AI Security Code Reviewer

Technical Security Analysis

Generated: 2025-06-22 22:48:58

Security Score: 3/100

Detailed Findings:

• Insecure Deserialization (Critical)

Line: N/A
Description: The application deserializes user-provided data without any validation
Explanation: Insecure deserialization can lead to remote code execution if an attacker provides a serialized object that includes a malicious payload.
Fix: Avoid deserializing user-provided data when possible. If it's necessary, use safe deserialization mechanisms and implement proper input validation.
CWE: N/A

• Broken Authentication (High)

Line: N/A
Description: The application uses a hardcoded username and password for authentication
Explanation: Hardcoded credentials can be easily discovered and exploited by attackers, leading to unauthorized access.
Fix: Implement a proper authentication mechanism with hashed and salted passwords. Avoid using hardcoded credentials.
CWE: N/A

• SQL Injection (High)

Line: N/A
Description: The application constructs SQL queries using string concatenation with user-provided data
Explanation: This allows an attacker to manipulate the SQL query by providing specially crafted input, leading to unauthorized data access or modification.
Fix: Use parameterized queries or prepared statements to prevent SQL injection.
CWE: N/A

Code Analysis:

```
import sqlite3

import os
import pickle # [Critical] A08: Insecure deserialization
from flask import Flask, request

app = Flask(__name__)

@app.route("/login", methods=["POST"])
def login():
# [Critical] A07: Broken authentication (hardcoded password)
if request.form['user'] == "admin" and request.form['pass'] == "1234":
return "Logged in"

return "Access denied"

@app.route("/search")
def search():
query = request.args.get("q")
conn = sqlite3.connect("db.sqlite")
cursor = conn.cursor()
# [Critical] A03: SQL Injection vulnerability
cursor.execute(f"SELECT * FROM users WHERE name = '{query}'")
return str(cursor.fetchall())

@app.route("/load")
def load():
# [Critical] A08: Insecure deserialization
data = request.args.get("data")
obj = pickle.loads(bytes.fromhex(data))
return f"Loaded: {obj}"
```