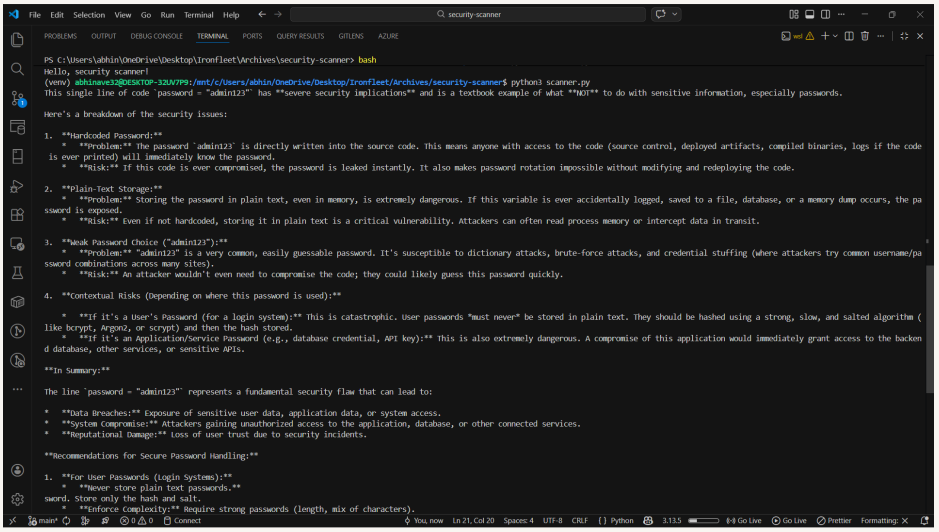
Connecting to Gemini API

In this step, I'm setting up the Gemini API connection. This involves configuring the Google Generative AI SDK, loading environment variables securely, authenticating using my API key, and initializing a Gemini model instance. I need to do this so my application can communicate with Google’s AI infrastructure, send prompts, and receive model-generated responses in real time. Establishing this connection is a foundational step for integrating AI capabilities into my project.



I verified the connection by running a test request to the Gemini API. Gemini responded with a valid output, which confirmed that the API key and client configuration were working correctly.

Abhinave P.B
NextWork Student

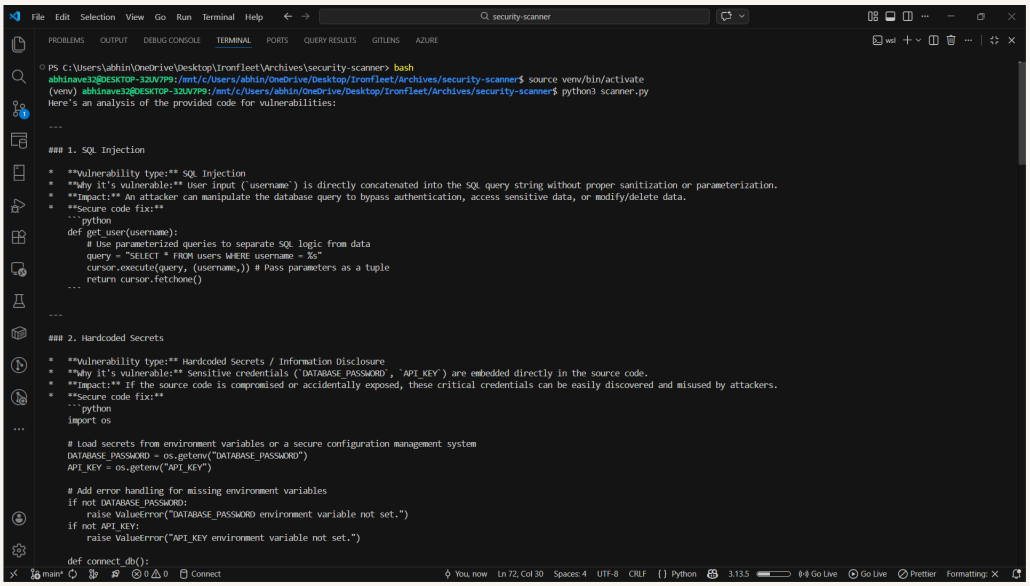


My scanner.py file works by sending a code snippet to the Gemini API using the generate_content method and requesting a security analysis. When I ran it, Gemini identified that the line password = "admin123" contains severe security vulnerabilities, including hardcoded credentials, plain-text storage risks, and weak password practices. This shows that the Gemini API can successfully analyze source code, detect common security flaws, and provide structured explanations along with best-practice recommendations.

Abhinave P.B
NextWork Student

Building the Vulnerability Scanner

In this step, I am designing a prompt-driven static vulnerability scanner that evaluates source code for common security misconfigurations and injection risks. The system will feed vulnerable samples into a large language model and assess detection quality, remediation accuracy, and severity classification. This forms the basis of an AI-assisted code auditing pipeline.



placeholder

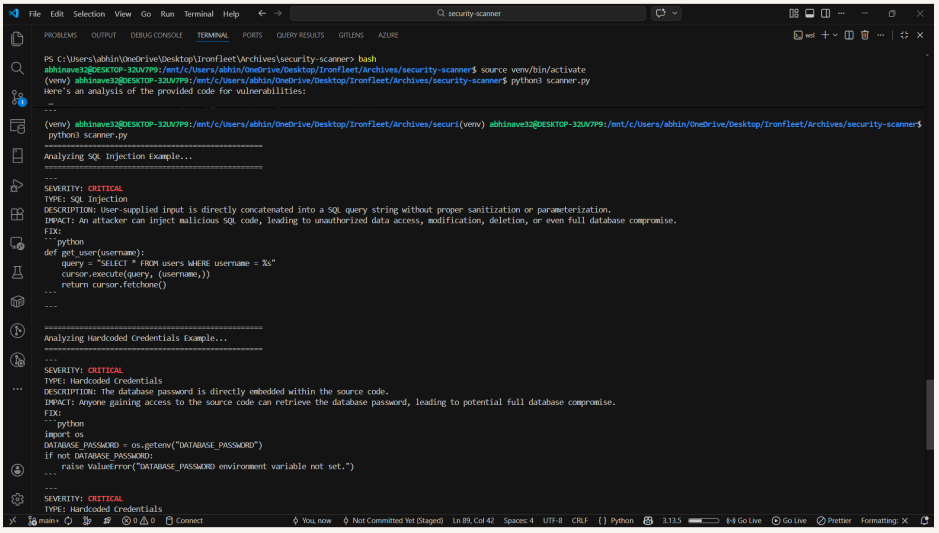
The vulnerabilities Gemini detected were SQL injection through unsafe string interpolation, hardcoded API keys and database credentials stored in plain text, and weak password validation logic that only checked minimum length without enforcing complexity requirements.

It also highlighted unsafe input handling patterns and recommended parameterized queries, environment variables for secret management, and stronger password policies with secure hashing algorithms like bcrypt or Argon2. The security prompt I crafted asked for explicit identification of SQL injection, hardcoded secrets, and weak password logic; a clear explanation of each vulnerability; the exact vulnerable lines of code; a severity classification (Low/Medium/High/Critical); and a secure, corrected version of the code. It required precise, technical responses rather than generic advice.

Abhinave P.B
NextWork Student

Adding Severity Ratings

In this step, severity ratings are introduced to categorize identified issues by impact level. The colorama library is installed to enable colored terminal output, improving clarity and user experience when displaying scan results.

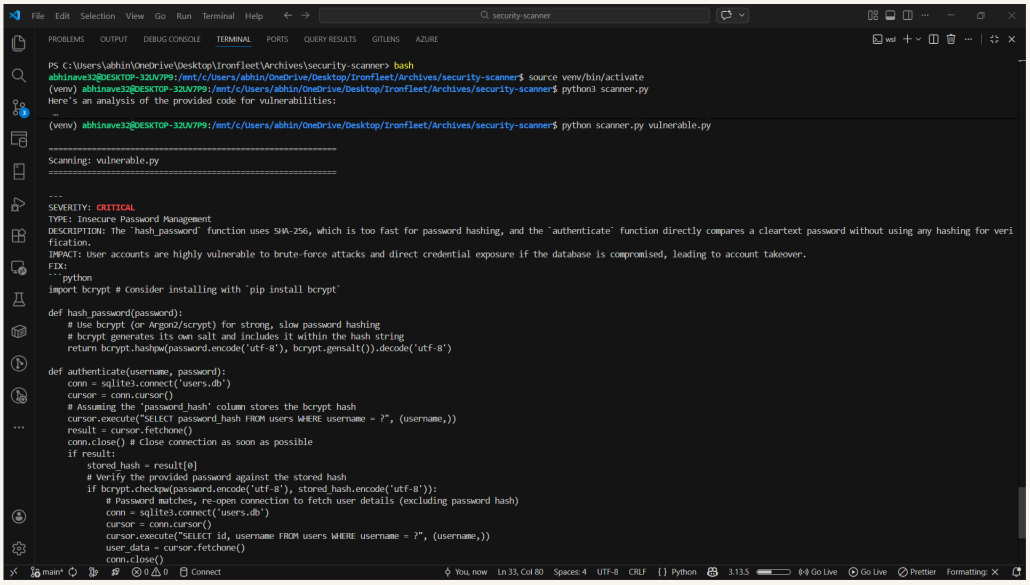


I built an AI-powered static security analysis tool that analyzes code using Gemini, enforces structured vulnerability reporting, and visually prioritizes issues using color-coded severity levels. This makes security review faster and more developer-friendly.

Abhinave P.B
NextWork Student

Scanning Real Python Files

In this secret mission, I'm adding file reading capability and command-line argument support to my security scanner. Professional tools do this because security scanners need to integrate into real developer workflows. Tools like Bandit, Flake8, and Semgrep don't require you to paste code — they scan actual files and entire projects from the command line. Supporting file paths makes the tool: Practical for real-world usage Scriptable and automation-friendly CI/CD ready Scalable to larger codebases



Abhinave P.B
NextWork Student

I scanned vulnerable.py by running: python scanner.py vulnerable.py The vulnerabilities detected were CRITICAL Insecure Password Management, including use of fast SHA-256 hashing and improper password comparison, making the system vulnerable to brute-force attacks and credential exposure. The scan_file function works by reading the target file, sending its code to Gemini with a security prompt, and printing structured, color-coded vulnerability results in the terminal.

The place to learn & showcase your skills

Check out nextwork.org for more projects