Security Vulnerability Report

Generated: 2025-03-04 22:29:41.772835

Summary

- Total Vulnerabilities: 1
- Critical: 1
- High: 0
- Medium: 0
- Low: 0
- Info: 0
- Risk Score: 10.00

Detailed Vulnerabilities

SQL_INJECTION (CRITICAL)

- **Description:** The application is vulnerable to SQL injection because it directly concatenates user-provided input into an SQL query. An attacker can inject malicious SQL code by manipulating the 'id' parameter in the request. This can lead to unauthorized data access, modification, or deletion.
- **Impact:** An attacker can execute arbitrary SQL queries, potentially gaining access to sensitive data, modifying data, or even deleting data. In severe cases, the attacker could gain control of the database server.
- **Location:** main.py:9
- **CWE ID:** CWE-89
- **OWASP Category:** A03:2021 - Injection
- **CVSS Score:** 5.0
- **Remediation:** Use parameterized queries or prepared statements to prevent SQL injection. This ensures that user input is treated as data rather than executable code.

References:

- https://owasp.org/Top10/A03_2021-Injection/
- https://cwe.mitre.org/data/definitions/89.html

Proof of Concept:

```
Send a request like: /user?id=1 OR 1=1;--
```

Secure Code Example:

```
import sqlite3
from flask import Flask, request, jsonify

app = Flask(__name__)

@app.route('/user', methods=['GET'])
def get_user():
    user_id = request.args.get('id', '')
    conn = sqlite3.connect('test.db')
    cursor = conn.cursor()
    query = "SELECT * FROM users WHERE id = ?"
    try:
        cursor.execute(query, (user_id,))
        result = cursor.fetchone()
    except Exception as e:
        result = str(e)
    conn.close()
    return jsonify({'result': result})
```