Secura: Code Vulnerability Detection and Auto-Fix

Arhaan Khaku

Abstract

Secura is an LLM-powered tool that detects, classifies, and remediates code vulnerabilities, emphasizing usability and structured outputs. Built for evaluating the SecurityEval dataset, Secura leverages GPT-4o to offer fast and understandable vulnerability analysis.

1 Introduction

Code security remains a persistent concern in modern software development. Manual vulnerability detection is often time-consuming and error-prone. **Secura** introduces an automated approach powered by large language models (LLMs), capable of analyzing code, identifying known weaknesses (via CWE), and suggesting meaningful remediations.

2 System Overview

Secura is composed of two main components: a Flask-based backend and a React.js frontend. The core logic and intelligence reside in the backend.

Backend Pipeline

1. Receives a POST request containing a code snippet.
2. Constructs a structured prompt, injecting the code into a secure context.
3. Sends the prompt to OpenAI's GPT-4o via API.
4. Parses the LLM's response into structured JSON with:
    - CWE classification
    - Vulnerability description
    - Error location
    - Secure code fix
    - Explanation
5. Returns the JSON to the frontend for rendering.

ML-Driven Detection Logic

We treat code vulnerability detection as a classification + generation task. GPT-4o is used as a zero-shot classifier and generative fixer. The backend logic (`detector.py`) includes:

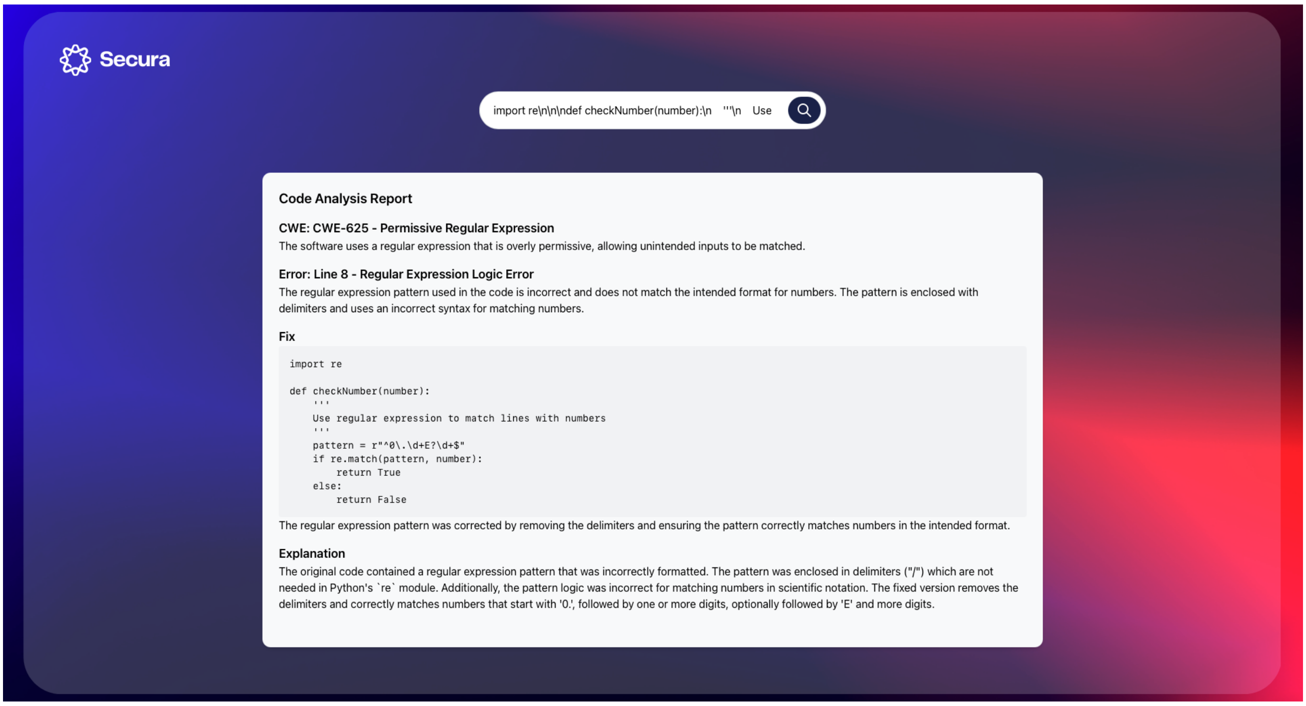
- A templated prompt combining code context + static analysis hints
- Safety guards to ensure minimal prompt injection
- Structured post-processing for predictable outputs

3 Output Example

Here is a sample JSON output:

```
{
  "cwe": {
    "id": "CWE-330",
    "title": "Use of Insufficiently Random Values"
  },
  "error": {
    "location": "Line 9",
    "attack_type": "Insecure Random"
  },
  "fix": {
    "code": "/* Secure fix here */"
  },
  "explanation": "Original code used a weak random generator. Fixed by
     switching to a secure RNG."
}
```

4 Demo



import re\n\n\ndef checkNumber(number):\n '''\n Use

Code Analysis Report

CWE: CWE-625 - Permissive Regular Expression

The software uses a regular expression that is overly permissive, allowing unintended inputs to be matched.

Error: Line 8 - Regular Expression Logic Error

The regular expression pattern used in the code is incorrect and does not match the intended format for numbers. The pattern is enclosed with delimiters and uses an incorrect syntax for matching numbers.

Fix

```
import re

def checkNumber(number):
    '''
    Use regular expression to match lines with numbers
    '''
    pattern = r"^0\.\d+E?\d+$"
    if re.match(pattern, number):
        return True
    else:
        return False
```

The regular expression pattern was corrected by removing the delimiters and ensuring the pattern correctly matches numbers in the intended format.

Explanation

The original code contained a regular expression pattern that was incorrectly formatted. The pattern was enclosed in delimiters ("/") which are not needed in Python's `re` module. Additionally, the pattern logic was incorrect for matching numbers in scientific notation. The fixed version removes the delimiters and correctly matches numbers that start with '0.', followed by one or more digits, optionally followed by 'E' and more digits.

Figure 1: LLM-generated vulnerability detection output

5 Future Enhancements

Planned improvements include:

- Fine-tuning on SecurityEval and custom vulnerability datasets for higher accuracy.
- Adding static analysis integration for hybrid rule + LLM detection.
- Model-switching logic for fallback to local LLMs when offline.
- GitHub integration for scanning PRs and suggesting inline patches.
- UI improvements for severity scoring, fix confidence, and CWE drill-downs.